GIÁO TRÌNH

TRÍ TUỆ NHÂN TẠO

Khoa Tin học

Trần Uyên Trang

MỤC LỤC

LỜI NÓI ĐẦU

CHƯƠNG I:

TỔNG QUAN VỀ KHOA HỌC TRÍ TUỆ NHÂN TẠO

I. LỊCH SỬ PHÁT TRIỂN CỦA KHOA HỌC TRÍ TUỆ NHÂN TẠO

Năm 1950, Alan Turing – nhà toán học người Anh đã công bố công trình nghiên cứu khoa học của ông "Tính toán một cách máy móc và một cách thông minh" (Computing Machinery and Intelligence). Ông đã đưa ra trò chơi "Turing Test" như là một cách để nhận dạng máy tính thông minh. Trong trò chơi này một hoặc nhiều người có thể đặt các câu hỏi về bất kỳ lĩnh vực nào cho hai đối tượng dấu mặt: một người và một máy tính. Người đặt câu hỏi sẽ dựa vào câu trả lời để xác định đối tượng trả lời là người hay máy. Nếu có thể liên tục làm cho người phỏng vấn nghĩ rằng các câu trả lời là của con người thì máy tính đó được xem là thông minh. Đó là mốc lịch sử được công nhận là thời điểm bắt đầu phát triển của lĩnh vực khoa học Trí tuệ nhân tạo.

Từ đó một loạt những chương trình ra đời, một trong những chương trình ứng dụng to lớn nhất của những năm 50 là chương trình trò chơi cờ vua (của Arthur Samuel).

Hai ngôn ngữ lập trình thông minh trong lĩnh vực này cũng đươc phát triển vào những năm 50. Đầu tiên là IPL được Newell, Simon, và Shaw đưa ra trong quá trình thiết kế "Lý luận logic" (Logic Theorist). IPL là ngôn ngữ xử lý danh sách và sau này được thay thế bởi một ngôn ngữ được nhiều người biết đến là ngôn ngữ LISP. LISP được John McCarthy, một trong những người tiên phong của lĩnh vực trí tuệ nhân tạo đưa ra tại phòng thí nghiệm MIT vào cuối những năm 50 và được xem như là ngôn ngữ được chọn lựa cho những ứng dụng của trí tuệ nhân tạo.

Thập niên 60 được xem như là thời kỳ thịnh vượng nhất của trí tuệ nhân tạo. Một loạt những chương trình thông minh được xây dựng:

- Năm 1961 chương trình tính tích phân bất định
- Năm 1963 các chương trình heuristics
- Năm 1964 chương trình giải phương trình đại số sơ cấp
- Năm 1966 chương trình phân tích và tổng hợp tiếng nói
- Năm 1968 chương trình điều khiển người máy (robot) theo đồ án "Mắt-Tay", chương trình học nói.
- Năm 1972, Alain Calmerauer đưa ra ngôn ngữ lập trình Prolog
- Năm 1981, dự án của Nhật Bản xây dựng các máy tính thế hệ 5, lấy ngôn ngữ Prolog làm ngôn ngữ cơ sở.

Trong những năm 1990, có nhiều sản phẩm dân dụng được chế tạo sử dụng kỹ thuật trí tuệ nhân tạo mà cụ thể là máy giặt, máy ảnh, các hệ thống nhận dạng, xử lý ảnh, xử lý tiếng nói…

II. TRÍ TUỆ NHÂN TẠO LÀ GÌ ?

Trí tuệ nhân tạo là lĩnh vực khoa học chuyên nghiên cứu các phương pháp để xây dựng trí tuệ cho máy giống như trí tuệ con người.

Trí tuệ con người là gì ?

Trí tuệ con người là khả năng giải quyết vấn đề của người đó, khả năng này thường bao gồm bốn thao tác cơ bản :

1. Xác định các trạng thái đích của bài toán :

Xét quá trình suy nghĩ giúp con người giải một bài toán. Quá trình này phải bắt đầu từ một điểm (trạng thái ban đầu) và kết thúc tại một điểm (trạng thái đích). Giữa hai trạng thái của quá trình suy nghĩ này có thể được phân ra nhiều mảnh nhỏ suy nghĩ trong đó mỗi mảnh nhỏ suy nghĩ này có thể giúp con người đạt đến một mục đích nào đó có liên quan đến lời giải của bài toán. Mỗi mảnh nhỏ như vậy được xem như một trạng thái đích từng phần hay còn gọi là lời giải từng phần của bài toán và tập các mảnh nhỏ suy nghĩ được xem như tập các trạng thái đích từng phần mà con người đã định hướng để đạt đến trạng thái đích cuối cùng hay còn gọi là lời giải của bài toán.

2. Thu thập các sự kiện và các luật cho bài toán :

Sự thông minh của mỗi con người tuỳ thuộc vào người đó có khả năng sử dụng khối tri thức có sẵn trong mỗi người để đối phó với bất kỳ tình huống nào và liên tục học từ những kinh nghiệm mới để có khả năng đáp ứng với các tình huống tương tự trong tương lai. Vấn đề thông minh được xem xét đó là thu thập các sự kiện và sử dụng các sự kiện này để đạt đến các mục đích của bài toán. Công việc này được làm xong bằng cách công thức hoá tập các luật có quan hệ đến tất cả các sự kiện được lưu trữ trong bộ óc.

VD : Sự kiện và luật được thu thập để công thức hoá như sau :

Sự kiện 1 : lò đang đốt thì rất nóng

Luật 1 : *nếu tôi đặt bàn tay lên lò đang đốt thì nó sẽ bị bỏng*

Sự kiện 2 : Mùa đông vào buổi tối nhiệt độ xuống rất thấp

Luật 2 : *nếu tôi đi ra phố vào buổi tối mùa đông khi nhiệt độ xuống thấp mà không mặc áo ấm thì sẽ bị cảm lạnh*

3. Cơ chế thu gọn của bài toán :

Cơ chế thu gọn loại bỏ các đường suy nghĩ không có liên quan đến mục tiêu tức thời, chỉ tập trung đến đường suy nghĩ có khả năng đạt đến đích của bài toán.

4. Cơ chế suy diễn của bài toán : là nơi cho phép ta giải quyết vấn đề tức thời của bài toán và đồng thời thu thập tri thức mới cho bài toán.

VD : Sự kiện 1 : Ba mẹ của Nam là Lâm và Uyên

Sự kiện 2 : Ba mẹ của Trân là Lâm và Uyên

Hãy xác định quan hệ giữa Nam và Trân

Luật được công thức hoá để giải quyết vấn đề tức thời đó là : *Nếu một người nam và một người nữ có cùng ba mẹ thì họ là anh em hoặc chị em* .

Dựa vào luật này ta có thể đi đến kết luận : Quan hệ giữa Nam và Trân là quan hệ giữa anh em hoặc chị em

Vậy, phần thông minh ở đây giúp ta giải quyết vấn đề tức thời và đồng thời cho ta một sự kiện mới về bài toán được gọi là cơ chế suy diễn. Cơ chế này giúp chúng ta có khả năng học từ kinh nghiệm vì nó có khả năng cho phép ta phát sinh ra các sự kiện mới từ các sự kiện sẵn có. Các sự kiện mới này lại được ứng dụng trong các tình huống mới để phát sinh ra các sự kiện mới hơn cho bài toán.

Trí tuệ máy là gì?

Trí tuệ máy là khả năng giải quyết vấn đề của máy. Người ta muốn xây dựng trí tuệ máy giống như trí tuệ con người sao cho nó có khả năng giải quyết các vấn đề như sau :

Khả năng học

Khả năng mô phỏng các hành vi sáng tạo của con người
Khả năng trừu tượng hoá, tổng quát hoá và suy diễn
Khả năng tự giải thích hành vi
Khả năng thích nghi với tình huống mới (thu nạp tri thức và dữ liệu)
Khả năng xử lý các biểu diễn hình thức (ký hiệu tượng trưng, danh sách)
Khả năng sử dụng các tri thức và thông tin heuristics
Khả năng xử lý các thông tin không đầy đủ

III. NHỮNG ỨNG DỤNG TRONG LĨNH VỰC TRÍ TUỆ NHÂN TẠO VÀ PHẠM VI NGHIÊN CỨU

Phạm vi nghiên cứu :

Mục tiêu nghiên cứu để phát triển những kỹ thuật trí tuệ nhân tạo có thể nói trong phạm vi như sau :

- Tìm kiếm không gian lời giải của bài toán
- Thu thập tri thức từ con người
- Biểu diễn tri thức bằng các quy luật và các quan hệ
- Suy diễn ra những quy luật mới và những quan hệ mới
- Thích nghi với tri thức mới (là vấn đề học)
- Nhận dạng mẫu
- Mô hình định tính
- Các hệ cơ sở tri thức (dành cho các hệ chuyên gia)
- Lô gích mờ (xử lý thông tin không chắc chắn)
- Mạng neuron nhân tạo (cung cấp các phương pháp mới về việc suy diễn các mối quan hệ, việc học và việc nhận dạng mẫu)
- Giải thuật lan truyền (genetic algorithms) cung cấp các phương pháp mới và nhanh của việc tìm kiếm không gian lời giải của bài toán.

Ứng dụng trong lĩnh vực trí tuệ nhân tạo :

Những ứng dụng sớm nhất của lĩnh vực trí tuệ nhân tạo gồm :

- Trò chơi
- Chứng minh định lý
- Giải quyết các vấn đề tổng quát
- Cảm nhận : nhìn và nói
- Hiểu được ngôn ngữ tự nhiên
- Giải quyết các vấn đề chuyên gia gồm :
  + Phân tích hoá chất
  + Thiết kế kỹ thuật
  + Các ký hiệu toán học
  + Chẩn đoán y khoa

Một số ứng dụng trí tuệ nhân tạo được thể hiện cụ thể hoá trong các ngành kỹ thuật như lĩnh vực điều khiển và hệ thống điện gồm các ứng dụng sau :

- Phân tích và bảo vệ hệ thống năng lượng điện dựa trên việc xây dựng các quy luật điều khiển và cập nhật thu thập dữ liệu
- Các hệ điều khiển xử lý thông tin không chắc chắn và môi trường có nhiễu ứng dụng logic mờ
- Điều khiển không lưu để phát hiện sự cố và tránh sự va chạm sử dụng hệ cơ sở tri thức
- Trong điều khiển đường sắt để điều khiển tàu dừng tự động sử dụng hệ cơ sở tri thức mờ

- Lĩnh vực giao thông đường thuỷ : điều khiển tàu, cung cấp thông tin cho các bến cảng và tàu tránh sự va chạm
- Lĩnh vực giao thông đường bộ : thiết kế và bảo quản xe cộ nhờ hệ chuyên gia, vận hành xe cộ, phân tích và kiểm soát tai nạn giao thông, quản lý an toàn ra quyết định nhờ các hệ cơ sở tri thức, phương án phục vụ vận chuyển hành khách nhờ các hệ cơ sở tri thức sắp xếp, dự báo các cuộc hành trình, lý thuyết lô gích mờ được sử dụng để xử lý thông tin không chắc chắn
- Mạng neuron nhân tạo sử dụng các hệ thống điều khiển để nhận dạng, dự báo và điều khiển.

CHƯƠNG II:

CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ CƠ BẢN

I. VAI TRÒ CỦA TÌM KIẾM TRONG CÁC LĨNH VỰC CỦA TRÍ TUỆ NHÂN TẠO

1. Giới thiệu :

Tìm kiếm đóng vai trò chủ đạo trong phần lớn các khái niệm liên quan đến trí tuệ nhân tạo. Giải thuật này cung cấp một bộ khung có tính chất khái niệm của hầu hết mọi phương pháp tiếp cận đến sự khám phá có tính hệ thống của những sự chọn lựa.

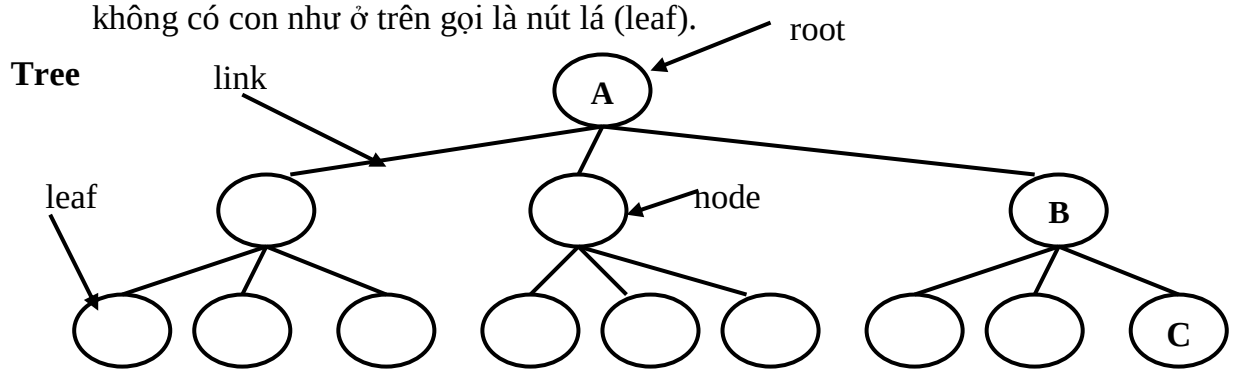
Chúng ta sẽ bắt đầu với một vài yếu tố nền tảng, thuật ngữ, và các chiến lược thực thi cơ bản sau đó sẽ tìm hiểu bốn nhóm giải thuật tìm kiếm khác nhau về hai khía cạnh:

- Sự khác nhau giữa tìm kiếm thông tin không đầy đủ (uninformed search hay blind search) và tìm kiếm thông tin đầy đủ (informed search hay heuristic search). Trong đó informed search sẽ truy xuất đến những thông tin mang tính chất tác vụ chuyên biệt mà có thể được sử dụng nhằm mục đích làm cho tiến trình tìm kiếm hữu hiệu hơn.
- Sự khác nhau giữa phương pháp tìm kiếm theo một đường bất kỳ (any-path search) và tìm kiếm tối ưu (optimal search). Optimal search tìm kiếm một đường tốt nhất có thể trong khi any-path search chỉ giải quyết cho việc tìm kiếm đối với một vài trường hợp.

2. Những thuật ngữ thông dụng trên cây và đồ thị tìm kiếm

Những phương pháp tìm kiếm mà chúng ta sẽ gặp được định nghĩa trên cây (tree) và đồ thị (graph), nên chúng ta phải nhắc lại một số thuật ngữ cần cho những cấu trúc này.

- Cây được tạo từ những hình tròn và đường thẳng gọi là nút (node) và đường nối (link) được kết nối với nhau sao cho không tạo thành vòng khép kín. Nút thình thoảng được hiểu như là những đỉnh và đường nối là đường biên. (Điều này thường gặp phổ biến khi xét một đồ thị)
- Một cây có nút gốc (root node) tại vị trí khởi đầu của cây. Mỗi nút ngoại trừ nút gốc có một nút cha (parent) (nút ngay trước nó, ở mức cao hơn nó)
- Mỗi nút ngoại trừ nút cuối cùng (ở xa nút gốc nhất) đối với mỗi nhánh,đều có một nút được nối tiếp sau nó (ở mức thấp hơn nó) gọi là nút con (children). Nút không có con như ở trên gọi là nút lá (leaf).



Hình 2.1

B là cha (parent) của C

C là con (child) của B

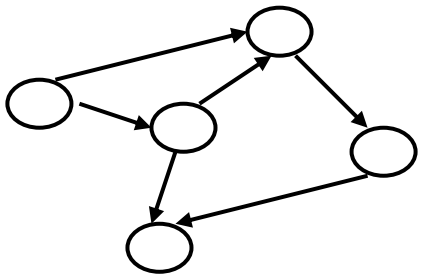
A là ông (ancestor) của C

C là cháu (descendant) của A

- Đồ thị cũng là tập những nút được nối với nhau bằng các đường nối (link) và cho phép tạo thành vòng. Bên cạnh đó, một nút (node) có thể có nhiều cha (parent).
- Chúng ta có hai loại đồ thị:

  + Đồ thị có hướng (directed graph): các đường nối có hướng (gần giống như những đường một chiều)

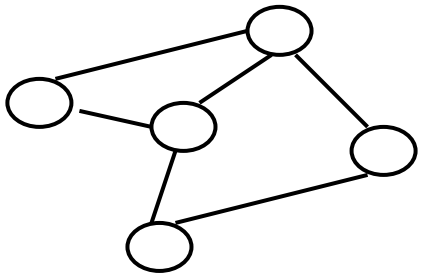
Directed graph



Hình 2.2

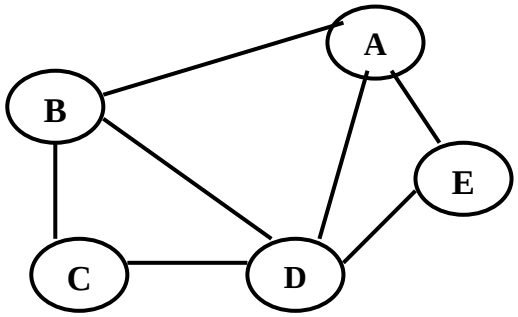
+ Đồ thị vô hướng (undirected graph): các đường nối không xác định hướng (có thể đi theo cả hai hướng)

Undirected graph



Hình 2.3

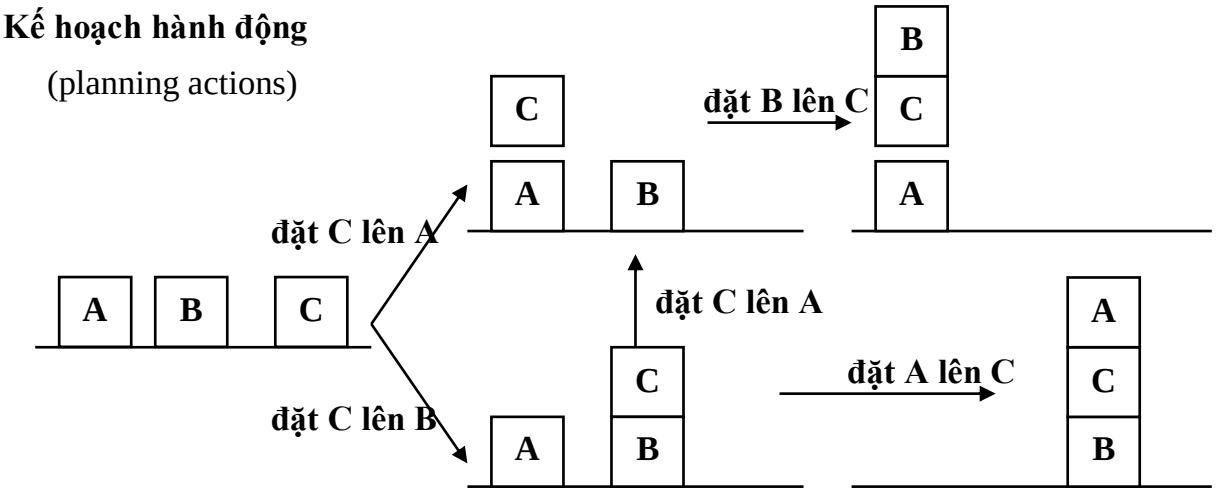
Vd 6.1: Xét mạng lưới đường giao thông hoặc lộ trình chuyến bay hoặc mạng máy tính. Điều chúng ta quan tâm ở đây trong tất cả mọi trường hợp là tìm kiếm một đường đi trên đồ thị thoả mãn một vài yêu cầu nào đó. Chẳng hạn như chúng ta có thể tìm kiếm một đường bất kỳ nào đó mà có số chặng là ít nhất



Lộ trình chuyến bay

Hình 2.4

Tuy nhiên đồ thị còn có thể trừu tượng hơn. Xét một đồ thị được định nghĩa như sau:



Hình 2.5 : Đồ thị biểu thị những trạng thái có thể của thế giới khối

- Các nút biểu thị sự mô tả trạng thái của thế giới khối, chẳng hạn một khối có thể ở trên đỉnh của một khối khác. Và ở đây các đường nối sẽ đại diện cho những hành động thay đổi từ trạng thái này sang trạng thái khác
- Một đường xuyên suốt đồ thị (từ nút khởi đầu đến nút đích) được gọi là "một kế hoạch hành động (plan of action)" để đạt được được một vài trạng thái đích mong đợi từ một vài trạng thái khởi đầu đã biết
- Đồ thị dạng này thường gặp rất nhiều trong AI.

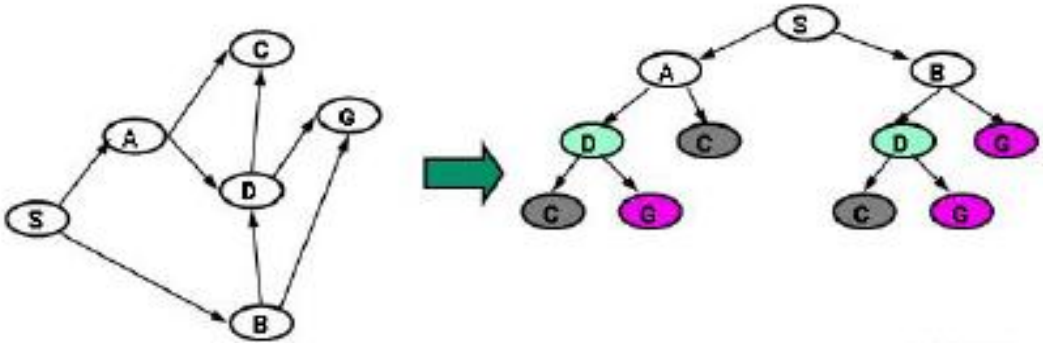
3. Tìm kiếm trên cây và tìm kiếm trên đồ thị

Cây là một lớp con của đồ thị có hướng mặc dù đường kết nối giữa các nút trong cây không có mũi tên định hướng. Các kết nối trong cây không tạo thành vòng và mỗi nút (ngoại trừ gốc) có một cha.

Khi được yêu cầu tìm kiếm trên một đồ thị chúng ta có thể chuyển đổi sang tìm kiếm tương đương trên cây thông qua 2 bước sau:

- Chuyển kết nối vô hướng thành hai kết nối có hướng
- Tránh tạo thành vòng, hay tốt hơn là không được thăm một nút hai lần.

Chúng ta có thể xem xét một ví dụ chuyển đổi một đồ thị thành một cây. Giả sử ở đây, S là khởi đầu của quá trình tìm kiếm và từ đó chúng ta cố gắng để tìm ra một đường đến G thì chúng ta sẽ đi xuyên suốt đồ thị và tạo ra những kết nối từ mỗi nút đến mỗi nút được kết nối sao cho không tạo thành vòng và ngừng bất cứ khi nào chúng ta tìm được G. Lưu ý rằng mỗi cây như vậy có một nút lá cho mỗi đường không có vòng lặp trên đồ thị khởi đầu từ S.



Hình 2.6

Tuy nhiên, cũng cần lưu ý rằng, mặc dù chúng ta tránh được những vòng lặp nhưng một vài nút (được minh hoạ có cùng màu trong hình) lại được gặp lại hai lần trên cây. Nói rõ hơn, những nút trùng nhau này nằm ở những đường không tạo thành vòng lặp khác nhau. Điều này có nghĩa là một tiến trình tìm kiếm hoàn chỉnh trên cây này có thể sẽ có một số công đoạn thừa.

Vấn đề của việc phải nỗ lực như thế nào để tránh được vòng lặp và tránh được thăm viếng thừa đến một số nút là một vấn đề quan trọng mà chúng ta sẽ xem xét lại sau này khi chúng ta thảo luận đến những thuật toán tìm kiếm khác nhau.

4. Phân loại các giải thuật tìm kiếm

Có nhiều loại thuật toán tìm kiếm. Chúng ta sẽ gặp một loạt các thuật toán tìm kiếm từ đơn giản nhưng ít hiệu quả cho đến thuật toán tối ưu nhất nhưng lại phức tạp theo bảng sau:

Loại	Tên giải thuật	Phương thức hoạt động
Any Path Uninformed	**Depth-First Breadth-First**	Tìm kiếm một cách hệ thống trên toàn bộ cây cho đến khi nút đích được tìm thấy
Any Path Informed	**Best-First**	Sử dụng phương pháp đo lường (heuristic) cụ thể phần tốt nhất của một trạng thái để đạt đến đích nhanh nhất hoặc tìm thấy trạng thái đích mong đợi
Optimal Uninformed	**Uniform-Cost**	Sử dụng phương pháp đo chiều dài của đường (path length), đảm bảo tìm ra đường ngắn nhất
Optimal Informed	**A***	Sử dụng phương pháp đo chiều dài đường và khai thác thông tin heuristic đảm bảo tìm ra đường ngắn nhất nhưng nhanh hơn so với phương pháp uninformed

II. ĐỊNH NGHĨA KHÔNG GIAN CỦA BÀI TOÁN

Hai thành phần cơ bản của lĩnh vực trí tuệ nhân tạo đó là biểu diễn tri thức và tìm kiếm tri thức trong miền đã được biểu diễn.

Tri thức của một bài toán có thể được phân ra làm 3 loại: tri thức mô tả, tri thức thủ tục và tri thức điều khiển.

- Tri thức mô tả: để mô tả các sự kiện, các quan hệ giữa các sự kiện, đối tượng, các tính chất của bài toán.
- Tri thức thủ tục: là các thủ tục để giải bài toán được thể hiện bằng các luật dưới dạng if-then
- Tri thức điều khiển: là loại tri thức heuristics có khả năng thực hiện hai chức năng đó là chọn luật thích hợp để đưa ra ứng dụng và thu gọn không gian tìm kiếm của bài toán.

Tập các luật sẽ giúp định nghĩa không gian của bài toán hay còn gọi là không gian trạng thái tìm kiếm của bài toán. Không gian của bài toán được biểu diễn bằng đồ thị sẽ là công cụ giúp người lập trình có khả năng phân tích và dự báo các đặc thù của bài toán cụ thể như khả năng phân rã bài toán mẹ ra nhiều bài toán con, chọn chiến lược và giải thuật thích hợp với các đặc thù của bài toán, đường dẫn đến lời giải của bài toán phải được đảm bảo tối ưu…

Không gian trạng thái tìm kiếm của bài toán được định nghĩa sử dụng lý thuyết đồ thị như sau: không gian trạng thái tìm kiếm của bài toán được biểu diễn bằng đồ thị gồm tập bốn thành phần [N, A, S, G] trong đó:

- N: tập các đỉnh hay các trạng thái của đồ thị
- A: tập các cung hay các liên kết giữa các đỉnh. Liên kết này tương ứng với các bước trong quá trình giải quyết bài toán
- S: tập con của N chứa các trạng thái ban đầu của bài toán
- G: tập con của N chứa các trạng thái đích của bài toán

Đường đi đến lời giải của bài toán đó là đường thông qua đồ thị bắt đầu từ một đỉnh trong S đến một đỉnh trong G.

Vd 2.1: Cho hai bình đựng chất lỏng, một bình có dung tích 4 lít và một bình có dung tích 3 lít. Cả hai bình không có dấu dung tích. Có thể dùng một đường ống để làm đầy nước ở hai bình. Làm thế nào để có chính xác 2 lít nước trong bình 4 lít. Hãy biểu diễn không gian của bài toán bằng đồ thị?

Không gian của bài toán có thể được mô tả bằng các cặp số nguyên (x, y), trong đó x = 0, 1, 2, 3, 4 biểu diễn số lít nước trong bình 4 lít và y = 0, 1, 2, 3 biểu diễn số lít nước trong bình 3 lít. Trạng thái ban đầu của bài toán là hai bình đều rỗng, do đó ta có cặp số nguyên (0, 0). Trạng thái đích của bài toán là có 2 lít nước trong bình 4 lít, do đó ta sẽ có (2, n), với n là giá trị bấtkỳ từ 0->3.

Không gian của bài toán được định nghĩa bằng tri thức thủ tục của bài toán đó chính là các luật để giải bài toán được thiết kế như sau:

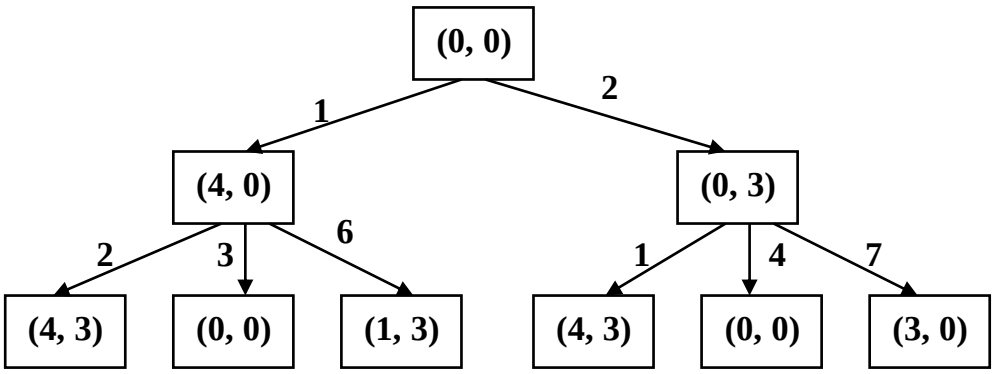
- Luật 1: Nếu x < 4 thì làm đầy bình 4 lít: $(x, y / x < 4) \rightarrow (4, y)$
- Luật 2: Nếu y < 3 thì làm đầy bình 3 lít: $(x, y / y < 3) \rightarrow (x, 3)$
- Luật 3: Nếu x > 0 thì làm rỗng bình 4 lít: $(x, y / x > 0) \rightarrow (0, y)$
- Luật 4: Nếu y > 0 thì làm rỗng bình 3 lít: $(x, y / y > 0) \rightarrow (x, 0)$
- Luật 5: Nếu x + y >=4 và y > 0 thì đưa nước từ bình 3 lít sang bình 4 lít cho đến khi bình 4 lít đầy: $(x, y / x + y >= 4 \wedge y > 0) \rightarrow (4, y – (4 – x))$
- Luật 6: Nếu x + y >=3 và x > 0 thì đưa nước từ bình 4 lít sang bình 3 lít cho đến khi bình 3 lít đầy: $(x, y / x + y >= 3 \wedge x > 0) \rightarrow (x – (3 – y), 3)$

- Luật 7: Nếu x + y <=4 và y > 0 thì đưa tất cả nước từ bình 3 lít sang bình 4 lít: (x, y / x + y <=4 ^ y > 0) → (x + y, 0)
- Luật 8: Nếu x + y <=3 và x > 0 thì đưa tất cả nước từ bình 4 lít sang bình 3 lít: (x, y / x + y <=3 ^ x > 0) → (0, x + y)

Từ trạng thái ban đầu của bài toán là (0, 0) thoả mãn hai luật 1 và 2 phát sinh 2 trạng thái mới (4, 0) và (0, 3).

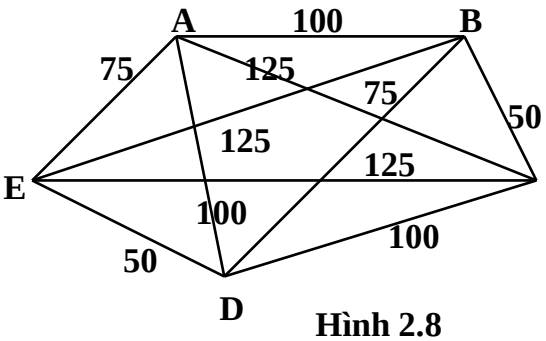
Tại trạng thái mới (4, 0) thoả mãn ba luật 2, 3, 6 phát sinh ra 3 trạng thái mới hơn là (4, 3), (0, 0) và (1, 3).

Tại trạng thái mới (0, 3) cũng thoả mãn ba luật 1, 4, 7 phát sinh ra 3 trạng thái mới hơn là (4, 3), (0, 0) và (3, 0). Quá trình phát sinh như thế cứ tiếp diễn cho đến khi có một trạng thái bất kỳ (2, n) xuất hiện thì dừng. Số trạng thái được phát sinh kể cả trạng thái ban đầu và trạng thái đích được gọi là không gian của bài toán.

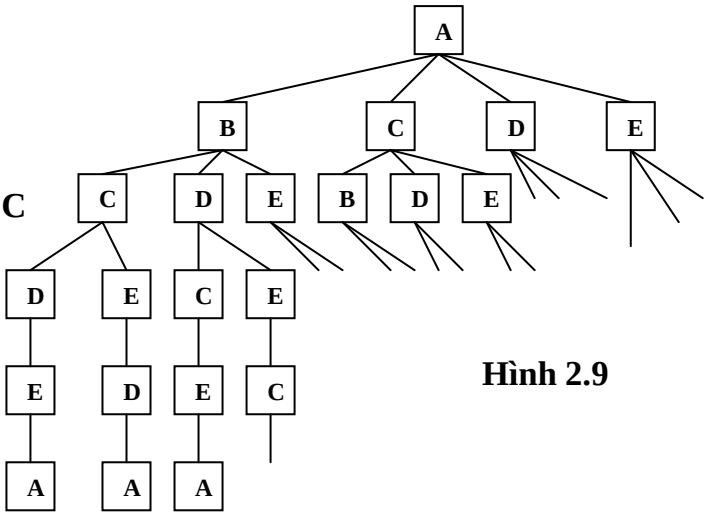


Hình 2.7 *Một phần không gian trạng thái của bài toán bình đựng nước được biểu diễn bằng đồ thị*

Vd 2.2: Xét bài toán hành trình người bán hàng.Giả sử người bán hàng có năm thành phố cần đến giao hàng và sau đó phải trở về nhà. Mục đích của bài toán là tìm đường đi ngắn nhất cho cuộc hành trình người bán hàng để đi đến tất cả các thành phố, mỗi thành phố ông ta chỉ đến một lần và sau đó trở về lại thành phố bắt đầu cuộc hành trình



Hình 2.8



Hình 2.9

Hình 2.8 là một ví dụ cụ thể của bài toán trên. Hãy biểu diễn không gian trạng thái của bài toán

Giả sử cuộc hành trình của người bán hàng bắt đầu từ thành phố A và trở về lại A. Không gian của bài toán này là số đường đi khác nhau, trong đó sẽ có một đường đi ngắn nhất cho cuộc hành trình.

Nếu cuộc hành trình đi qua n thành phố, ta sẽ có số (n-1)! đường đi khác nhau. Hình 2.9 mô tả một phần không gian trạng thái của bài toán.

III. CÁC CHIẾN LƯỢC CHO KHÔNG GIAN TRẠNG THÁI TÌM KIẾM

1. Tìm kiếm hướng dữ liệu và hướng đích

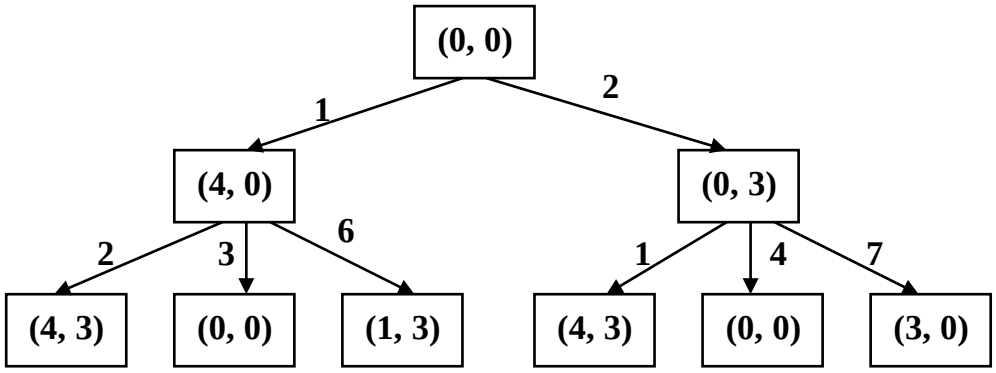
Một không gian trạng thái có thể được tìm kiếm trong hai hướng : từ dữ liệu được cho của bài toán tiến đến đích hoặc từ đích lùi về dữ liệu.

Trong hướng tìm kiếm từ dữ liệu còn được gọi là chuỗi suy diễn tiến, người giải bài toán bắt đầu với các sự kiện được cho của bài toán và tập các luật để thay đổi trạng thái. Diễn biến tìm kiếm bằng cách ứng dụng các luật với các vế bên trái của chúng thoả mãn các sự kiện để sản xuất ra các sự kiện mới, cách như vậy được sử dụng cho các luật để phát sinh ra các sự kiện mới hơn. Quá trình này tiếp tục cho đến khi ta hy vọng nó phát sinh ra một đường mà đường đó thoả mãn điều kiện đích.

Trong hướng tìm kiếm từ đích lùi về dữ liệu còn được gọi là chuỗi suy diễn lùi, người giải bài toán bắt đầu từ sự kiện đích được cho của bài toán và tập các luật để thay đổi trạng thái. Diễn biến tìm kiếm bằng cách chọn tất cả các luật mà các vế bên phải của chúng đã phát sinh ra sự kiện đích và xác định các sự kiện ở các vế bên trái của các luật cho phép phát sinh ra các đỉnh đích này. Các sự kiện này trở thành các đích mới cho công việc tìm kiếm. Tìm kiếm tiếp tục cho đến khi các sự kiện ban đầu của bài toán được tìm thấy.

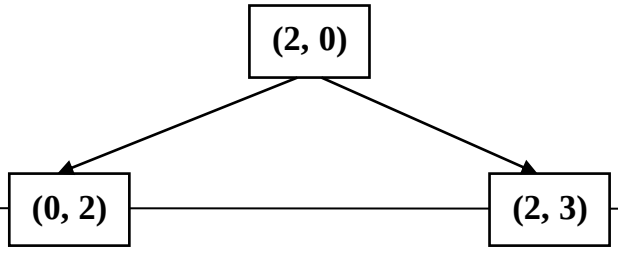
Vd 2.3: Xét bài toán bình đựng nước. Có hai cách để giải bài toán :

- Tìm kiếm từ dữ liệu đến đích, minh hoạ ở hình 2.10
- Tìm kiếm từ đích lùi về dữ liệu, minh hoạ ở hình 2.11



Hình 2.10

Hình 2.10 hướng tìm kiếm bắt đầu từ dữ liệu ban đầu (0, 0). Ứng dụng các luật 1, 2 đến trạng thái này để sản xuất ra các trạng thái mới (4, 0) và (0, 3). Tìm kiếm tiếp tục để phát sinh ra các trạng thái mới hơn cho đến khi có một đỉnh đích (2, n) được tìm thấy thì dừng.



Hình 2.11

Hình 2.11, hướng tìm kiếm bắt đầu từ đích (2, 0). Chọn tất cả các luật mà vế bên phải của chúng có khả năng phát sinh ra đỉnh đích này, đó là luật 4, 7 và xác định các điều kiện thoả mãn các luật để phát sinh ra đỉnh đích này đó là (0, 2) và (2, 3). Các điều kiện này trở thành các đích mới cho công việc tìm kiếm. Tìm kiếm tiếp tục cho đến khi tìm thấy sự kiện ban đầu (0, 0) của bài toán xuất hiện thì dừng.

2. Giải thuật truyền lùi (back tracking)

Trong cách giải bài toán sử dụng hướng tìm kiếm đích hoặc dữ liệu, thông qua đồ thị không gian trạng thái chúng ta phải tìm được một đường từ trạng thái ban đầu đến trạng thái đích. Sự nối tiếp của các cung trong đường này tương ứng với thứ tự các bước của lời giải. Chúng ta phải xem xét nhiều đường khác nhau cho đến khi đích mong muốn được tìm thấy. Giải thuật truyền lùi là một công cụ cần thiết cho việc tìm kiếm này.

Tìm kiếm truyền lùi bắt đầu tại trạng thái ban đầu và diễn tiếp một đường cho đến khi nó đạt đến đích hay đường cụt.

- Nếu tìm thấy đích, dừng và thiết lập một đường đi đến lời giải.

- Nếu đến đường cụt, nó truyền lùi về đỉnh gần nhất trên đường chưa được duyệt qua và tiếp tục nhìn xuống một trong các nhánh của đỉnh này.

Giải thuật truyền lùi sử dụng 3 danh sách: SL, NSL và DE:

- SL: danh sách trạng thái, liệt kê các trạng thái trên đường đi hiện hành đang được duyệt qua. Nếu đích được tìm thấy, SL sẽ là danh sách chứa thứ tự của các trạng thái trên đường đi đến lời giải của bài toán.

- NSL: danh sách trạng thái mới, chứa các đỉnh đang chờ được duyệt qua, chẳng hạn các đỉnh chưa được phát sinh và chưa được tìm kiếm.

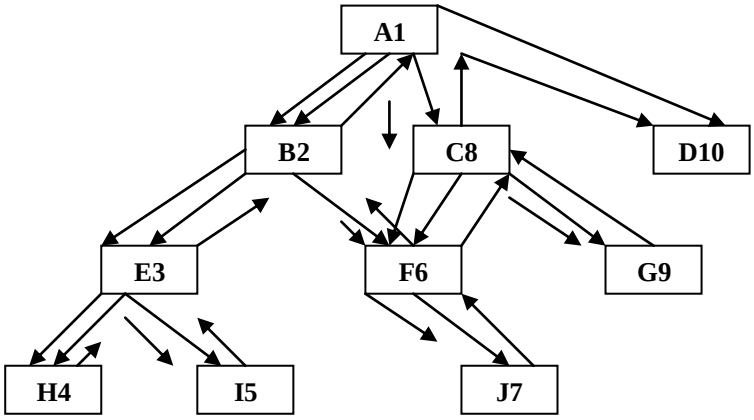
- DE: danh sách chứa các đỉnh của các đường cụt

Giải thuật được mô tả như sau:

```
function backtrack;
begin
    SL:= [Start]; NSL:= [Start]; DE = [ ]; CS:= Start;
    while NSL ≠ [ ] do
    begin
      if CS = goal then return (SL);
      if CS không có kế thừa (ngoại trừ các đỉnh đã có sẵn trên DE, SL, NSL)
            then begin
                while SL ≠ [ ] và CS = phần tử đầu tiên của SL do
                begin
```

```
                    cộng CS vào DE; // ghi nhận trạng thái như đỉnh cụt loại
                                    // bỏ phần tử đầu tiên từ SL
                                    // truyền lùi loại bỏ phần tử đầu tiên từ NSL
                    CS:= phần tử đầu tiên của NSL;
                end;
                    cộng CS vào SL;
            end;
            else begin
                    đặt các con của CS vào NSL (trừ các đỉnh đã có sẵn trên DE, SL,
NSL);
                    CS:= phần tử đầu tiên của NSL;
                    cộng CS vào SL
            end
    end;
    return FAIL;
end.
```

Vd 2.4 Cho không gian trạng thái của bài toán như hình 2.12 dưới đây. Hãy sử dụng giải thuật truyền lùi để xây dựng đường đi đến lời giải của bài toán đích là G bắt đầu từ trạng thái ban đầu A?



Hình 2.12

Với A, B, C, D, E, F, G, H, I, và J là tên các đỉnh, các chữ số 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 đánh dấu số thứ tự của các đỉnh được duyệt qua và các mũi tên có đường không liên tục chỉ hướng tìm kiếm trong không gian trạng thái của bài toán.

Sử dụng giải thuật truyền lùi với đồ thị biểu diễn không gian trạng thái trên, ta có kết quả theo bảng sau:

Đầu tiên: Gán SL = [A]; NSL = [A]; DE = [ ]; CS = A;

Vòng lặp	CS	SL	NSL	DE

0	A	[A]	[A]	[ ]
1	B	[BA]	[BCDA]	[ ]
2	E	[EBA]	[EFBCDA]	[ ]
3	H	[HEBA]	[HIEFBCDA]	[ ]
4	I	[IEBA]	[IEFBCDA]	[H]
5	F	[FBA]	[FBCDA]	[EIH]
6	J	[JFBA]	[JFBCDA]	[EIH]
7	C	[CA]	[CDA]	[BFJEIH]
8	G	[GCA]	[GCDA]	[BFJEIH]

Kết quả được biểu diễn trên đây sử dụng giải thuật truyền lùi với chiến lược suy diễn tiến, lấy đỉnh gốc của đồ thị làm trạng thái ban đầu và đánh giá con của nó để tìm kiếm đường đến đích.

Giải thuật cũng có thể sử dụng với chiến lược suy diễn lùi, lấy đỉnh gốc của đồ thị làm đích và đánh giá con của nó để tìm kiếm trạng thái ban đầu.

3. Giải thuật tìm kiếm đơn giản

Từ giải thuật tìm kiếm đơn giản này chúng ta sẽ hình thành các giải thuật tìm kiếm kế theo như DFS, BFS hay Best-First hay Uniform-Cost…

Với giải thuật tìm kiếm chung này, đầu tiên chúng ta sử dụng danh sách các nút tìm kiếm là open, sau đó nhặt một nút từ danh sách open, xem nó có là đích không hoặc mở rộng đường đi của nó đến các lân cận và đặt chúng vào danh sách open.

Lưu ý rằng chúng ta phải theo dõi các trạng thái mà chúng ta đạt đến (visited) và không được đặt chúng vào open nhiều hơn một lần. Điều này giúp chúng ta tránh được vòng lặp bởi vì chúng ta chỉ có thể đạt đến mỗi trạng thái một lần.

1. Khởi tạo open với nút tìm kiếm (S); set closed = (S)

2. If open là rỗng, fail. Else, nhặt một nút tìm kiếm (X) từ open

3. If state(X) là đích, return X (chúng ta đã đạt đến đích)

4. else remove X từ open

5. Tìm tất cả các children của state(X) không có trong closed (chưa được thăm) và tạo sự mở rộng một mức của X đến mỗi descendant

6. Thêm những đường đã mở rộng này vào open; đặt các children của state(X) vào closed

7. Quay lại bước 2

Từ đây chúng ta sẽ mở rộng cho từng giải thuật chuyên biệt.

- Chẳng hạn với DFS luôn nhìn vào nút sâu nhất trên cây tìm kiếm trước nên chúng ta sẽ có một số sửa đổi sau:
  - Nhặt thành phần đầu tiên của open làm nút kiểm tra và mở rộng (Bước 2)
  - Thêm đường đã mở rộng mới vào đầu danh sách open để đường tiếp theo được kiểm tra sẽ là một trong những mở rộng của đường hiện hành đến một trong những descendants của trạng thái của nút đó (Bước 6)
- Với BFS
  - Nhặt thành phần đầu tiên của open làm nút kiểm tra và mở rộng (Bước 2)
  - Thêm đường đã mở rộng mới vào cuối danh sách open. Vậy đường tiếp theo sẽ không nối đến một trong những descendants của nút hiện hành mà đến một nút ở cùng mức với nút đó trên cây (Bước 6)
- Với Best-First
  - Nhặt thành phần tốt nhất (được đo bằng giá trị heuristic của trạng thái) của open để kiểm tra và mở rộng (Bước 2)
  - Thêm đường đã mở rộng mới vào bất cứ chỗ nào trong open miễn là nó sẽ hữu hiệu hơn để giữ cho open theo trật tự sắp xếp dưới dạng nào đó nhằm giúp dễ dàng hơn để tìm ra thành phần tốt nhất. (Bước 6)

4. Giải thuật tìm kiếm theo chiều sâu và chiều rộng

(depth first search and breadth first search)

Hai loại giải thuật này giúp các chiến lược tìm kiếm (suy diễn tiến hoặc suy diễn lùi) xác định thứ tự các trạng thái đã được duyệt qua trong không gian trạng thái của bài toán.

Giải thuật tìm kiếm theo chiều sâu (depth first search)

- Trong giải thuật tìm kiếm theo chiều sâu (DFS), khi một trạng thái được duyệt qua, tất cả các con và cháu chắt của nó được duyệt qua trước khi duyệt qua bất kỳ anh em nào của nó.

- Tìm kiếm theo chiều sâu sẽ đi sâu dần trong không gian tìm kiếm mà nó có khả năng

- Khi nào không còn con cháu của một trạng thái được tìm thấy thì giải thuật sẽ chuyển sang duyệt qua các anh em của nó.

- Giải thuật tìm kiếm theo chiều sâu sử dụng hai danh sách open và closed để giữ đường của quá trình tìm kiếm thông qua không gian trạng thái.

+ Danh sách open chứa các đỉnh đang chờ duyệt qua

+ Danh sách closed chứa các đỉnh đã được duyệt qua

```
procedure depth_first_search
begin
    open:= [Start];
```

```
closed:= [ ];
while open ≠ [ ];
    begin
        Huỷ bỏ đỉnh đầu tiên từ danh sách open, đặt tên nó là X;
        if X là đích then return (success)
        else begin
                phát sinh các con của X;
                đặt X vào đầu danh sách closed;
                loại bỏ con của X đã có trên open hoặc closed;
                đặt các con còn lại của nó vào đầu ds open
                end
        end;
    return (failure)
end.
```

Giải thuật tìm kiếm theo chiều rộng (breadth first search)

- Ngược với giải thuật tìm kiếm theo chiều sâu, giải thuật tìm kiếm theo chiều rộng (BFS) sẽ thăm dò không gian của bài toán mức theo mức.

- Khi không còn trạng thái nào có thể được thăm dò trên mức, thì giải thuật di chuyển sang mức kế theo.

- Giải thuật tìm kiếm theo chiều rộng cũng sử dụng hai danh sách open và closed để giữ đường của quá trình tìm kiếm thông qua không gian trạng thái.

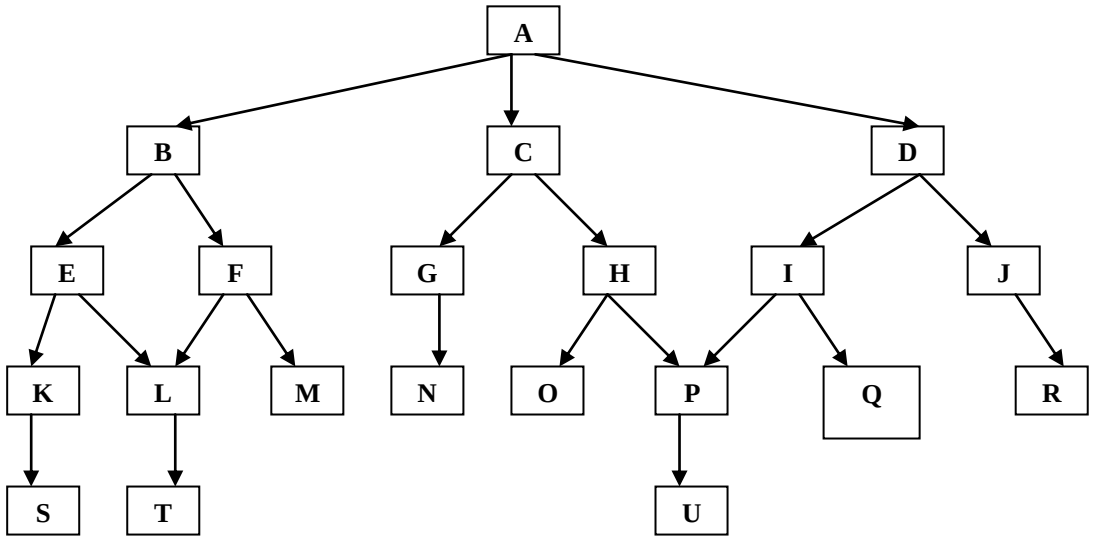
+ Danh sách open chứa các đỉnh đang chờ duyệt qua

+ Danh sách closed chứa các đỉnh đã được duyệt qua

```
procedure breadth_first_search
begin
    open:= [Start];
    closed:= [ ];
    while open ≠ [ ];
        begin
            Huỷ bỏ đỉnh đầu tiên từ danh sách open, đặt tên nó là X;
            if X là đích then return (success)
            else begin
                    phát sinh các con của X;
                    đặt X vào đầu danh sách closed;
                    loại bỏ con của X đã có trên open hoặc closed;
                    đặt các con còn lại của nó vào cuối ds open
```

```
                    end
                end;
            return (failure)
        end.
```

Vd 2.5 Cho không gian trạng thái của bài toán như sơ đồ dưới đây:



Hình 2.13

Giả sử U là trạng thái đích của bài toán. Hãy sử dụng giải thuật tìm kiếm theo chiều rộng để giải bài toán?

Sử dụng giải thuật tìm kiếm theo chiều rộng để tìm kiếm trạng thái đích U trên đồ thị không gian trạng thái của Hình 2.13 cho kết quả của các vòng lặp như sau:

1- open = [A]; closed = [ ]
2- open = [B, C, D]; closed = [A]
3- open = [C, D, E, F]; closed = [B, A]
4- open = [D, E, F, G, H]; closed = [C, B, A]
5- open = [E, F, G, H, I, J]; closed = [D, C, B, A]
6- open = [F, G, H, I, J, K, L]; closed = [E, D, C, B, A]
7- open = [G, H, I, J, K, L, M] (L đã có sẵn trên danh sách open);
   closed = [F, E, D, C, B, A]
8- open = [H, I, J, K, L, M, N]; closed = [G, F, E, D, C, B, A]
9- tiếp tục cho đến khi U được tìm thấy hoặc open = [ ]

Vd 2.6 Sử dụng không gian trạng thái như Hình 2.13. Hãy sử dụng giải thuật tìm kiếm theo chiều sâu để tìm kiếm trạng thái đích U trên đồ thị.

Kết quả các vòng lặp như sau:

1- open = [A]; closed = [ ]
2- open = [B, C, D]; closed = [A]
3- open = [E, F, C, D]; closed = [B, A]
4- open = [K, L, F, C, D]; closed = [E, B, A]
5- open = [S, L, F, C, D]; closed = [K, E, B, A]
6- open = [L, F, C, D]; closed = [S, K, E, B, A]

7- open = [T, F, C, D]; closed = [L, S, K, E, B, A]
8- open = [F, C, D]; closed = [T, L, S, K, E, B, A]
9- open = [M, C, D]; closed = [F, T, L, S, K, E, B, A]
10-open = [C, D]; closed = [M, F, T, L, S, K, E, B, A] tiếp tục cho đến khi U được tìm thấy hoặc open = [ ]

IV. TÌM KIẾM VỚI THÔNG TIN ĐÁNH GIÁ HEURISTIC

1. Thông tin đánh giá heuristic

Tương tự như não người với một cơ chế thu gọn nhằm thu thập những thông tin cần cho bài toán để thu gọn không gian bài toán và tìm kiếm lời giải bài toán có tối ưu.

Thông tin theo kiểu này được gọi là tri thức điều khiển của bài toán hay còn gọi là thông tin đánh giá heuristic gồm:

- Thông tin hàm đánh giá heuristic về bài toán được thể hiện dưới dạng hàm tính
- Thông tin luật đánh giá heuristic về bài toán được biểu diễn bằng luật điều khiển dưới dạng if-then.

Giải thuật sử dụng thông tin hàm đánh giá heuristic để thực hiện việc tìm kiếm được gọi là giải thuật heuristic. Nó thể hiện cách giải bài toán với các đặc tính sau:

- Thường tìm được lời giải tốt (nhưng không chắc là lời giải tối ưu nhất)
- Sử dụng thuật giải heuristic thường nhanh chóng và dễ dàng đưa ra kết quả do vậy chi phí thấp
- Giải thuật heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Có nhiều phương pháp để xây dựng một thuật giải heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ sở:

Nguyên lý vét cạn thông minh:

Khi không gian tìm kiếm của bài toán quá lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu

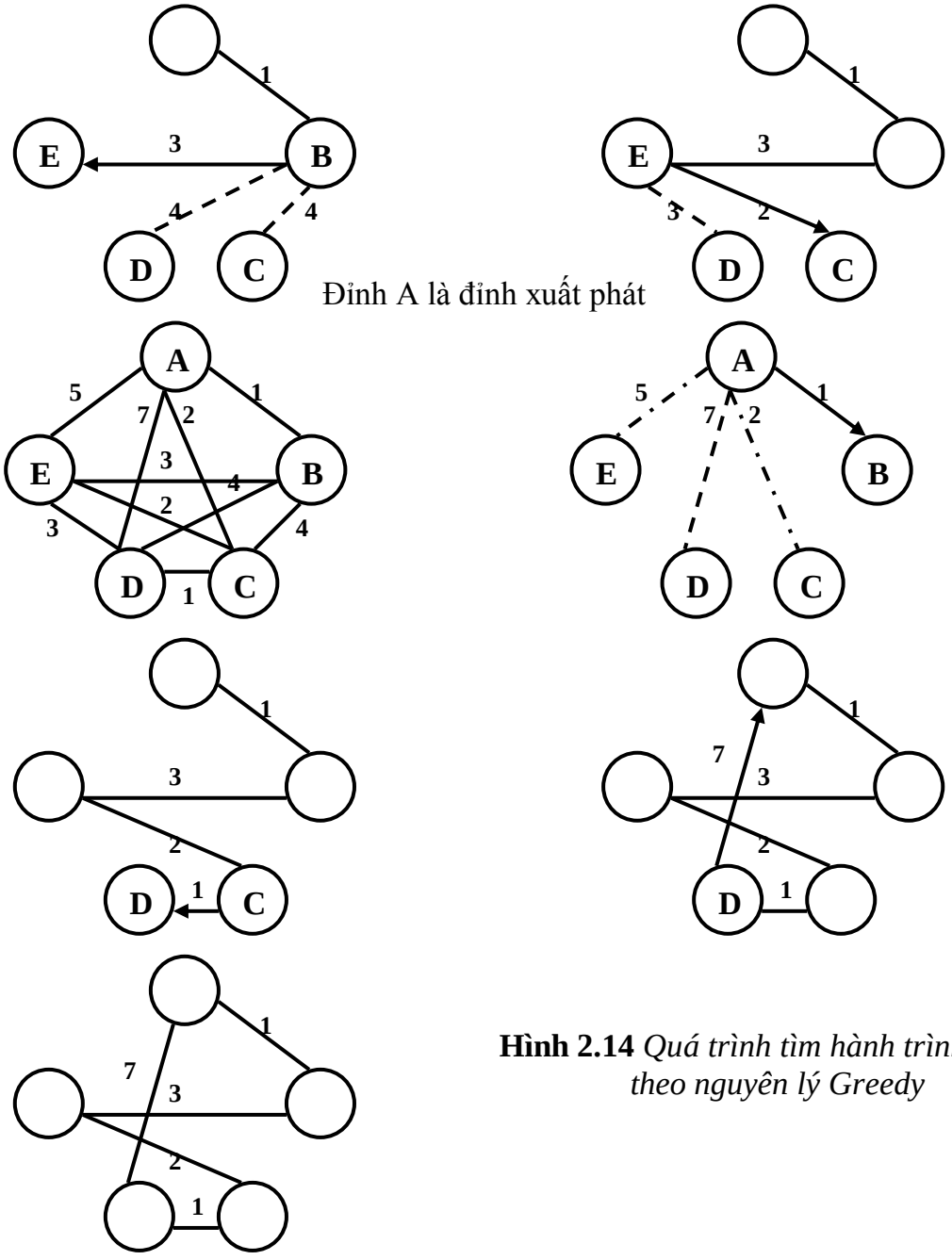
Nguyên lý tham lam (Greedy):

Lấy tiêu chuẩn tối ưu của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước trong quá trình tìm kiếm lời giải.

Nguyên lý thứ tự:

Thực hiện dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm mục đích đạt được một lời giải tốt.

Vd 2.7 Bài toán hành trình ngắn nhất - ứng dụng nguyên lý Greedy



Hình 2.14 *Quá trình tìm hành trình theo nguyên lý Greedy*

Sử dụng bài toán hành trình người bán hàng cho trong ví dụ trước. Yêu cầu là tìm hành trình ngắn nhất có thể được

- Ta có thể giải bài toán bằng cách

+ liệt kê danh sách tất cả các con đường có thể đi
+ tính chiều dài mỗi con đường đó
+ tìm con đường có chiều dài ngắn nhất

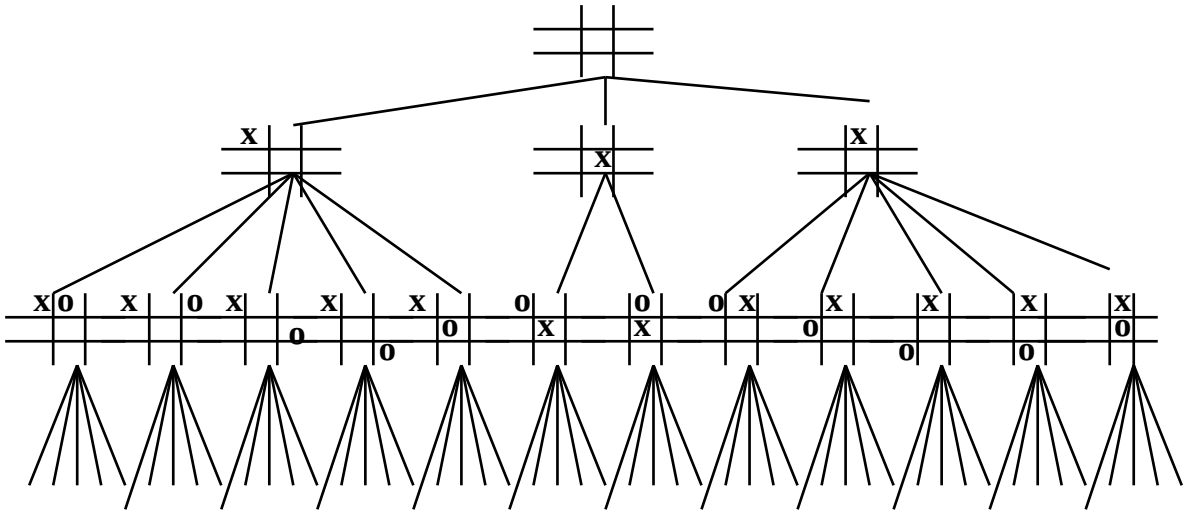
Cách giải này có độ phức tạp O(n!). Do đó khi số thành phố tăng tỉ lệ thuận với số đường đi phải xét sẽ tăng lên.

- Ta có thể sử dụng cách giải thứ hai đơn giản hơn và thường cho kết quả tương đối tốt là dùng giải thuật heuristic ứng dụng nguyên lý Greedy: (Hình 2.14)

+ Từ điểm khởi đầu, liệt kê tất cả các con đường bắt đầu từ thành phố xuất phát cho đến n thành phố rồi chọn đi theo con đường ngắn nhất

+ Khi đã đến một thành phố, chọn đi đến thành phố kế tiếp cũng theo nguyên tắc trên: liệt kê tất cả các đường đi từ thành phố ta đang đứng đến những thành phố chưa đi đến. Chọn con đường ngắn nhất. Lặp lại quá trình này cho đến lúc không còn thành phố nào để đi.

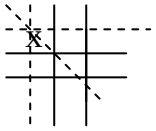
Vd 2.8 Xem xét bài toán trò chơi caro trong phạm vi kích thước 3x3. Có hai đấu thủ Plus và Minus: Plus đặt nước cờ X, Minus đặt nước cờ O. Plus đặt nước cờ đầu tiên, Minus đặt nước cờ kế theo. Không gian trạng thái của bài toán được biểu diễn như Hình 2.15



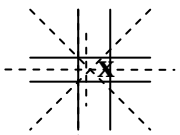
Hình 2.15

Không gian bài toán đã được thu gọn nhờ tính đối xứng, minh hoạ hình 2.15

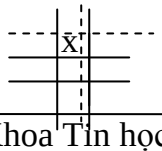
Một thông tin heuristic đơn giản có thể loại bỏ phần lớn các đường tìm kiếm không cần thiết trong không gian bài toán đó là ta có thể chọn nước cờ X sao cho nó có nhiều đường mở nhất để di chuyển nước cờ. Các đường mở cho ba trạng thái đầu tiên trong trò chơi được tính như sau :



Nếu nước cờ X được đặt tại ô vuông góc, X có ba đường mở để di chuyển nước cờ



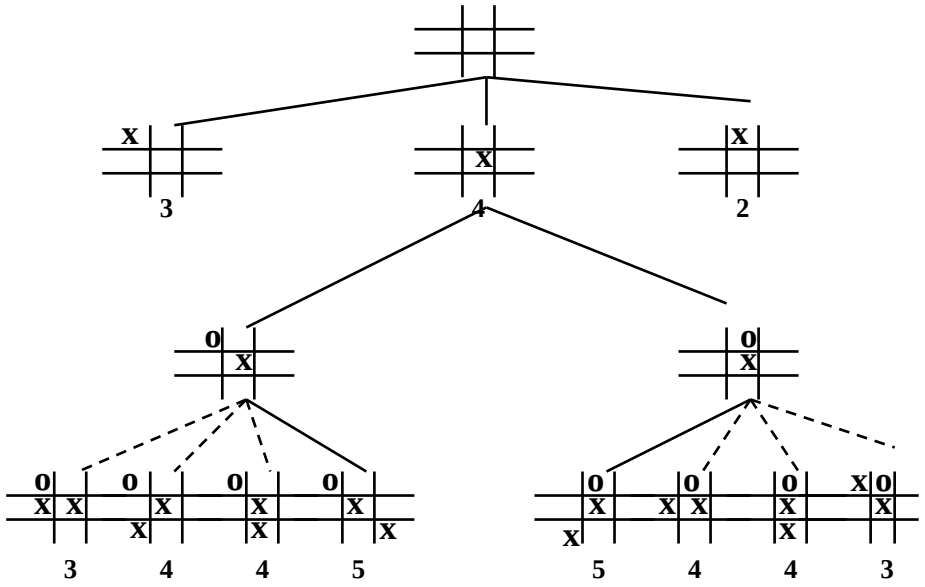
Nếu nước cờ X được đặt tại ô vuông tâm, X có bốn đường mở để di chuyển nước cờ



Nếu nước cờ X được đặt tại ô vuông cạnh, X có hai đường mở để di chuyển nước cờ

Với thông tin heuristic này, ta sử dụng để thu gọn không gian trạng thái cho bài toán trò chơi caro (Hình 2.16)

Nếu các trạng thái có cùng số đường mở, ta chọn trạng thái đầu tiên đã được tìm thấy để di chuyển nước cờ



Hình 2.16

2. Giải thuật best-first-search

Giải thuật best-first-search sử dụng hai danh sách open và closed

- Danh sách open chứa các trạng thái đang chờ để được duyệt qua
- Danh sách closed chứa các trạng thái đã được duyệt qua
- Thứ tự các trạng thái được chọn duyệt qua trên danh sách open phải tuân theo một vài ước lượng heuristic của chúng tốt nhất để đến đích.

Mô tả giải thuật như sau:

```
procedure best_first_search
begin
    open:= [start]; closed:= [];
    while open ≠ [] do
        begin
            huỷ bỏ trạng thái đầu tiên từ danh sách open, gọi nó là X;
            if X=đích then trả về đường đi từ start đến X
            else begin
                Phát sinh các kế thừa của X;
                for mỗi kế thừa của X do
```

case

kế thừa chưa có mặt trên open hoặc closed:

begin

gán kế thừa với một giá trị heuristic

cộng kế thừa vào danh sách open

end;

kế thừa đã có mặt trên danh sách open:

if kế thừa có đường đi tốt hơn then chọn trạng thái mới và huỷ bỏ trạng thái cũ trên danh sách open.

kế thừa đã có mặt sẵn trên danh sách closed:

if kế thừa có đường đi tốt hơn then

begin

huỷ bỏ trạng thái cũ khỏi danh sách closed;

cộng trạng thái mới vào danh sách open

end;

end;

đặt X vào danh sách closed

tổ chức lại thứ tự các trạng thái trên danh sách open theo thứ tự từ trái sang phải ứng với giá trị heuristic từ tốt đến xấu.

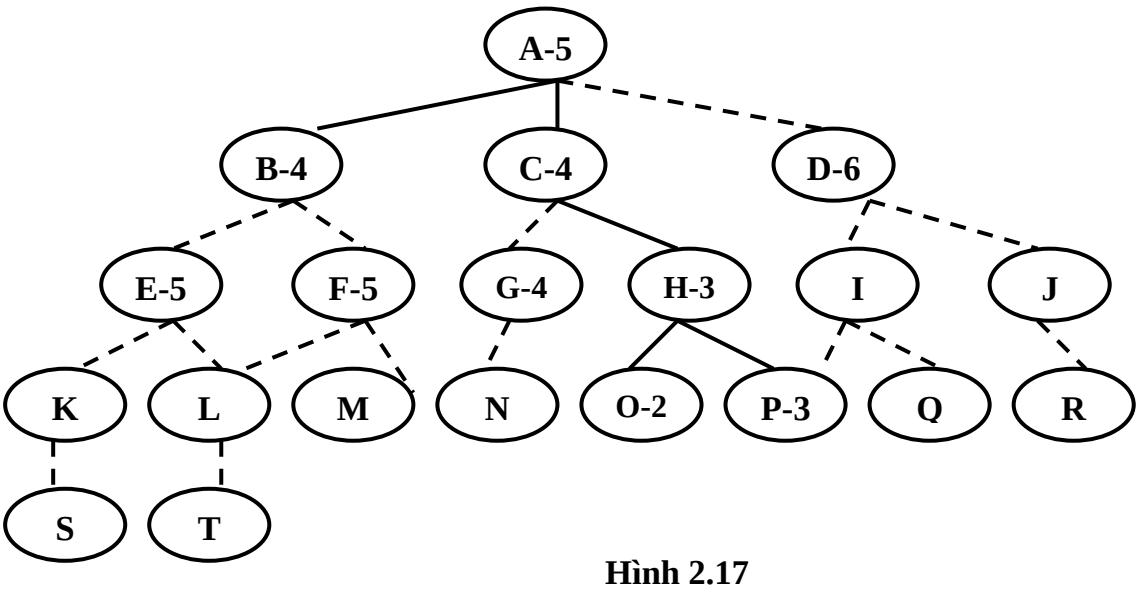
end;

return failure

end;

end.

Vd 2.9 Cho không gian bài toán được biểu diễn bằng đồ thị dưới đây:



Hình 2.17

Hãy sử dụng giải thuật best-first-search để tìm trạng thái đích P của bài toán ?

Giải thuật best-first-search ứng dụng các thông tin đánh giá heuristic đến các trạng thái trên danh sách open và danh sách được sắp xếp tuân theo các giá trị heuristic.

Điều này cho phép trạng thái tốt nhất được đưa về đứng đầu danh sách open. Giải thuật tìm đường đi đến trạng thái đích P thông qua các vòng lặp như sau:

1- open = [A5]; closed = []

2- chọn A5; open = [B4, C4, D6]; closed = [A5]

3- chọn B4; open = [C4, E5, F5, D6]; closed = [B4, A5]

4- chọn C4; open = [H3, G4, E5, F5, D6]; closed = [C4, B4, A5]

5- chọn H3; open = [O2, P3, G4, E5, F5, D6]; closed = [H3, C4, B4, A5]

6- chọn O2; open = [P3, G4, E5, F5, D6]; closed = [O2, H3, C4, B4, A5]

7- chọn P3; tìm thấy lời giải!

3. Hàm đánh giá heuristic

Vd 2.10 Xem xét bài toán trò chơi 8 số như một bàn cờ có 3 hàng và 3 cột tạo thành 9 ô, trong đó 8 ô chứa 8 viên ngói được đánh số từ 1 đến 8, ô còn lại là ô trống. Trạng thái ban đầu của bài toán trò chơi 8 số được biểu diễn như hình bên

2	**8**	**3**
1	**6**	**4**
7		**5**

Yêu cầu: Dịch chuyển các viên ngói sao cho từ trạng thái ban đầu đạt đến một trạng thái đích nào đó

Luật chơi: Cho phép trượt các viên ngói đến ô trống kề nó, không cho phép dịch chuyển các viên ngói theo đường chéo.

Thông tin đánh giá heuristic của bài toán là số viên ngói đặt không đúng chỗ tại mỗi trạng thái khi nó được so sánh với trạng thái đích.

Trạng thái nào có ít viên ngói đặt không đúng chỗ nhất so với trạng thái đích đó là trạng thái tốt nhất được chọn duyệt qua để tiến đến đích

Nếu có hai trạng thái có cùng thông tin đánh giá heuristic, thì lúc này trạng thái được chọn để duyệt là trạng thái gần trạng thái gốc nhất của đồ thị. Trạng thái này sẽ có đường ngắn nhất để đi đến đích.

Khoảng cách từ trạng thái ban đầu đến các con cháu của nó được đo bằng cách tính độ sâu cho mỗi trạng thái. Tính 0 cho trạng thái ban đầu và được tăng lên 1 cho mỗi mức kế theo

Phép tính độ sâu này phản ánh số phép dịch chuyển thực sự mà đã được sử dụng đi từ trạng thái ban đầu trong việc tìm kiếm đến mỗi trạng thái con cháu, vì thế nó phải được bổ sung vào việc đánh giá heuristic. Do đó để có đầy đủ thông tin đánh giá heuristic cho mỗi trạng thái trong không gian trạng thái tìm kiếm của bài toán, một hàm đánh giá heuristic f được định nghĩa đó là tổng của hai thành phần:

f(n) = g(n) + h(n)

với: g(n) – đo chiều dài thực sự của đường từ bất kỳ trạng thái n nào đến trạng thái bắt đầu

h(n) – là một ước lượng heuristic của khoảng cách từ trạng thái n đến trạng thái đích

Trong trò chơi tám số, chẳng hạn ta xem h(n) là số viên ngói đặt không đúng chỗ so với trạng thái đích.

Trong ví dụ giả sử ta có trạng thái ban đầu và trạng thái đích như hình vẽ

2	**8**	**3**
1	**6**	**4**
7		**5**

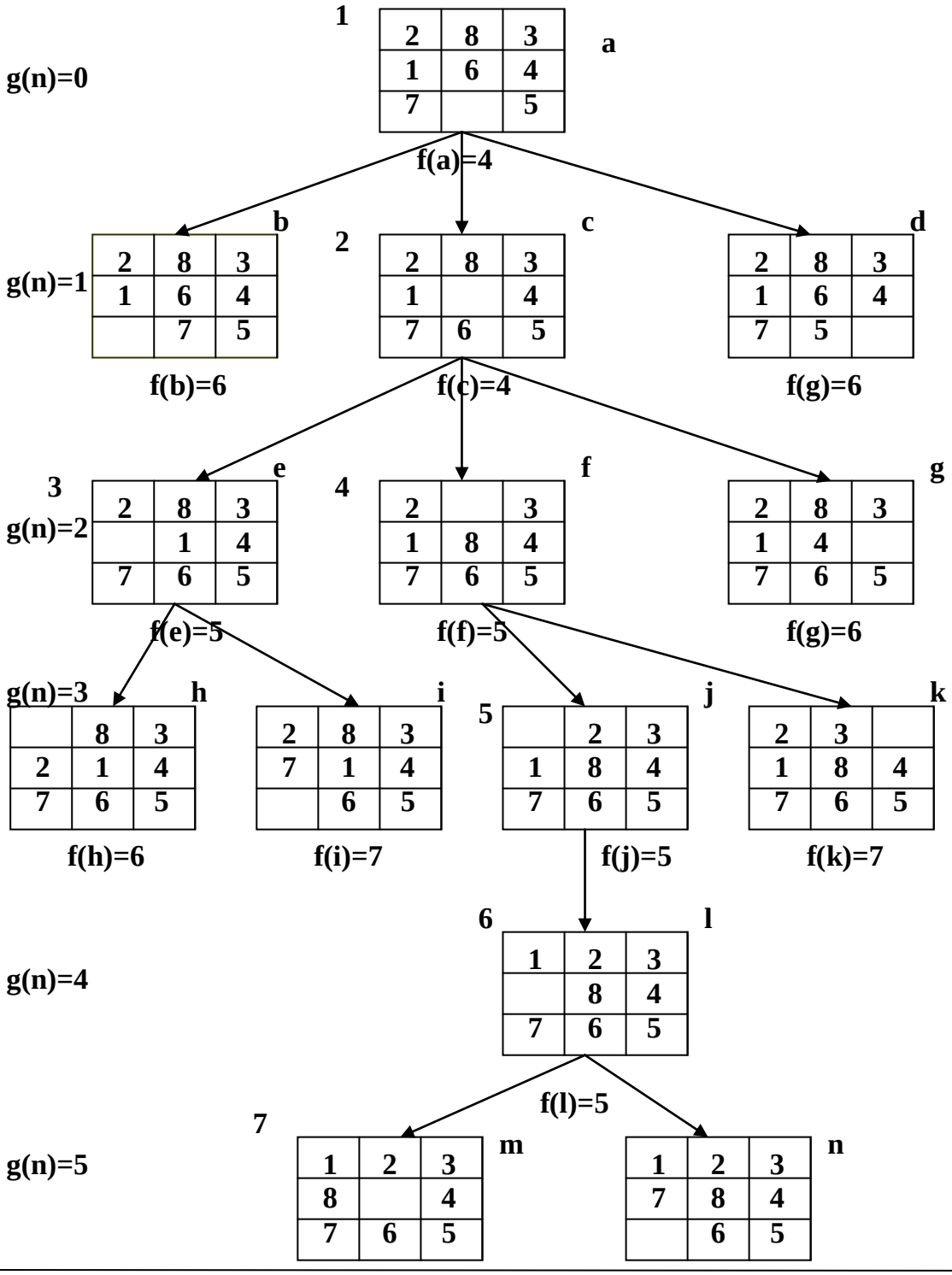
Start state

1	**2**	**3**
8		**4**
7	**6**	**5**

Goal state

Hãy sử dụng giải thuật best-first-search với hàm đánh giá heuristic f đã được định nghĩa để giải bài toán.

Giải thuật best-first-search với hàm đánh giá heuristic f(n) = g(n)+h(n) được sử dụng để giải bài toán trò chơi 8 số được mô tả ở hình 2.18.


1
a
g(n)=0
f(a)=4
b
2
c
d
g(n)=1
f(b)=6
f(c)=4
f(g)=6
3
e
4
f
g
g(n)=2
f(e)=5
f(f)=5
f(g)=6
g(n)=3
h
i
5
j
k
f(h)=6
f(i)=7
f(j)=5
f(k)=7
6
l
g(n)=4
f(l)=5
7
m
n
g(n)=5
f(m)=5
f(n)=7
Goal

Hình 2.18

Trong hình 2.18, mỗi trạng thái được đánh nhãn với một chữ cái và trọng số heuristic của nó f(n) = g(n) + h(n).

- Số tại đỉnh của mỗi trạng thái chỉ số thứ tự của trạng thái được chọn để duyệt từ danh sách open
- Các trạng thái không được đánh số thứ tự vì chúng vẫn có mặt trên danh sách open khi giải thuật đã được hội tụ.
- Các chữ cái trong danh sách biểu diễn các trạng thái và các chữ số biểu diễn các giá trị của hàm đánh giá heuristic tại các trạng thái đó.

Qua đồ thị ta có kết quả các trạng thái của danh sách open và closed được phát sinh như sau:

8- open = [a4]; closed = []
9- open = [c4, b6, d6]; closed = [a4]
10-open = [e5, f5, g6, b6, d6]; closed = [a4, c4]
11-open = [f5, h6, g6, b6, d6, i7]; closed = [a4, c4, e4]
12-open = [j5, h6, g6, b6, d6, k7, i7]; closed = [a4, c4, e5, f5]
13-open = [l5, h6, g6, b6, d6, k7, i7]; closed = [a4, c4, e5, f5, j5]
14-open = [m5, h6, g6, b6, d6, n7, k7, i7]; closed = [a4, c4, e5, f5, j5, l5]
15-thành công m = đích

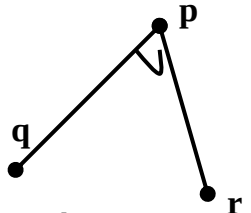
V. ĐỒ THỊ VÀ - HOẶC (and-or graphs)

Đồ thị dùng để biểu diễn không gian bài toán mà các luật của nó được thiết kế với dạng p → q, gọi là đồ thị hoặc như ta đã thảo luận ở trên.

Một cấu trúc khác dùng để biểu diễn không gian trạng thái của bài toán mà các luật của nó được thiết kế với các dạng q ■ r → p và q ■ r → p, trong đó ■ là các toán tử và, hoặc, cấu trúc này được gọi là đồ thị **và-hoặc**.

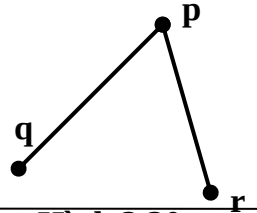
Trong đồ thị **và-hoặc**, luật với dạng q ■ r → p, giá trị chân lý của p phụ thuộc vào cả hai giá trị chân lý của q và r, trong khi đó luật với dạng q ■ r → p, giá trị chân lý của p chỉ phụ thuộc vào một trong hai giá trị chân lý q hoặc r. Luật với dạng q ■ r → p có thể được tách ra thành hai luật đó là q → p và r → p

Nói chung nếu luật được thể hiện dưới dạng if-then mà vế bên trái của luật có nhiều toán tử kết nối **và** thì các toán tử **và** này tạo thành một cung, và cung này được gọi là cung **và** (hay còn gọi là đỉnh **và).** Mỗi đỉnh **và** có nhiều đỉnh kế thừa và mỗi kế thừa của nó được gọi là đỉnh thành phần. Hình 2.19 ở bên biểu diễn luật với dạng q ■ r → p, trong đó q, r được gọi là các đỉnh thành phần và p được gọi là đỉnh **và** (hay còn gọi là cung **và**).



Hình 2.19

Nếu luật được thiết kế dưới dạng if-then mà vế bên trái của luật có nhiều toán tử kết nối **hoặc**, thì các toán tử **hoặc** này tạo thành một cung và cung này được gọi là cung **hoặc** (hay còn gọi là đỉnh **hoặc**). Mỗi đỉnh **hoặc** có nhiều đỉnh kế thừa và mỗi kế thừa được gọi là đỉnh thành



Hình 2.20

phần. Hình 2.20 biểu diễn luật với dạng q ■ r → p, với q, r là các đỉnh thành phần của cung **hoặc** p

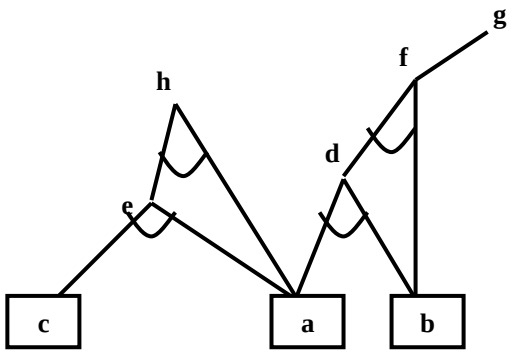
Vd 2.11 Cho tập các đề xuất sau là đúng:

Các tiên đề này phát sinh ra đồ thị và-hoặc như hình 2.21

Việc tìm kiếm trên đồ thị và-hoặc cũng giống như việc tìm kiếm trên đồ thị hoặc.

Tuy nhiên, trên đồ thị và-hoặc, để xác minh đỉnh cha giải được thì tất cả các đỉnh thành phần của nó phải giải được

a
b
c
a■b → d
a■c → e
b■d → f
f → g
a■e → h



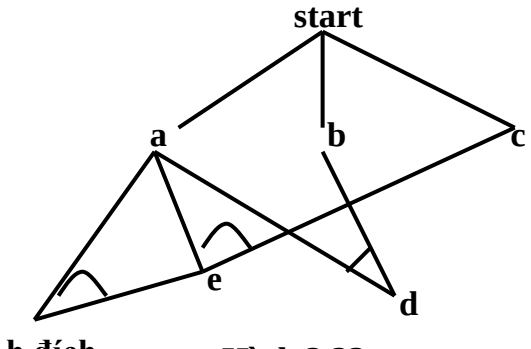
Hình 2.21

Nếu một trong các đỉnh thành phần không giải được thì đỉnh cha sẽ không giải được và đỉnh đó chính là đỉnh cụt trong đồ thị

Có hai chiến lược tìm kiếm trên đồ thị và-hoặc: chiến lược tìm kiếm hướng dữ liệu và chiến lược tìm kiếm hướng đích.

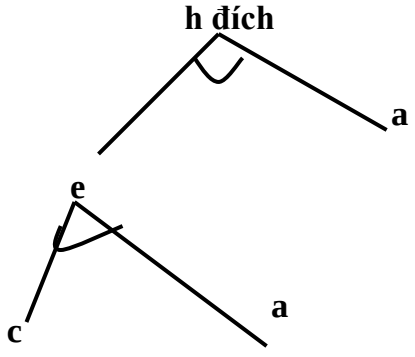
Vd 2.12. Hãy xác minh h trên đồ thị **và-hoặc** ở hình 2.21 sử dụng chiến lược tìm kiếm hướng dữ liệu và hướng đích.

Chiến lược tìm kiếm hướng dữ liệu để xác minh h đúng phải bắt đầu từ các dữ liệu đó là a, b và c. Từ đó hỗ trợ các luật a■b → d, a■c → e, và a■e → h đều có trị đúng. Qui trình của chiến lược này để chứng minh h đúng như được biểu diễn ở hình 2.22.



Hình 2.22

Chiến lược tìm kiếm hướng đích để xác minh h đúng phải bắt đầu từ dữ liệu đích đó là h. Xem xét tất cả các luật có khả năng phát sinh ra đỉnh đích này, chỉ có một luật đó là a■e → h. Sử dụng các sự kiện a và e thỏa mãn luật để phát sinh ra đỉnh đích làm các đỉnh thành phần. Xem xét tất cả các luật có khả năng phát sinh đỉnh e, chỉ có một luật duy nhất đó là a■c → e. Sử dụng các điều kiện a và c thỏa mãn luật để phát sinh ra đỉnh đích này làm các đỉnh



Hình 2.23

thành phần. Qui trình tìm kiếm hướng đích để chứng minh h đúng như được mô tả ở hình 2.23

Ta cũng có thể sử dụng giải thuật tìm kiếm truyền lùi trên đồ thị **và-hoặc.** Trên đồ thị **hoặc**, giải thuật truyền lùi sẽ xây dựng một đường lời giải của bài toán bắt đầu từ trạng thái đích đến trạng thái ban đầu nếu lời giải được tìm thấy; khi đó bài toán xem như đã được giải xong. Nếu có một đường cụt, giải thuật sẽ truyền lùi về gốc và sẽ thử tìm kiếm lời giải của bài toán trên một nhánh khác của đồ thị. Tương tự,trên đồ thị **và-hoặc**, đỉnh có thể giải được chỉ khi nào tất cả các đỉnh thành phần của nó phải giải được; mặt khác một trong các đỉnh thành phần của nó không giải được thì nó được xem như là đỉnh không giải được và nó chính là đỉnh cụt trên đồ thị **hoặc**.

1. Cách tính ước lượng heuristic trong đồ thị và-hoặc

Có thể sử dụng thông tin đánh giá heuristic để thu gọn không gian trạng thái của bài toán được biểu diễn bằng đồ thị và-hoặc. Cách tính ước lượng chi phí heuristic cho các cung trong đồ thị và chọn cung có giá trị ước lượng chi phí heuristic nhỏ nhất để triển khai các sự kế thừa như sau:

Ước lượng chi phí của cung bằng tổng số các ước lượng chi phí các đỉnh thành phần cộng tổng số các giá chi phí các đỉnh thành phần đó.

Ước lượng chi phí của cung = $\sum_i \sum_i st_i$

Vd 2.13 Giả sử đỉnh A là đỉnh đầu tiên được chọn để mở rộng, xây dựng hai cung, một dẫn đến B và một dẫn đến C và D, ước lượng chi phí tại mỗi đỉnh thành phần: h(B) = 5, h(C) = 3, h(D) = 4

Vậy ước lượng chi phí của mỗi cung được tính:

Cung AB = h(B) + cost(B) = 5 + 1 = 6

Cung ACD = h(C) + h(D) + cost(C) + cost(D) = 3 + 4 +1 + 1 = 9

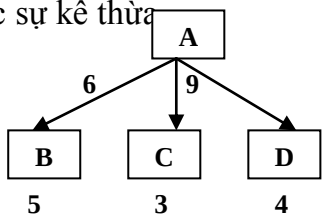
Ta nhận thấy cung AB = 6 có ước lượng chi phí nhỏ nhất, nên đỉnh B được chọn để triển khai các sự kế thừa. Giả sử các kế thừa của B là E và F với ước lượng chi phí h(E) = 5 và h(F) = 10.

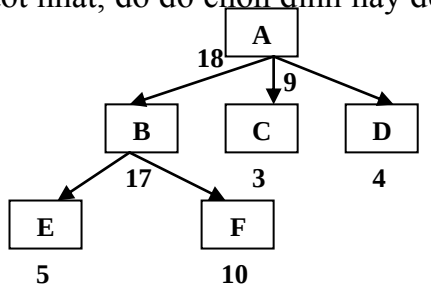
Ước lượng chi phí của cung EBF = h(E) + h(F) + cost(E) + cost(F)

= 5 + 10 + 1 + 1 = 17

Do đó, cung ABEF = cung EBF + cost(B) = 17 + 1 = 18

So sánh hai cung, ta thấy cung đi từ A dẫn đến C và D có ước lượng chi phí nhỏ nhất và đỉnh thành phần C có ước lượng chi phí tốt nhất, do đó chọn đỉnh này để khai triển các sự kế thừa





Giả sử C có các kế thừa là G và H với h(G) = 3 và h(H) = 4.

Ước lượng chi phí của cung GCH = h(G) + h(H) + cost(G) + cost(H)

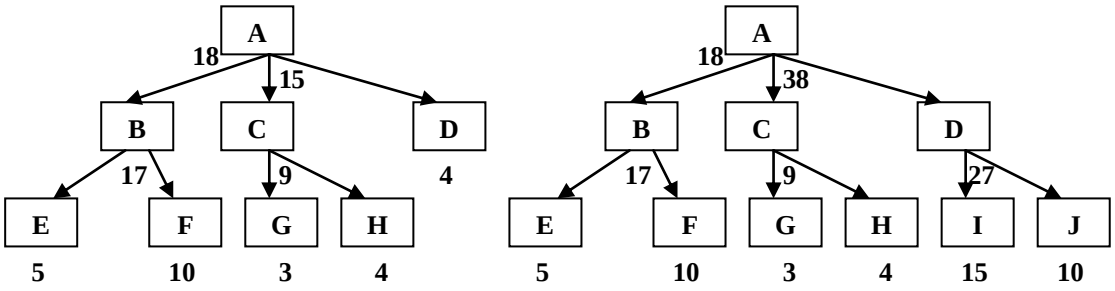
= 3 + 4 + 1 + 1 = 9

Cung ADCGH = h(D) + cung GCH + cost(D) + cost(C)

= 4 + 9 +1 +1 = 15 là cung vẫn có ước lượng chi phí nhỏ hơn cung vạch từ A dẫn đến B và từ B dẫn đến E và F, do đó chọn đỉnh D để triển khai các kế thừa. Các kế thừa của D là I và J với h(I) = 15 và h(J) = 10

Ước lượng chi phí của cung IDJ = h(I) + h(J) + cost(I) + cost(J)
= 15 + 10 + 1 + 1 = 27

Vì thế cung vạch từ A đến cung C và cung D =
= cung(C) + cung(D) + cost(C) + cost(D) = 9 + 27 + 1 + 1 = 38



2. Giải thuật đồ thị và-hoặc

Sử dụng giải thuật đồ thị và-hoặc để tìm kiếm trên đồ thị và-hoặc. Giải thuật đồ thị và-hoặc dùng một danh sách G để biểu diễn một phần không gian trạng thái tìm kiếm của đồ thị. Giải thuật sử dụng chỉ một hàm ước lượng heuristic h để đánh giá mỗi trạng thái trong đồ thị. Tại mỗi trạng thái trong đồ thị, giải thuật nhìn xuống các kế thừa của nó và đồng thời nhìn lên tổ tiên của nó. Giải thuật cũng sử dụng một danh sách S nhằm mục đích cho quá trình truyền lùi về gốc của đồ thị. Giải thuật được mô tả như sau:

Bước 1: Cho G chứa một đỉnh biểu diễn trạng thái ban đầu. Gọi đỉnh này là init. Ước lượng h (init)

Bước 2 : Cho đến khi đỉnh init được đánh nhãn solved, lặp lại thủ tục sau :

1- Vạch các cung được đánh dấu từ init và chọn cung có ước lượng chi phí tốt nhất và đỉnh có ước lượng heuristic tốt nhất trong cung để mở rộng các kế thừa. Gọi đỉnh này là node.

2- Phát sinh các kế thừa của node :

Nếu node không có kế thừa thì gán giá trị vô ích đến đỉnh node tức là h(node) = ■, xem đỉnh này như là đỉnh không giải được, tức là đỉnh cụt.

Nếu node có các kế thừa thì cho mỗi kế thừa thực hiện :

- Cộng kế thừa vào đồ thị G
- Nếu kế thừa là một đỉnh cuối, đánh nhãn nó là solved và gán nó với một giá trị h bằng 0
- Nếu kế thừa không phải là đỉnh cuối, ước lượng giá trị h của nó.

3- Truyền lùi về gốc của đồ thị để khám phá thông tin mới bằng cách thực hiện thủ tục sau : cho S là một danh sách chứa các đỉnh mà đã được đánh dấu solved hoặc các giá trị h của chúng đã được thay đổi và vì vậy ta đã có các giá trị truyền lùi về các ông cha của chúng. Đầu tiên lấy một đỉnh trong S là node. Cho đến khi S là một danh sách rỗng thực hiện thủ tục sau :

- Chọn từ S một đỉnh không có con cháu của nó trong S. Gọi đỉnh này là đỉnh current. Huỷ bỏ nó khỏi S.

- Ước lượng giá chi phí của mỗi của các cung vạch ra từ current. Giá chi phí mỗi cung bằng tổng các giá trị h các đỉnh thành phần công với tổng các giá các đỉnh thành phần đó. Lấy giá trị mới h của đỉnh current là giá trị cực tiểu của các cung vừa được tính vạch ra từ nó.
- Đánh dấu đường đi tốt nhất từ current bằng cách đánh dấu cung có ước lượng chi phí cực tiểu vừa được tính ở bước trước đó.
- Đánh dấu current solved nếu tất cả các đỉnh liên kết với nó thông qua cung mới đều được đánh dấu solved.
- Nếu current đã được đánh dấu solved hoặc nếu giá của current vừa được thay đổi, thì trạng thái mới của nó phải được truyền lùi về đồ thị.Vì thế tất cả các đỉnh trong S là các tổ tiên của current.

3. Trò chơi max-min

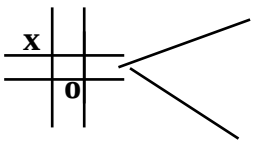
Xem xét bài toán trò chơi caro với phạm vi 3 x 3. Có hai đấu thủ Plus và Minus, Plus đặt các nước cờ X, và Minus đặt các nước cờ O. Ở mỗi nước đi, các đấu thủ xem trước hai nước cờ. Đấu thủ Plus luôn muốn cực đại số nước cờ còn mở của mình trong khi đó đấu thủ Minus thì ngược lại, luôn muốn cực tiểu số nước cờ còn mở của Plus. Ứng với đồ thị và-hoặc, đỉnh hoặc là đỉnh max, đỉnh và là đỉnh min. Ước lượng chi phí cho mỗi nước cờ p được tính như sau :

h(p) = (số dòng + số cột + số đường chéo còn mở đối với Plus) -
(số dòng + số cột + số đường chéo còn mở đối với Minus)

Nếu p là thế thắng đối với Plus thì h(p) = ■

Nếu p là thế thắng đối với Minus thì h(p) = -■

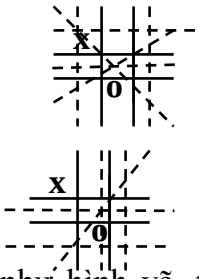
Nếu Plus và Minus đặt các nước cờ như hình vẽ, thì X và O có các đường mở được tính :



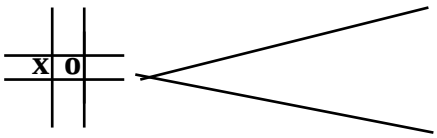
X có 6 đường còn mở

O có 5 đường còn mở

h(p)= 6 – 5 = 1



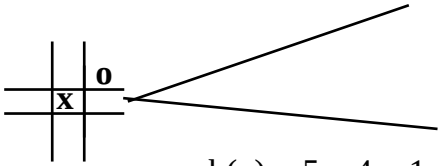
Tương tự, nếu Plus và Minus đặt các nước cờ như hình vẽ, thì X và O có các đường mở được tính:



X có 4 đường còn mở

O có 6 đường còn mở

h(p) = 4 – 6 = -2

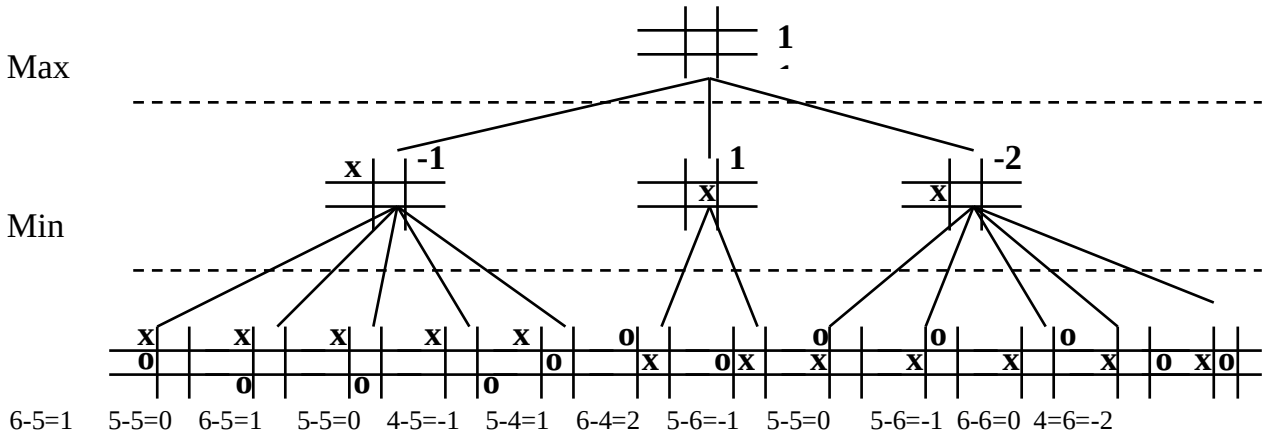


X có 5 đường còn mở

O có 4 đường còn mở

h(p) = 5 – 4 = 1

Sử dụng thông tin heuristic này với phép tính max-min, không gian bài toán caro được thu gọn lại sau hai nước cờ như H.2.24.



Hình 2.24

Cách tính ước lượng heuristic để đánh giá tại mỗi đỉnh cuối p đó là

h(p) = tổng số các đường mở đối với đấu thủ Plus - tổng số các đường mở đối với đối thủ Minus.

Heuristic tại mỗi đỉnh min bằng giá trị cực tiểu h của các đỉnh thành phần của đỉnh min đó

Heuristic tại đỉnh max bằng giá trị cực đại h của các đỉnh thành phần của đỉnh max đó.

Nước cờ tiếp theo của Plus là plus sẽ chọn đỉnh có đường đi tốt nhất (đường có nhiều đường mở nhất đối với Plus) để bắt đầu cho nước cờ kế theo.

VI. TÌM KIẾM VỚI GIẢI THUẬT UNIFORM COST

Bây giờ chúng ta sẽ xem xét giải thuật đầu tiên tìm kiếm đường đi tối ưu hoạt động bằng phương pháp đo "chiều dài đường".

Giải thuật uniform cost (UC) cũng dựa trên giải thuật tìm kiếm cơ bản đã được giới thiệu trước đây mà chúng ta đã sử dụng để minh hoạ các giải thuật tìm kiếm khác nhau.

1. Khởi tạo open với nút tìm kiếm (S); (set closed = (S))
2. If open là rỗng, fail. Else, nhặt một nút tìm kiếm (X) từ open
3. If state(X) là đích, return X (chúng ta đã đạt đến đích)
4. else remove X từ open
5. Tìm tất cả các children của state(X) (không có trong closed (chưa được thăm) và) tạo sự mở rộng một mức của X đến mỗi descendant
6. Thêm những đường đã mở rộng này vào open (đặt children của (X) vào closed)
7. Quay lại bước 2

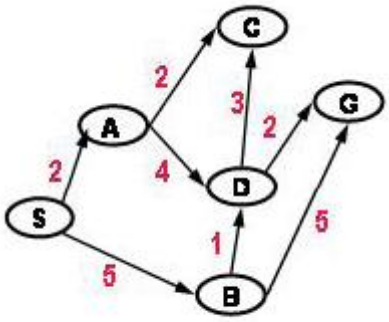
Không sử dụng closed cho tìm kiếm tối ưu

Chiến lược của tìm kiếm tối ưu với giải thuật uniform cost là :

- Nhặt thành phần tốt nhất của open làm nút kiểm tra và mở rộng (Bước 2)
- Thêm đường đã mở rộng vào bất cứ ở đâu trong open.

Điểm lưu ý ở đây là việc xác định thành phần tốt nhất trong open. Thay vì thành phần tốt nhất (được đo bằng giá trị heuristic của trạng thái) được chọn trong giải thuật best-first-search, thì thành phần tốt nhất được chọn ở đây sẽ dựa vào phép đo tổng chiều dài, hay tổng chi phí của đường đi.

Chúng ta muốn tổng chi phí đường đi là ít nhất, chúng ta sẽ chọn trạng thái tốt nhất là trạng thái với chi phí đường đi là nhỏ nhất hay chiều dài đường đi là ngắn nhất.



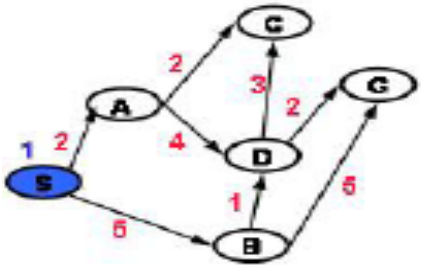
Tổng chi phí đường đi:

(S A C)	**4**
(S B D G)	**8**
(S A D C)	**9**
(S B G)	**10**

Hình 2.25

Bây giờ chúng ta sẽ áp dụng phương pháp tìm kiếm uniform-cost trên đồ thị có hướng trên. Tương tự những giải thuật kia, chúng ta sẽ khởi đầu với nút chứa trạng thái khởi đầu S. Đường đi này có chiều dài là 0 nên ta sẽ chọn đường này cho sự mở rộng

	Open
1	(0 S)

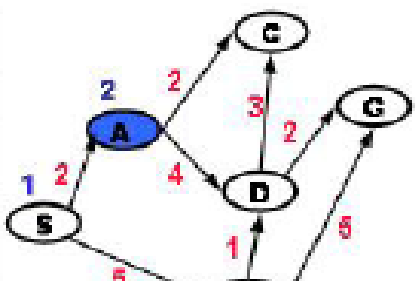


Nhặt thành phần tốt nhất của open, thêm đường đã mở rộng vào bất cứ ở đâu trong open.

Ghi chú: - Đường thêm vào là đường có hai cặp dấu ngoặc móc

- Đường được gạch chân là đường chọn cho sự mở rộng

	open
1	(0 S)

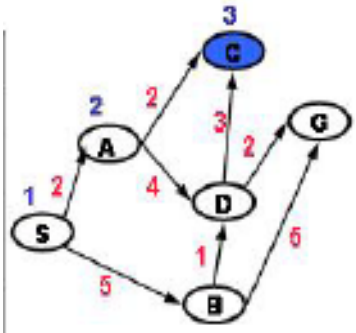


2	((2 A S)) ((5 B S))

Sẽ phát sinh 2 thực thể mới được thêm vào

danh sách open, đường đến A có chiều dài 2, đường đến B có chiều dài 5 nên chúng ta chọn đường đến A để mở rộng.

	Open
1	(0 S)
2	((2 A S)) ((5 B S))
3	((4 C A S)) ((6 D A S)) (5 B S)
4	(6 D A S) (5 B S)
5	((6 D B S)) ((10 G B S)) (6 D A S)
6	((8 G D B S)) ((9 C D B S)) (10 G B S) (6 D A S)
7	***((8 G D A S))*** ((9 C D A S)) (8 G D B S) (9 C D B S) (10 G B S)



Sẽ phát sinh 2 thực thể mới được thêm vào danh sách open, đường mới đến C là đường ngắn nhất trên open nên ta sẽ chọn nó để mở rộng. Nhưng C lại không có nút con nào nên chúng ta sẽ không mở rộng được một đường mới nào để thêm vào open. Ta sẽ chọn đường tốt nhất của những đường còn lại, bây giờ là đường đến B.

Tương tự đường đến B được mở rộng đến D và G, trong đó đường đến D là ngắn nhất, đường đến D từ B (S B D) bằng đường đến D từ A (S A D). Theo trật tự trong open ta sẽ chọn đường từ B để mở rộng.

Từ D lại mở rộng đến C và G, bây giờ đường đến D từ A là đường tốt nhất nên chúng ta chọn nó làm đường mở rộng

Sẽ phát sinh 2 trạng thái mới, một đường đến G và một đường đến C. Đường đi mới đến G là đường tốt nhất trên open nên ta sẽ nhặt nó từ danh sách open.

Vậy chúng ta đã tìm thấy đường đi ngắn nhất (S A D G) có chiều dài là 8

VII. TÌM KIẾM VỚI GIẢI THUẬT A*

Bây giờ chúng ta sẽ làm quen với một giải thuật tìm kiếm phổ biến nhất trong AI, giải thuật A*. Chúng ta sẽ làm quen bằng việc xác định những tương phản của nó so với uniform-cost.

- UC được mô tả là một giải thuật liên quan đến việc mở rộng chỉ những đường đi ngắn, cụ thể là nó không chú ý đến đích bởi vì không có cách nào để xác định được đích đấy ở đâu. UC thực chất chỉ là một giải thuật phục vụ cho việc tìm kiếm đường ngắn nhất đối với tất cả mọi trạng thái của đồ thị, nó không có bất kỳ một xu hướng cụ thể nào để tìm ra một đường đến đích sớm hơn trong quá trình tìm kiếm.
- Chúng ta có thể khiến cho UC có được khuynh hướng để tìm thấy được đường ngắn nhất đến đích mà chúng ta quan tâm bằng việc sử dụng một ước lượng heuristic khoảng cách đến đích. Dĩ nhiên khoảng cách đường đi này sẽ không chính xác được

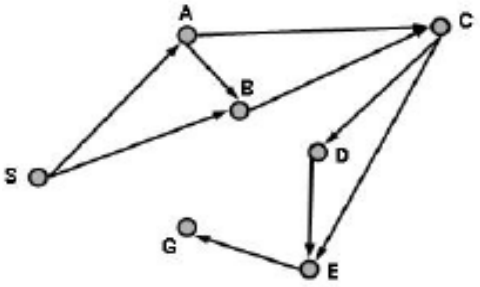
(nhưng chúng ta sẽ không cần thực hiện nhiều quá trình tìm kiếm trên nhiều đường), mặc dù vậy nó sẽ là một sự thay thế cho khoảng cách thực sự mà có thể cung cấp cho chúng ta một vài sự chỉ đạo.

- Thay vì liệt kê tất cả các đường đi chỉ theo trật tự của chiều dài đường length(g), chúng ta sẽ liệt kê các đường dưới dạng hàm f = g + h
  . f là ước lượng tổng chiều dài đường đi
  . g là chiều dài đường đi, length(g)
  . h là ước lượng khoảng cách từ trạng thái đến đích
- Một ước lượng luôn đánh giá không đúng mức chiều dài đường đi thực sự đến đích được gọi là có thể chấp nhận được.

Vd: ước lượng 0 là một ước lượng có thể chấp nhận được; khoảng cách đường chim bay là một ước lượng có thể chấp nhận được cho chiều dài đường đi trong không gian Euclid.

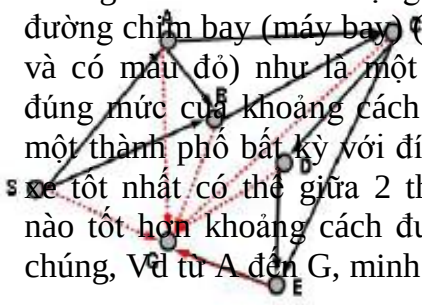
- Để đảm bảo rằng UC sẽ vẫn tìm thấy đường ngắn nhất bằng cách mở rộng những đường đi không hoàn chỉnh dựa trên ước lượng tổng chiều dài đường đi đến đích (tương tự UC nhưng không có danh sách mở rộng) chúng ta sẽ phải đảm bảo rằng ước lượng heuristic của chúng ta có thể chấp nhận được.
- Một tìm kiếm UC với một ước lượng heuristic có thể chấp nhận được được gọi là tìm kiếm A*_ giải thuật tìm kiếm rất phổ biến trong AI.

Xét đồ thị bên, giả sử rằng nó tồn tại trong không gian Euclid. Mỗi trạng thái trong đồ thị đóng vai trò là một thành phố và các đường nối tương ứng với khoảng cách lái xe từ thành phố này đến thành phố khác dọc theo các con đường tốt nhất (chi phí thấp).



Hình 2.26

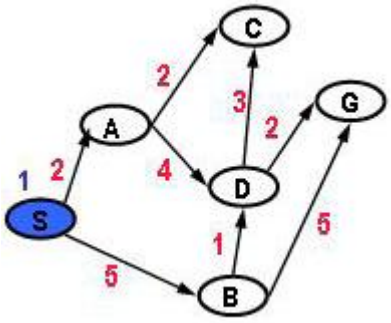
Chúng ta có thể sử dụng khoảng cách theo đường chim bay (máy bay) (đường biểu diễn đứt và có màu đỏ) như là một sự đánh giá không đúng mức của khoảng cách lái xe thực sự giữa một thành phố bất kỳ với đích. Khoảng cách lái xe tốt nhất có thể giữa 2 thành phố không thể nào tốt hơn khoảng cách đường chim bay giữa chúng, Vd từ A đến G, minh hoạ Hình 2.27



Hình 2.27

Sử dụng lại sơ đồ Hình 2.25 với giải thuật tìm kiếm A* (tìm kiếm uniform-cost sử dụng heuristic)

	Open
1	(0 S)
2	((4 A S)) ((8 B S))
3	((5 C A S)) ((7 D A S)) (8 B S)
4	(7 D A S) (8 B S)
5	***((8 G D A S))*** ((10 C D A S)) (8 B S)



- Chọn thành phần tốt nhất của open (chiều dài đường đi + heuristic)
- Thêm đường đã mở rộng vào vị trí bất kỳ trong Q

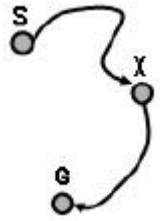
Quy định : đường được bao bởi hai cặp dấu ngoặc móc là những đường được thêm vào; đường được gạch chân là đường được chọn cho sự mở rộng

Chúng ta bắt đầu từ S như thường lệ. Tiếp theo sẽ mở rộng đến A và B. Ở đây chúng ta đang sử dụng chiều dài đường + ước lượng heuristic nên đường S-A có giá trị 4, S-B có giá trị 8, do đó ta sẽ chọn đường đến A làm đường mở rộng. Tương tự chúng ta sẽ tiếp tục mở rộng đường đi đến đích theo kiểu đường nào có chi phí tốt nhất (ít nhất) sẽ được chọn làm đường mở rộng. Ta sẽ dừng với đường đến đích (S-A-D-G)có giá trị chi phí là 8.

VIII. NGUYÊN TẮC LẬP TRÌNH ĐỘNG TỐI ƯU TRONG UNIFORM COST VÀ A*

Trong tìm kiếm UC và A* ở trên chúng ta vẫn chưa đề cập đến vấn đề một trạng thái được thăm lại (revisiting) nhiều lần, hay nói cách khác chúng ta chưa xây dựng một danh sách closed chứa các trạng thái đã được thăm.

1. Nguyên tắc lập trình động tối ưu:



Theo sơ đồ bên, đường ngắn nhất từ S đến G thông qua trạng thái X được cấu thành bởi đường ngắn nhất từ S đến X và đường ngắn nhất từ X đến G.
Đây chính là nguyên tắc lập trình động tối ưu.

Điều này có nghĩa là chúng ta chỉ cần giữ một đường tốt nhất duy nhất từ S đến một trạng thái X bất kỳ, nếu có một đường mới được tìm thấy đến một trạng thái đã có trong open thì đường dài hơn phải bị loại bỏ đi.

Lưu ý rằng, lần đầu tiên UC lôi một nút tìm kiếm ra khỏi Q (mở rộng) mà trạng thái của nó là X thì đường này chính là đường ngắn nhất từ S đến X bởi vì UC mở rộng các nút theo trật tự của chiều dài đường đi thật sự. Như vậy trạng thái mà chúng ta đã mở rộng sẽ nằm trong danh sách closed (đã duyệt). Nếu chúng ta mở rộng một nút mà trạng thái của nó có trong danh sách closed thì chúng ta phải bỏ đường này. Tức là giải thuật của chúng ta sẽ không mở rộng những trạng thái đã được thăm.

Chú ý: Tìm kiếm UC vẫn đảm bảo đúng nếu không dùng danh sách closed, nhưng sẽ không hiệu quả cao đối với những đồ thị có nhiều kết nối.

2. Áp dụng cho giải thuật tìm kiếm uniform-cost (UC)

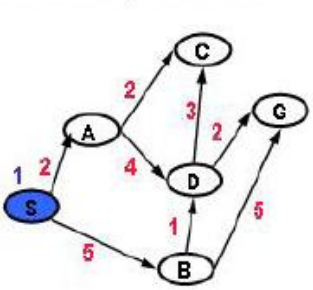
Giải thuật UC sẽ có một ít thay đổi để áp dụng nguyên tắc tối ưu
Sau đây là giải thuật tìm kiếm tối ưu đơn giản (Uniform cost + closed list).

1. Khởi tạo open với nút tìm kiếm (S); set closed = ( )
2. If open là rỗng, fail. Else, nhặt một nút tìm kiếm có chi phí thấp nhất X từ open
3. If state(X) là đích, return X (chúng ta đã đạt đến đích)
4. else remove X từ open
5. If state(X) có trong closed, go to bước 2, else thêm state(X) vào closed
6. Tìm tất cả các children của state(X) không có trong closed (chưa được thăm) và tạo sự mở rộng một mức của X đến mỗi descendant
7. Thêm những đường đã mở rộng này vào open; if trạng thái của descendant đã có trong open thì chỉ giữ lại đường ngắn hơn
8. Go to bước 2

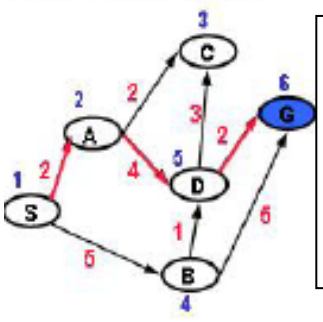
Bây giờ chúng ta sẽ áp dụng giải thuật trên cụ thể đối với tìm kiếm sơ đồ Hình 2.25 trong ví dụ trước. Khởi đầu với S có chiều dài đường đi là 0

	Open	closed
1	(0 S)	
2	((2 A S)) ((5 B S))	S
3	((4 C A S)) ((6 D A S)) (5 B S)	SA
4	(6 D A S) (5 B S)	SAC
5	~~((6 D B S))~~ ((10 G B S)) (6 D A S)	SACB
6	***((8 G D A S))*** ~~((9 C D A S))~~ ~~(10 G B S)~~	SACBD

Trạng thái ban đầu



- Đường thêm vào trong danh sách open là đường có hai cặp dấu ngoặc móc
- Đường được gạch chân là đường được chọn cho sự mở rộng



Chúng ta có thể dễ dàng nhận thấy rằng việc sử dụng danh sách closed sẽ giúp chúng ta chỉ giữ lại một đường ngắn nhất đến bất cứ trạng thái nào trong open, duy trì được tính tối ưu quan trọng trong tìm kiếm UC.

Bây giờ chúng ta hãy xem xét liệu điều này có thể đúng đối với tìm kiếm A* không?

3. Áp dụng cho giải thuật tìm kiếm A*

Trước tiên chúng ta hãy xem xét lại giải thuật A* không có danh sách closed

- A* chọn nút với giá trị f nhỏ nhất để mở rộng, f(N) = g(N) + h(state(N)): ước lượng tổng chi phí đường đi của một nút tìm kiếm (search node)

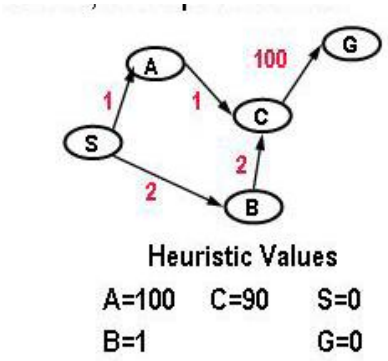
+ Với g(N) là chi phí (chiều dài) thực sự của đường đi từ trạng thái N bất kỳ đến trạng thái bắt đầu.

+ h(state(N)) là ước lượng heuristic của khoảng cách từ trạng thái N đến trạng thái đích.

- Vậy A* (mà không kèm danh sách closed) với một ước lượng heuristic có thể chấp nhận được được đảm bảo tìm thấy đường đi tối ưu

Tiếp theo, nếu chúng ta cũng thực hiện xây dựng một danh sách closed cho A* giống như trong UC thì để đảm bảo tìm thấy được đường tối ưu, đòi hỏi có một điều kiện mạnh hơn cho heuristic

Vd 2.14: Với những ước lượng heuristic được liệt kê dưới đây, A* sử dụng một danh sách closed sẽ không tìm thấy được đường tối ưu. Ước lượng heuristic tại B cố gắng để tìm đường tối ưu đã làm cho giải thuật tìm kiếm A* đi nhầm đường. C được mở rộng trước khi đường tối ưu đến nó được tìm thấy.



	open	closed
1	(0 S)	
2	((3 B S)) ((101 A S))	S
3	((94 C B S)) (101 A S)	B S
4	(101 A S) ((104 G C B S))	C B S
5	***(104 G C B S)***	A C B S

Các thao tác của giải thuật A* được trình bày một cách chi tiết cho chúng ta thấy nó đã đi sai đường. Đường chính xác qua A đã bị khoá khi đường đến C qua B được mở rộng.

Trong bước 4, khi A cuối cùng được mở rộng thì đường mới đến C không được đặt vào trong open bởi vì C đã được mở rộng rồi.

Hướng giải quyết giúp chúng ta có thể thực thi được A* với một danh sách closed mà vẫn đảm bảo được tính tối ưu để tìm ra đường đi ngắn nhất đến đích là heuristic phải thoả mãn hai điều kiện sau:

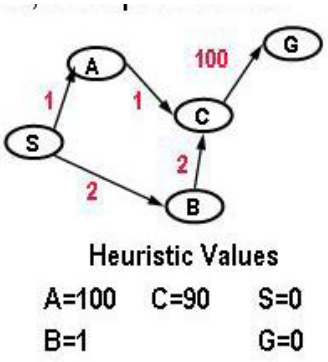
- Ước lượng heuristic của trạng thái đích phải bằng 0: $h(s_i) = 0$; với n_i là đích
- Ước lượng heuristic giữa một trạng thái đến các trạng thái con của nó phải nhỏ hơn hoặc bằng chi phí đường đi thực sự giữa chúng.

Điều muốn nhấn mạnh ở đây đó là điều kiện đối với heuristic như được đề cập ở trên chỉ cần thiết cho việc đảm bảo tính triệt để của tối ưu hoá thuật toán A* (có closed) khi chúng ta muốn loại bỏ đường đi qua những trạng thái đã được thăm 1 lần trước đó (có mặt trong danh sách closed)

Vậy một tìm kiếm A* mà không kèm theo danh sách closed thì chỉ cần một ước lượng heuristic có thể chấp nhận được là đủ để đảm bảo tìm ra đường đi tối ưu (đường

ngắn nhất). Điều này sẽ vẫn đúng trong trường hợp heuristic không thoả mãn hai điều kiện được nêu ở trên.

	Open
1	(0 S)
2	((3 B S)) ((101 A S))
3	((94 C B S)) (101 A S)
4	(101 A S) ((104 G C B S))
5	((92 C A S)) (104 G C B S)
6	***((102 G C A S))*** (104 G C B S)



Sơ đồ tìm kiếm trên cũng với ví dụ 2.14 chúng ta đã quan sát trước đây minh hoạ một tìm kiếm A* không có danh sách closed thì sẽ không phát sinh vấn đề sai lệch trong tìm kiếm đường đi tốt nhất. Ước lượng heuristic ở đây là một ước lượng có thể chấp nhận được nhưng lại không thoả mãn đủ hai điều kiện mà chúng ta đã nêu trên nhưng nó vẫn đảm bảo được tính tối ưu đối với giải thuật A* trong trường hợp này (không có danh sách closed)

VII. BÀI TẬP

Bài 1. Xây dựng không gian trạng thái đối với bài toán sau:

Cho n thành phố đánh số từ 1 đến n. Giao thông đường bộ giữa hai thành phố i và j được cho bởi giá trị aij như sau: aij = -1 có nghĩa là không có đường bộ đi từ thành phố i sang thành phố j và aij = 1 nếu có đường đi trực tiếp từ thành phố i sang thành phố j. Tìm đường đi từ thành phố i0 sang thành phố j0.

Bài 2. Xây dựng không gian trạng thái đối với bài toán sau:

Một dãy các số nguyên dương a1, a2, …, an được gọi là hợp lý nếu thoả mãn hai điều kiện:

- a_n là số nguyên tố.
- $a_{i+1} = a_i + 1$ hoặc $2*a_i$

Cho trước số a_1, hãy tìm dãy hợp lý $a_1, a_2, \ldots, a_n$.

Bài 3. Bài toán người đưa hàng.

Người đưa hàng cần phải xác định được hành trình ngắn nhất sao cho mỗi thành phố đi đến đúng một lần và quay trở lại thành phố xuất phát. Giả sử thành phố xuất phát là thành phố 1, có tất cả n thành phố đánh số từ 1 đến n. Hãy xây dựng không gian trạng thái đối với bài toán.

Bài 4. Thiết kế giải thuật truyền lùi trên đồ thị và-hoặc.

Bài 5. Phát triển không gian trạng thái của bài toán bình đựng nước sử dụng chiến lược suy diễn tiến và chiến lược suy diễn lùi.

Bài 6. Người ta sử dụng hai bình chứa có dung tích lần lượt là 3(lít) và 4(lít) để đong 2(lít) nước. giả sử lượng nước được lấy từ vòi không hạn chế và công để lấy nước từ vòi cho đầy một bình là 3, công để đổ nước trong một bình ra ngoài là 2 và đổ nước từ bình này sang bình khác thì tốn công là 5.

Hãy chỉ ra quá trình tìm kiếm lời giải bằng phương pháp tìm kiếm theo chiều rộng

Bài 7. Đại dương được xem như là một mặt phẳng toạ độ trên đó có n hòn đảo với toạ độ lần lượt là (x1, y1), (x2, y2), …, (xn, yn). Một chiếc ca nô xuất phát từ đảo d1 muốn tuần tra đến đảo d2. Bình xăng của ca nô chỉ chứa đủ xăng để đi được một quãng đường dài không quá m (km). Trên đường đi ca nô có thể ghé một số đảo nào đó để tiếp thêm xăng, lúc này ca nô được tiếp thêm xăng đầy bình chứa. Hãy chỉ ra một đường đi từ đảo d1 đến đảo d2 sao cho số lần ghé đảo trung gian để tiếp thêm xăng là ít nhất.

CHƯƠNG III:

HỆ CHUYÊN GIA VÀ CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

II. GIỚI THIỆU VỀ CÁC HỆ CHUYÊN GIA

1. Hệ chuyên gia là gì ?

Hệ chuyên gia (expert system) là một chương trình cơ sở tri thức được xây dựng để có khả năng giải các bài toán trong một lĩnh vực chuyên môn giống như người chuyên gia. Tổng quát, tri thức của nó được sao chép từ các người chuyên gia trong một lĩnh vực và nó mô phỏng phương pháp và cách diễn đạt của họ.

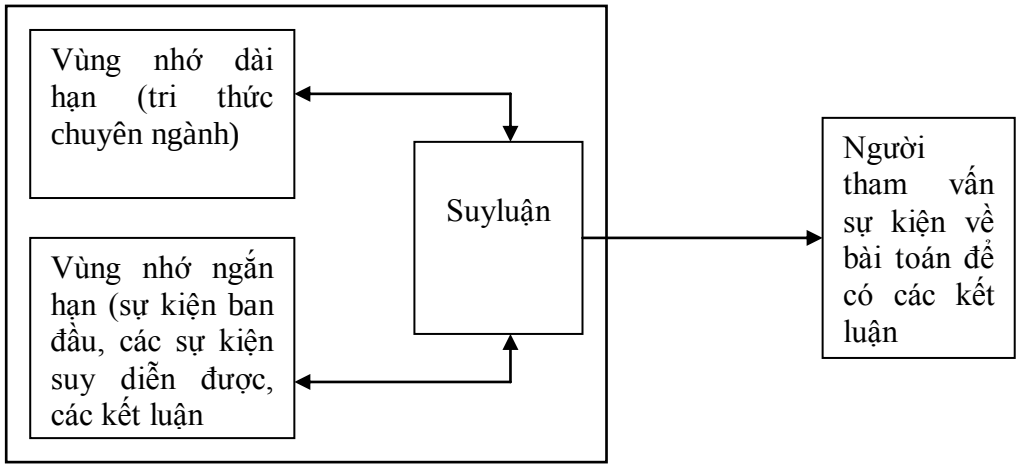
Một hệ chuyên gia có các đặc trưng cơ bản sau :

- Sử dụng tri thức chuyên gia
- Sử dụng kỹ thuật tìm kiếm
- Sử dụng thông tin Heuristics
- Có khả năng suy diễn tri thức mới từ tri thức sẵn có
- Có khả năng xử lý ký hiệu
- Có khả năng tự giải thích, lý giải

Các lĩnh vực sử dụng hệ chuyên gia hiện nay: chẩn đoán, xây dựng phương án, dự báo, thiết kế, phiên dịch, điều khiển, giám sát trạng thái và hướng dẫn.

2. Cấu trúc tổng quát của các hệ chuyên gia

Để hiểu rõ các nguyên lý làm việc của các hệ chuyên gia, ta dựa vào một chuyên gia con người xét về phương diện cơ chế làm việc và cách giải bài toán, minh hoạ ở hình 3.1.



Hình 3.1: *Cơ chế làm việc của một chuyên gia con người*

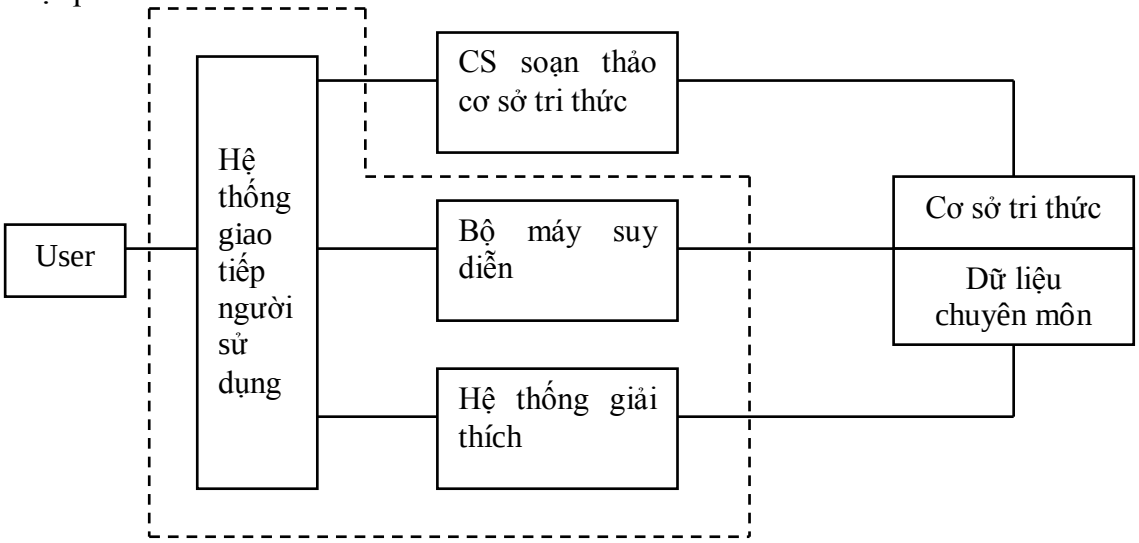
Thông thường người chuyên gia ghi nhớ một cách kỹ lưỡng các tri thức liên quan đến lĩnh vực chuyên môn của mình. Phần tri thức này được lưu trữ trong vùng nhớ dài hạn và đây cũng chính là cơ sở tri thức (tập hợp các quy luật để giải quyết bài toán).

Khi người chuyên gia nhận các sự kiện cần thiết về bài toán cần được giải, các sự kiện này được lưu trữ trong vùng nhớ ngắn hạn. Chuyên gia giải quyết bài toán

bằng cách kết hợp các sự kiện đã thu thập được trong vùng nhớ ngắn hạn với cơ sở tri thức sẵn có trong vùng nhớ dài hạn để suy diễn ra sự kiện mới cho bài toán.

Ứng dụng qui trình này, người chuyên gia suy diễn liên tục các thông tin mới về bài toán và cuối cùng đi đến kết luận của bài toán.

Cấu trúc tổng quát của một hệ chuyên gia gồm có các thành phần cơ bản: người sử dụng, hệ thống giao tiếp người sử dụng, hệ thống giải thích, bộ máy suy diễn, cửa sổ soạn thảo cơ sở tri thức, cơ sở tri thức và dữ liệu chuyên môn được minh hoạ qua sơ đồ hình 3.2 như sau:



Hình 3.2: *Sơ đồ cấu trúc tổng quát của một hệ chuyên gia*

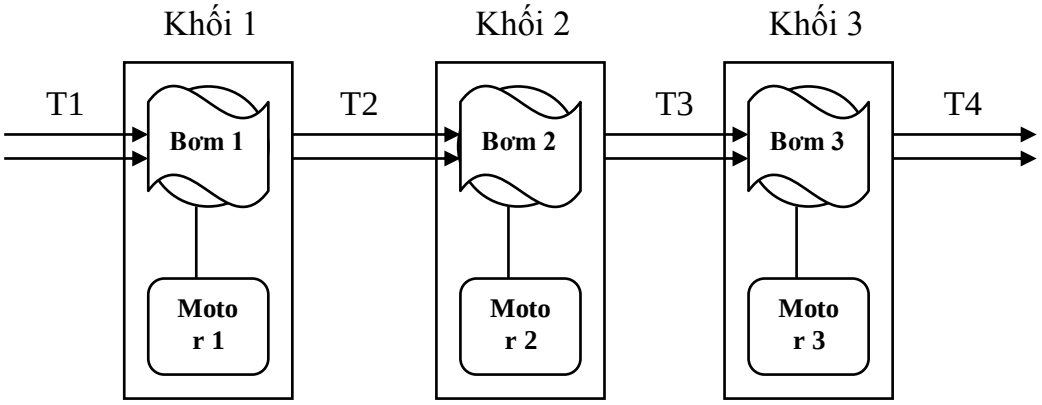
Chức năng cụ thể của từng thành phần trong sơ đồ như sau:

- User: là người sử dụng hệ chuyên gia tham vấn các sự kiện về bài toán để có kết luận.
- Hệ thống giao tiếp người sử dụng: có khả năng chấp nhận mọi thông tin từ người sử dụng và dịch nó sang dạng chấp nhận được để tồn tại trong hệ hoặc chấp nhận thông tin từ hệ và chuyển đổi nó sang dạng mà người sử dụng có thể hiểu được. Mặc nhiên rằng hệ thống giao tiếp này được trang bị bằng một hệ thống xử lý ngôn ngữ tự nhiên cho phép người sử dụng hiểu hệ chuyên gia và ngược lại.
- Cửa sổ soạn thảo cơ sở tri thức: tồn tại trong hệ chuyên gia. Các cửa sổ này có thể truy cập đến hệ thống giải thích và cho phép người lập trình soạn thảo, gỡ rối hoặc bổ sung tri thức mới vào hệ.
- Cơ sở tri thức: Là thành phần quan trọng nhất của một hệ chuyên gia, nó chứa tri thức giải bài toán của một ứng dụng đặc biệt nào đó. Trong một hệ chuyên gia, tri thức này được thể hiện dưới dạng các luật if…then…cụ thể như các luật để giải bài toán bình đựng nước hoặc các luật để dịch chuyển các viên ngói hợp lệ của trò chơi nhiều số…
- Hệ thống giải thích: Đây cũng là một thành phần quan trọng của hệ chuyên gia. Cho phép chương trình giải thích cách lý giải bài toán của nó với người sử dụng. Đây là cách diễn giải quá trình giải bài toán để có các kết luận của hệ, tại sao hệ lại cần đến một vài mảnh thông tin đặc biệt…
- Bộ máy suy diễn: Có chức năng kết hợp các sự kiện bài toán với cơ sở tri thức của bài toán để suy diễn ra các sự kiện mới cho bài toán. Cứ như thế liên tục suy diễn các sự kiện mới hơn cho đến khi đạt đến lời giải của bài toán

- Dữ liệu chuyên môn: hay bộ phận thu thập dữ liệu, tương tự vùng nhớ ngắn hạn của người chuyên gia, hệ chuyên gia phải có khả năng thu thập các sự kiện, các kết luận và các thông tin liên quan khác về bài toán
  Hệ chuyên gia được phát triển dựa trên các ngôn ngữ lập trình Lisp, Prolog, và gần đây phổ biến là ngôn ngữ lập trình C cũng được sử dụng.

3. Hệ chuyên gia nhận dạng và chẩn đoán sự cố

Xây dựng một trạm bơm nước gồm có nhiều khối, mỗi khối gồm một máy bơm và một động cơ.
Áp suất bình thường của các đường ống là:
- Đường ống T1 có áp suất bình thường 50 psi
- Đường ống T2 có áp suất bình thường 100 psi
- Đường ống T3 có áp suất bình thường 150 psi
- Đường ống T4 có áp suất bình thường 200 psi



Hình 3.3: *Sơ đồ trạm bơm nước*

Chỉ số làm việc bình thường của các motor là 1.
Yêu cầu:
- Xây dựng một hệ chuyên gia để phát hiện sự cố, nhận dạng sự cố và chẩn đoán sự cố xảy ra trên trạm. Một hệ chuyên gia được xây dựng dựa trên các kinh nghiệm chuyên gia:
  1. **Phát hiện sự cố:** Để phát hiện sự cố trên trạm, cơ sở luật được thiết kế dựa trên các thông tin:

- Áp suất thấp so với áp suất bình thường của các đường ống
- Chỉ số làm việc thấp so với chỉ số làm việc bình thường của các motor.

Cơ sở luật được xây dựng như sau:

Luật 1: nếu (áp suất T1<50) thì (áp suất T1 là thấp) và (sự cố đã được phát hiện trên trạm)

Luật 2: nếu (áp suất T1>=50) thì (áp suất T1 là bình thường)

Luật 3: nếu (áp suất T2<100) thì (áp suất T2 là thấp) và (sự cố đã được phát hiện trên trạm)

Luật 4: nếu (áp suất T2>=100) thì (áp suất T2 là bình thường)

Luật 5: nếu (áp suất T3<150) thì (áp suất T3 là thấp) và (sự cố đã được phát hiện trên trạm)

Luật 6: nếu (áp suất T3>=150) thì (áp suất T3 là bình thường)

Luật 7: nếu (chỉ số motor 1<1) thì (chỉ số motor 1 hiện có là thấp)

Luật 8: nếu (chỉ số motor 1>=1) thì (chỉ số motor 1 hiện có là bình thường)

Luật 9: nếu (chỉ số motor 2<1) thì (chỉ số motor 2 hiện có là thấp)

Luật 10: nếu (chỉ số motor 2>=1) thì (chỉ số motor 2 hiện có là bình thường)

Luật 11: nếu (chỉ số motor 3<1) thì (chỉ số motor 3 hiện có là thấp)

Luật 12: nếu (chỉ số motor 3>=1) thì (chỉ số motor 3 hiện có là bình thường)

2. Nhận dạng sự cố: Mỗi trạm có nhiều khối, do đó sự cố có thể xảy ra ở khối 1, khối 2, hoặc khối 3. Để nhận dạng chính xác khối có sự cố, cơ sở luật được thiết kế dựa trên các thông tin là

- Áp suất vào của khối là bình thường nhưng áp suất ra của khối là thấp.

Từ đó ta xây dựng cơ sở luật như sau:

Luật 13: Nếu (áp suất T1 là bình thường) và (áp suất T2 là thấp) thì (sự cố xảy ra ở khối 1)

Luật 14: Nếu (áp suất T2 là bình thường) và (áp suất T3 là thấp) thì (sự cố xảy ra ở khối 2)

Luật 15: Nếu (áp suất T3 là bình thường) và (áp suất T4 là thấp) thì (sự cố xảy ra ở khối 3)

3. Chẩn đoán sự cố: Sau khi nhận dạng được chính xác khối có sự cố, vấn đề tiếp theo là ta phải xác định thành phần nào (motor, máy bơm, hay đường ống) trong khối có sự cố?

- Nếu chỉ số làm việc của motor<1 thì sự cố xảy ra bởi motor, mặt khác sự cố gây ra bởi máy bơm hay đường ống.
- Nếu áp suất giữa hai khối không thay đổi có nghĩa là máy bơm không hoạt động thì sự cố gây ra bởi máy bơm.
- Nếu áp suất vào của khối nhỏ hơn áp suất ra của khối thì sự cố gây ra bởi đường ống bị rỉ nứt.

Cơ sở luật được xây dựng như sau để chẩn đoán thành phần trong khối có sự cố:

Luật 16: Nếu (khối 1 có sự cố) và (chỉ số motor 1 hiện có là thấp) thì (sự cố được chẩn đoán là motor 1)

Luật 17: Nếu (khối 2 có sự cố) và (chỉ số motor 2 hiện có là thấp) thì (sự cố được chẩn đoán là motor 2)

Luật 18: Nếu (khối 3 có sự cố) và (chỉ số motor 3 hiện có là thấp) thì (sự cố được chẩn đoán là motor 3)

Luật 19: Nếu (khối 1 có sự cố) và (chỉ số motor 1 hiện có là bình thường) và (áp suất T1=áp suất T2) thì (sự cố được chẩn đoán là máy bơm 1)

Luật 20: Nếu (khối 2 có sự cố) và (chỉ số motor 2 hiện có là bình thường) và (áp suất T2=áp suất T3) thì (sự cố được chẩn đoán là máy bơm 2)

Luật 21: Nếu (khối 3 có sự cố) và (chỉ số motor 3 hiện có là bình thường) và (áp suất T3=áp suất T4) thì (sự cố được chẩn đoán là máy bơm 3)

Luật 22: Nếu (khối 1 có sự cố) và (chỉ số motor 1 hiện có là bình thường) và (áp suất T1 < áp suất T2) thì (sự cố được chẩn đoán là đường ống T2)

Luật 23: Nếu (khối 2 có sự cố) và (chỉ số motor 2 hiện có là bình thường) và (áp suất T2 < áp suất T3) thì (sự cố được chẩn đoán là đường ống T3)

Luật 24: Nếu (khối 3 có sự cố) và (chỉ số motor 3 hiện có là bình thường) và (áp suất T3 < áp suất T4) thì (sự cố được chẩn đoán là đường ống T4)

Hệ chuyên gia phát hiện, nhận dạng và chẩn đoán sự cố theo yêu cầu của vấn đề bài toán trạm bơm nước sẽ làm việc theo cách sau:

- Đầu tiên ta giả sử dữ liệu ban đầu trong bộ nhớ làm việc được cho là:

+ Áp suất T1 = 60
+ Áp suất T2 = 100
+ Áp suất T3 = 100
+ Chỉ số motor 1 = 1
+ Chỉ số motor 2 = 1
+ Chỉ số motor 3 = 1

- Bộ máy suy diễn kết hợp các sự kiện hiện có trong bộ nhớ làm việc với cơ sở luật: luật 2, luật 4, luật 5, luật 8, luật 10, luật 12 sẽ cho ra các sự kiện mới trong bộ nhớ làm việc kế theo đó là:
  + Áp suất T1 là bình thường
  + Áp suất T2 là bình thường
  + Áp suất T3 là thấp
  + Chỉ số motor 1 hiện có là bình thường
  + Chỉ số motor 2 hiện có là bình thường
  + Chỉ số motor 3 hiện có là bình thường
- Cùng với các sự kiện mới này, luật 5 cho biết đã phát hiện có sự cố trên trạm, nhưng chưa biết cụ thể khối nào có sự cố.
- Luật 14 nhận dạng khối có sự cố là khối 2
- Luật 20 chẩn đoán sự cố là máy bơm 2.

III. CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

Cách thể hiện việc mô tả các sự kiện, các quan hệ giữa các đối tượng trong miền của bài toán một cách đầy đủ, chính xác, rõ ràng làm sao cho máy có thể cho ta các lời giải thông minh được xem như là vấn đề biểu diễn tri thức. Tri thức được chia thành ba dạng, mỗi dạng có các phương pháp biểu diễn tri thức tương ứng.

- Tri thức mô tả : có các phương pháp biểu diễn tri thức như :
  - Biểu diễn tri thức nhờ logic vị từ : dùng cho các sự kiện đơn giản
  - Biểu diễn tri thức nhờ mạng ngữ nghĩa : mô tả các mối quan hệ giữa các đối tượng
  - Biểu diễn tri thức nhờ Frames : biểu diễn các thông tin tổng hợp
- Tri thức thủ tục : Được biểu diễn nhờ các luật sản xuất thể hiện dưới dạng if…then…
- Tri thức điều khiển : có các phương pháp biểu diễn như :
  - Các hàm đánh giá heuristics
  - Các luật đánh giá heuristic dạng if…then…

1. Biểu diễn tri thức nhờ logic vị từ :

Toán học logic xuất phát điểm từ tập hợp các câu đơn giản ghi nhận lại các sự kiện đã xảy ra trong một không gian và thời gian xác định nào đó. Tập các câu đơn giản này được gọi là các đề xuất. Có hai loại toán logic đó là logic đề xuất và logic vị từ.

a. Logic đề xuất

Gồm các ký hiệu đề xuất, các ký hiệu chân lý, và các phép toán logic.

- Ký hiệu đề xuất là các chữ cái in hoa A, B, C,…, được sử dụng để biểu diễn các đề xuất hoặc các câu đơn giản mà nội dung của chúng được xác định bằng các ký hiệu chân lý đúng sai.

- Ký hiệu chân lý đó là chữ cái T hoặc F được sử dụng để xác minh nội dung của một đề xuất đúng hoặc sai. Nếu đề xuất đúng thì giá trị chân lý của nó được xác minh là đúng biểu thị bằng ký hiệu chân lý T, ngược lại nếu đề xuất không tương thích thì giá trị chân lý của nó được xác minh là sai biểu thị bằng ký hiệu F.

- Phép toán logic gồm có phép toán liên từ và ■, phép toán giới từ hoặc ■, phép toán phủ định ■, phép toán kéo theo ■, và phép toán tương đương ■.

Vd 3.1 : Cho đề xuất "Nhiệt độ xuống 0^0C vào ngày mai". Đề xuất này có thể được biểu diễn bằng một ký tự đề xuất A đó là : A ="Nhiệt độ xuống 0^0C vào ngày mai".

b. Logic vị từ

Logic vị từ là một phần của toán học logic, nó được mở rộng từ logic đề xuất. Trong logic vị từ cho phép ta xây dựng các vị từ để thực hiện các khả năng truy cập các thành phần của một đề xuất hoặc mô tả các mối quan hệ giữa các đề xuất hoặc thay thế nội dung của một đề xuất. Ngược lại trong logic đề xuất, các ký hiệu A, B, C,…, chỉ sử dụng để biểu diễn các đề xuất mà không có các khả năng trên.

Vd 3.2 : Xem xét đề xuất "Nhiệt độ xuống 0^0C vào ngày mai"

- Trong logic đề xuất, ta có thể sử dụng chữ cái A để biểu diễn đề xuất này theo cách trên, tuy nhiên, trong logic vị từ nhiệt_độ để mô tả mối quan hệ giữa ngày và nhiệt độ như sau : nhiệt_độ(ngày mai, 0^0C).
- Một vị từ gồm :
  - ■ Tên vị từ (trong ví dụ là nhiệt độ)
  - ■ Các ký hiệu vị từ : có thể chứa các biến vị từ hoặc hằng vị từ, hàm vị từ, vị từ.
  - ■ Các ký hiệu chân lý logic T, F để xác minh các biểu thức của một vị từ đúng hoặc sai.
  - ■ Các phép toán logic : liên từ và, giới từ hoặc, phủ định, kéo theo, tương đương.
  - ■ Các vị từ định lượng : Đại lượng ■■ đứng trước biến vị từ để định lượng phạm vi cho biến. ■ : xác định đề xuất đúng cho tất cả mọi giá trị của biến. ■ : xác định đề xuất đúng cho một vài giá trị của biến.

Vd 3.3 : Cho đề xuất "Nhiệt độ xuống 0^0C vào ngày T". Từ đề xuất này ta xây dựng một vị từ nhiệt _độ mô tả quan hệ giữa thời tiết và ngày là : nhiệt_độ(T, 0^0C) với tên vị từ : nhiệt_ độ ; biến vị từ : T ; hằng vị từ : 0^0C.

Vd 3.4 : Cho đề xuất "Tất cả các người đàn ông cuối cùng cũng chết"

Ta có thể biểu diễn đề xuất này bằng logic vị từ với vị từ định lượng ■ như sau :

■X đàn_ông(X) → cuối_cùng_chết(X) / X có thể nhận tất cả các giá trị trong miền.

Cho đề xuất "Chỉ một vài sinh viên thích làm bài tập"

Đề xuất này có thể được biểu diễn bằng logic vị từ với vị từ định lượng ■ :

■X sinh_viên(X) → thích_làm_bài_tập(X) / X tiêu biểu một vài sinh viên trong miền thích làm bài tập.

c. Biểu diễn tri thức nhờ logic vị từ

Biểu diễn tri thức bằng logic vị từ cho phép ta có khả năng truy cập hoặc thay thế các thành phần của biểu diễn, và cũng cho phép ta suy diễn được các tri thức mới từ tri thức sẵn có thông qua các luật suy diễn.

Vd 3.5 : Cho một sự kiện : "Nếu ngày mai nhiệt độ xuống 0^0C, Henry sẽ đi dạo biển".

Sự kiện này có thể được biểu diễn nhờ logic vị từ như sau :

nhiệt_độ(ngày mai, 0^0C) → **đi_dạo**(Henry, biển)

Vd 3.6 : Cho tập các sự kiện :

1. Tất cả các con chó là thú vật
2. Kiki là một con chó
3. Tất cả các thú vật sẽ chết

Tập các sự kiện trên được biểu diễn bằng logic vị từ như sau :

1. X chó(X) → thú_vật(X)
2. chó(kiki)
3. X thú_vật(X) → chết(X)

Muốn chứng minh "Kiki là một thú vật" và "Kiki sẽ chết" từ tập các sự kiện trên ta phải làm thế nào ? Sau khi chúng ta biểu diễn bằng logic vị từ các sự kiện trên chúng ta sử dụng phương pháp hợp giải để giải quyết vấn đề này

2. Sử dụng phương pháp hợp giải trong logic vị từ

Đây là một phương pháp dùng chứng minh các định lý trong logic đề xuất hoặc logic vị từ

Là một phần trong việc tìm kiếm giải quyết bài toán trong lĩnh vực trí tuệ nhân tạo

Hợp giải là phương pháp chứng minh phản đề.

Các bước thực hiện chứng minh bằng hợp giải như sau :

Bước 1 : chuyển đổi các tiên đề đã được biểu diễn bằng logic vị từ sang dạng mệnh đề có dạng tổng quát : $a_1 \vee a_2 \vee a_3 \vee \ldots \vee a_n$, sử dụng các phép toán tương đương :

A → B	A B
(P)	P
(A B)	A B
(A B)	A B
X P(X)	X P(X)
X P(X)	X P(X)

Các vị từ định lượng được đưa về đứng trước mệnh đề để loại bỏ trước khi tiến hành chứng minh

Bước 2 : Muốn chứng minh tiên đề S, phải phủ định S và cộng nó vào cơ sở dữ liệu.

Bước 3 : Chọn cặp mệnh đề cha trong cơ sở tri thức, một có chứa R và một mệnh đề cha khác có chứa ¬R. Hợp giải hai mệnh đề này để sản xuất ra mệnh đề mới bằng cách khử bỏ cặp R và ¬R từ hai mệnh đề cha, vì chúng mâu thuẫn với nhau, phần còn lại của hai mệnh đề cha được hợp nhau tạo thành một mệnh đề mới. Cộng mệnh đề mới này vào cơ sở tri thức.

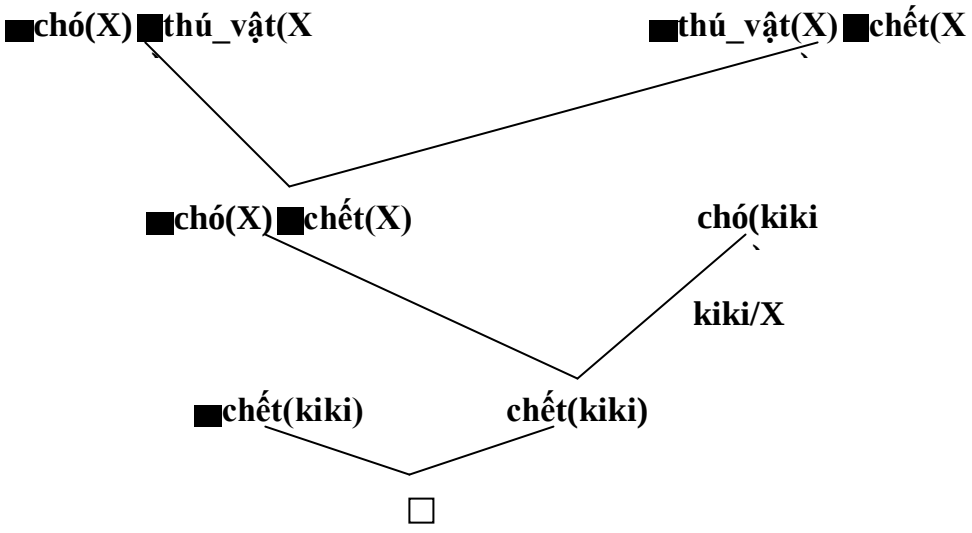
Bước 4 : Lặp lại thủ tục ở bước 3 cho đến khi mệnh đề mới phát sinh là một mệnh đề rỗng lúc đó tiên đề đã được chứng minh xong.

Vd 3.7 : Sử dụng cơ sở tri thức ở ví dụ 3.6, chứng minh "Kiki sẽ chết".

- Trước hết ta phải chuyển đổi tất cả các tiên đề sang dạng mệnh đề
- Tiếp theo, phủ định tiên đề "Kiki sẽ chết" thành "Kiki sẽ không chết". Tiên đề này được biểu diễn bằng logic vị từ ¬chết(kiki).
- Cộng mệnh đề ¬chết(kiki) vào cơ sở tri thức.

1. ¬chó(X) ∨ thú_vật(X)
2. chó(kiki)
3. ¬thú_vật(X) ∨ chết(X)
4. ¬chết(kiki)

Tiến trình thực hiện hợp giải như sau : chọn hai mệnh đề cha, một có chứa R và một có chứa ¬R từ cơ sở tri thức, thực hiện như bước 3 tiến hành khử bỏ chúng, còn lại của hai mệnh đề cha hợp nhau tạo thành mệnh đề mới. Thực hiện lặp lại quy trình này (bước 4) cho đến khi xuất hiện một mệnh đề mới là rỗng được ký hiệu bằng □. Theo dõi hình 3.4 minh hoạ bên dưới để nắm rõ phương pháp này.



Hình 3.4

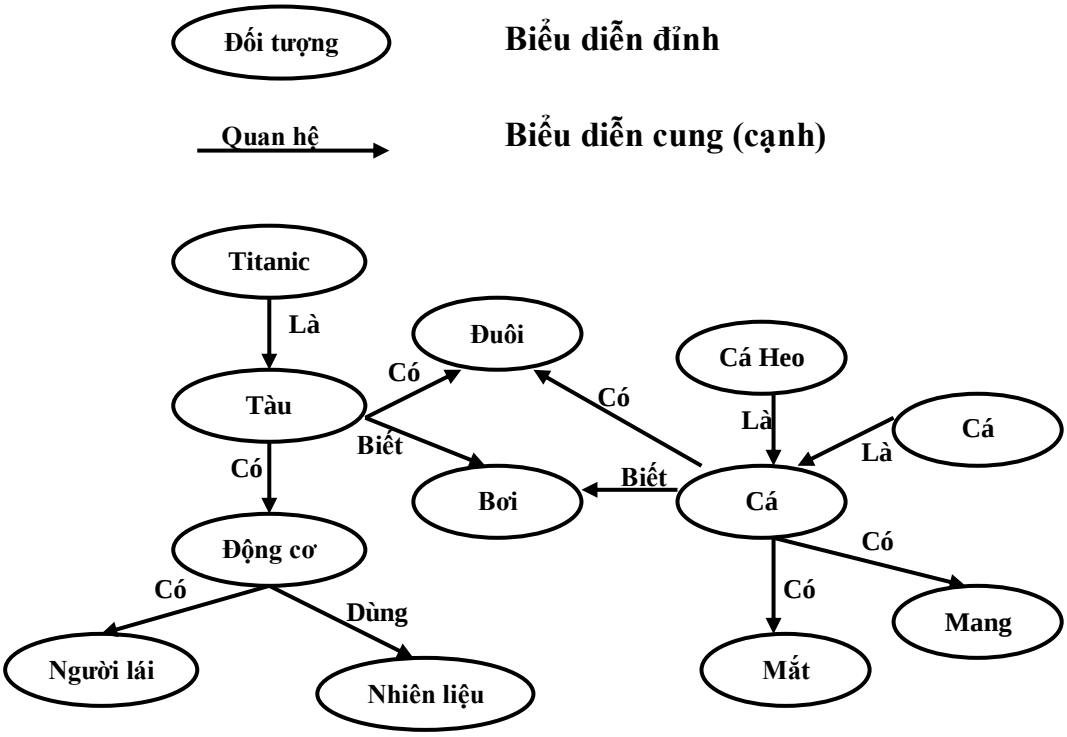
3. Biểu diễn tri thức nhờ mạng ngữ nghĩa

Mạng ngữ nghĩa là một dạng đồ thị có hướng với tập các đỉnh được biểu thị bằng các elip chứa các thông tin mô tả đối tượng và tập các cạnh được biểu thị bằng các mũi tên được gọi là các cung, chứa các thông tin mô tả các mối quan hệ giữa các đối tượng

Vd 3.8 : Xem xét tập các sự kiện về máy bay và chim :

Titanic là tàu;	Cá heo là cá ;
Tàu có đuôi ;	Cá mập là cá ;
Tàu có động cơ ;	Cá có đuôi ;
Động cơ dùng nhiên liệu ;	Cá có mang cá ;
Động cơ có người lái ;	Cá có mắt ;
Tàu biết bơi (di chuyển được trên mặt nước) ;	Cá biết bơi ;

Sử dụng mạng ngữ nghĩa để biểu diễn các sự kiện trên như hình 3.5. Thông tin được biểu diễn nhờ mạng ngữ nghĩa cũng được xem như là một cấu trúc dữ liệu được mô tả bằng đồ thị.



Hình 3.5

Cách biểu diễn dùng mạng ngữ nghĩa như hình 3.5 giúp người lập trình có khả năng phân tích và thiết kế bộ các luật cho cơ sở tri thức sao cho đủ và có hiệu lực để xử lý mọi tình huống đặt ra trong một cơ sở dữ liệu.

Chẳng hạn chúng ta có thể xây dựng bộ các luật nhận dạng các đối tượng là cá hay tàu :

Luật 1 : Nếu thuộc tính của đối tượng có đuôi thì đối tượng bơi là cá hoặc tàu.

Luật 2 : Nếu thuộc tính của đối tượng có mắt, mang, đuôi thì đối tượng là cá

Luật 3 : Nếu thuộc tính của đối tượng có đuôi, và động cơ thì đối tượng là tàu

Vd 3.9 : Xem xét thông tin về các hình đa giác được cho với tập các sự kiện như sau :

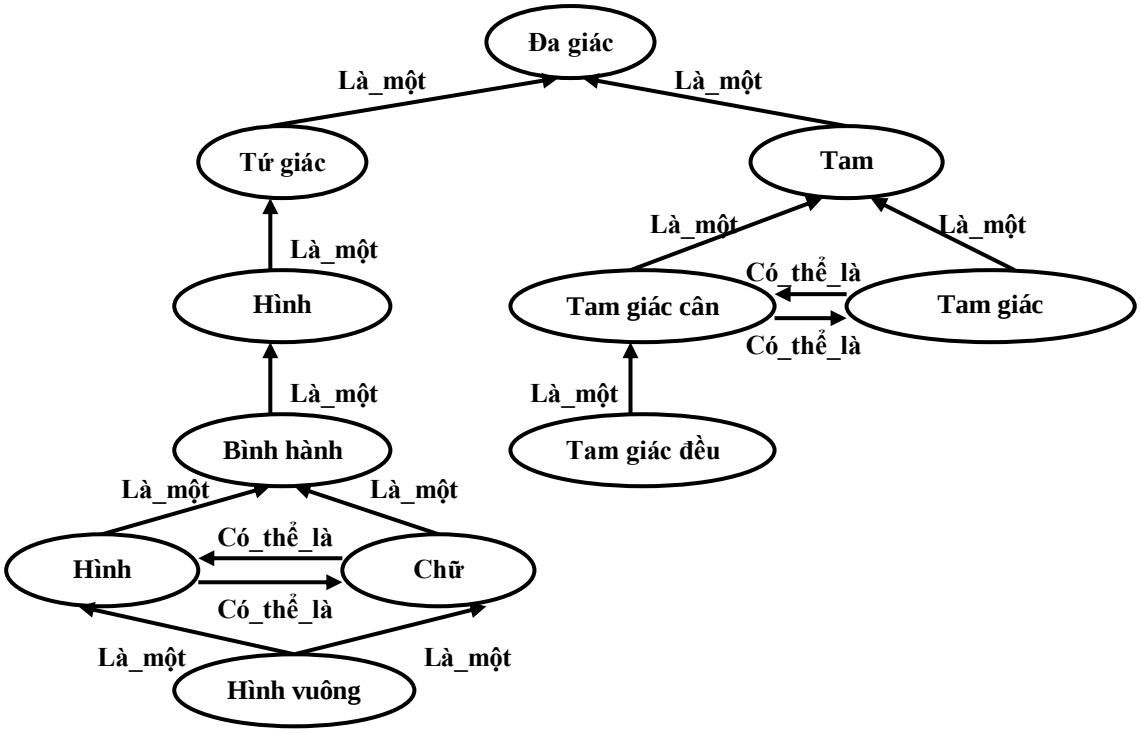
Hình vuông là một hình thoi	Tam giác đều là một tam giác cân
Hình vuông là một hình chữ nhật	Tam giác cân có thể là tam giác vuông

Hình chữ nhật có thể là một hình thoi
Hình thoi có thể là một hình chữ nhật
Hình thoi là một hình bình hành
Hình chữ nhật là một hình bình hành
Hình bình hành là một hình thang
Hình thang là một tứ giác
Tứ giác là một đa giác
Tam giác vuông có thể là tam giác cân
Tam giác cân là một tam giác
Tam giác vuông là một tam giác
Tam giác là một đa giác

Sử dụng mạng ngữ nghĩa ta có thể biểu diễn tập các sự kiện trên như sơ đồ hình 3.6

Các đỉnh của mạng là các đối tượng đa giác

Các cung là các quan hệ giữa các đa giác



Hình 3.6

4. Biểu diễn tri thức nhờ frames

Khung (frames) là sự mở rộng của mạng ngữ nghĩa, trong đó mỗi đỉnh của mạng là một cấu trúc dữ liệu hay còn gọi là một khung.

Mỗi khung có nhiều thuộc tính (properties), mỗi thuộc tính có nhiều giá trị (value). Ứng với mỗi thuộc tính của khung là một Slot và mỗi Slot có tên riêng của nó đó là tên của thuộc tính. Cấu trúc tổng quát của một khung có dạng như sau :

Frame <Tên frame>

Slot : <Tên property thứ nhất>

Value: <Các values của property thứ nhất>

…

Slot: <Tên property thứ n>

Value: <Các values của property thứ n>

Thông tin chứa trong các slot có thể là thông tin nhận dạng khung, thông tin quan hệ giữa khung này với nhiều khung khác, thông tin mô tả các thành phần thích hợp của một khung, thông tin thủ tục, thông tin mặc định khung, thông tin bổ sung thành phần mới của khung.

Phương pháp biểu diễn tri thức có cấu trúc nhờ khung được đánh giá là một phương pháp hữu hiệu bởi vì nó cung cấp một số tuỳ chọn và các kế thừa thích hợp như sau:

- Cho phép thay thế, truy cập đến mỗi thành phần của khung
- Cho phép xây dựng cấu trúc dữ liệu ở hệ phân cấp
- Cho phép các đối tượng ở cùng cấp có quyền sử dụng chung các thuộc tính ở cấp đó
- Cho phép thừa hưởng luật kế thừa trong hệ phân cấp (tính bắc cầu trong toán học)

if A **is_a** B ***and*** B **is_a** C ***then*** A **is_a** C

Vd 3.10: Xem xét ví dụ 3.9 (hình 3.6) cơ sở tri thức được biểu diễn bằng mạng ngữ nghĩa. Bây giờ ta sử dụng khung (frame) để biểu diễn thông tin về các hình đa giác:

Frame: đa_giác

Slot: hình_nhiều_cạnh_khép_kín

Value: true

Frame: tứ_giác

Slot: là_một

Value: đa_giác

Slot: số_cạnh

Value: 4

Frame: hình_thang

Slot: là_một

Value: tứ_giác

Slot: số_cặp_cạnh_song_song

Value: 1

Frame: hình_bình_hành

Slot: là_một

Value: hình_thang

Slot: số_cặp_cạnh_song_song

Value: 2

Frame: hình_thoi

Slot: là_một

Value: hình_bình_hành

Slot: có_thể_là

Value: hình_chữ_nhật

Slot: số_cạnh_bằng_nhau

Value: 4

Frame: hình_chữ_nhật

Slot: là_một

Value: hình_bình_hành

Slot: có_thể_là

Value: hình_thoi

Slot: số_góc_vuông

Value: 4

Frame: hình_vuông

Slot: là_một

Value: hình_thoi, chữ_nhật

Frame: tam_giác

Slot: là_một

Value: đa_giác

Slot: số_cạnh

Value: 3

Frame: tam_giác_vuông

Slot: là_một

Value: tam_giác

Slot: có_thể_là

Value: tam_giác_cân

Slot: số_góc_vuông

Value: 1

Frame: tam_giác_cân

Slot: là_một

Value: tam_giác

Slot: có_thể_là

Value: tam_giác_vuông

Slot: số_cạnh_bằng_nhau

Value: 2

Frame: tam_giác_đều

Slot: là_một

Value: tam_giác_cân

Slot: số_cạnh_bằng_nhau

Value: 3

Ta có thể thiết kế luật thừa hưởng kế thừa trong hệ phân cấp của các hình đa giác : if A is_a B and B is_a C then A is_a C

Nghĩa là, nếu cho A = hình vuông ; B = hình chữ nhật; C = hình bình hành; thì luật kế thừa trên sẽ cho ta kết luận: **hình vuông là một hình bình hành.**

Vd 3.11: Xây dựng hệ chuyên gia giám sát và điều khiển nhờ khung

Xét một ngôi nhà gồm 5 phòng: phòng khách, phòng ăn, phòng ngủ, phòng đọc sách, phòng sinh hoạt của gia đình. Mỗi phòng trang bị một điều hoà, một lò sưởi và một nhiệt kế.

Nhiệt kế gồm 2 chế độ: **air** và **heat**. Chế độ của nhiệt kế sẽ quyết định **on** hay **off** máy điều hoà và lò sưởi:

- Nếu thiết lập chế độ cho nhiệt kế là **air** thì hệ thống điều khiển máy điều hoà là **on** hoặc **off.**
- Nếu nhiệt kế được thiết lập là **heat** thì hệ thống điều khiển lò sưởi **on** hoặc **off**

Xây dựng 5 khung để tổ chức dữ liệu: khung nhà, khung phòng, khung nhiệt kế, khung máy điều hoà và khung lò sưởi.

- Khung nhà: có tên thuộc tính là phòng; các giá trị của nó là phòng khách, phòng ăn, phòng ngủ, phòng đọc sách, phòng sinh hoạt
- Khung phòng: có tên thuộc tính là lò sưởi, máy điều hoà, nhiệt kế và người; các giá trị tương ứng với các thuộc tính là tên các lò sưởi, tên các máy điều hoà, tên các nhiệt kế và có người hay không có người.
- Khung nhiệt kế: có tên các thuộc tính của nó là máy điều hoà, lò sưởi, chế độ nhiệt kế, nhiệt độ đặt, nhiệt độ môi trường và phòng.
- Khung máy điều hoà: có tên các thuộc tính đó là phòng, trạng thái, nhiệt kế.
- Khung lò sưởi: có tên các thuộc tính đó là phòng, trạng thái, nhiệt kế.

frame: nhà

slot: phòng

value:…

frame: phòng

slot: lò sưởi

value:…

slot: máy điều hoà

value:…

slot: nhiệt kế

value:…

slot: người

value: true/false

frame: nhiệt kế

slot: máy điều hoà
value:…
slot: lò sưởi
value:…
slot: chế độ nhiệt kế
value: heat
slot: nhiệt độ đặt
value: 27
slot: nhiệt độ môi trường
value: 21
slot: phòng
value:…
frame: máy điều hoà
slot: phòng
value:…
slot: trạng thái
value: off
slot: nhiệt kế
value:…
frame: lò sưởi
slot: phòng
value:…
slot: trạng thái
value: off
slot: nhiệt kế
value:…

Cơ sở luật điều khiển của hệ chuyên gia giám sát và điều khiển nhiệt độ môi trường trong các phòng được thiết kế để điều khiển các thiết bị lò sưởi và máy điều hoà đóng (**off**) hoặc mở (**on**) dựa trên dữ liệu của các cấu trúc khung gồm các luật:

Luật 1: nếu (nhiệt độ môi trường < nhiệt độ đặt) và (trạng thái lò sưởi = **off**) và (chế độ của nhiệt kế = **heat**) và (phòng có người) thì (mở lò sưởi **on**)

Luật 2: nếu (nhiệt độ môi trường < (nhiệt độ đặt – 5)) và (trạng thái lò sưởi = **off**) và (chế độ của nhiệt kế = **heat**) và (phòng không có người) thì (mở lò sưởi **on**)

Luật 3: nếu (nhiệt độ môi trường >= nhiệt độ đặt) và (trạng thái lò sưởi = **on**) và (chế độ nhiệt kế = **heat**) và (phòng có người) thì (đóng lò sưởi **off**)

Luật 4: nếu (nhiệt độ môi trường >= (nhiệt độ đặt – 5)) và (trạng thái lò sưởi = **on**) và (chế độ nhiệt kế = **heat**) và (phòng không có người) thì (đóng lò sưởi **off**)

Luật 5: nếu (nhiệt độ môi trường > nhiệt độ đặt) và (trạng thái máy điều hoà = **off**) và (chế độ nhiệt kế = **air**) và (phòng có người) thì (mở máy điều hoà **on**)

Luật 6: nếu (nhiệt độ môi trường < (nhiệt độ đặt + 5)) và (trạng thái máy điều hoà = **off**) và (chế độ nhiệt kế = **air**) và (phòng không có người) thì (mở máy điều hoà **on**)

Luật 7: nếu (nhiệt độ môi trường <= nhiệt độ đặt) và (trạng thái máy điều hoà = **on**) và (chế độ nhiệt kế = **air**) và (phòng có người) thì (đóng máy điều hoà **off**)

Luật 8: nếu (nhiệt độ môi trường <= (nhiệt độ đặt + 5)) và (trạng thái máy điều hoà = **on**) và (chế độ nhiệt kế = **air**) và (phòng không có người) thì (đóng máy điều hoà **off**

III. BÀI TẬP

Bài 1. Biểu diễn các tri thức sau dưới dạng logic mệnh đề

a. Trong tam giác vuông, tổng bình phương chiều dài hai cạnh góc vuông bằng bình phương chiều dài cạnh huyền

b. Một số nguyên dương có chữ số hàng đơn vị bằng 5 thì số đó chia hết cho 5

c. Nếu x là số lẻ và bình phương của x tận cùng bằng 1 thì x tận cùng bằng 1 hoặc bằng 9

d. Trong tam giác vuông, chiều dài của đường trung tuyến xuất phát từ góc vuông bằng nữa chiều dài của cạnh huyền

Bài 2. Ta có cơ sở tri thức của hệ chuyên gia về bệnh cảm cúm như sau:

1) "Nếu bệnh nhân rát họng và viêm nhiễm thì viêm họng và đi chữa họng"
2) "Nếu thân nhiệt >370 thì sốt"
3) " Nếu ốm trên 7 ngày và sốt thì viêm nhiễm"
4) "Nếu sốt và ho và kèm theo khó thở hoặc kèm theo tếng ran thì viêm phổi"

a. Hãy biểu diễn các tri thức trên dưới dạng logic mệnh đề.

b. Có một bệnh nhân khai : "Thân nhiệt > 370 " và "ốm trên 7 ngày". Hãy chứng minh bệnh nhân này bị "viêm nhiễm".

Bài 3. Biểu diễn các tri thức sau dưới dạng logic vị từ

a. Bất kỳ người nào cũng có cha mẹ

b. Mọi số nguyên tố lớn hơn 2 đều là số lẻ

c. Chuồn chuồn bay thấp thì mưa

Bài 4. Giả sử chúng ta biết các thông tin sau đây:

a. Mọi người đếu chết

b. Mọi phụ nữ đều chết

c. Thần thánh không chết

d. Tất cả cả những người bệnh phải được điều trị

e. Lix là phụ nữ

f. Carol là phụ nữ

g. Kelly là phụ nữ

h. Socrate là người

i. Zeus là thần thánh

k. Socrate bị bệnh

Dùng phương pháp hợp giải để có thể suy ra được Socrate có được điều trị hay không?

CHƯƠNG IV:

MÁY HỌC (MACHINE LEARNING)

I. MÁY HỌC LÀ GÌ ?

Chúng ta có thể học bằng nhiều cách :

- Nhớ một vài điều gì đó cần thiết
- Học các sự kiện thông qua những quá trình quan sát và khám phá
- Sự phát triển của những kỹ năng vận động hoặc nhận thức thông qua quá trình luyện tập, thực hành chẳng hạn sửa đổi, cải tiến những ý nghĩ sai để có những ý nghĩ đúng…
- Sự tổ chức, sắp xếp những tri thức mới được thu thập vào những khái niệm đại diện hoặc thể hiện tổng quát và hữu hiệu
- Được chỉ đạo trực tiếp bởi một vài sự kiện
- Qua các ví dụ
- Qua lời giảng của giáo viên…

Tất cả các phương pháp học đều nhằm vào mục tiêu thu thập và xử lý tri thức mới để thích nghi với tình huống mới. Herb Simon_ một nhân vật đóng vai trò lịch sử quan trọng trong lĩnh vực trí tuệ nhân tạo đã định nghĩa khái niệm "học" như thế này : *Việc học biểu thị sự thay đổi trong hệ thống thích nghi theo kiểu nó có thể cho phép hệ thống thực hiện được những công việc hoặc những công việc được rút ra sẽ hữu hiệu hơn và ngày càng hữu hiệu hơn trong những lần tiếp theo.* Hay nói cách khác tri thức của con người có thể được cải tiến là nhờ thông qua việc học.

Tương tự như việc học của con người, chúng ta phải xây dựng các chương trình cho máy sao cho nó có khả năng học để thu thập được tri thức mới và biết cách xử lý tri thức mới để thích nghi với tình huống mới.

Có nhiều phương pháp học cụ thể như :

- Học nhờ các mạng neuron : là phương pháp học thay đổi cấu trúc bên trong của mạng hoặc thay đổi các trọng số kết nối giữa các neuron trong mạng.
- Học ký ức (rote learning) : là phương pháp học bằng cách sử dụng một vài bộ nhớ để ghi nhận lại dữ liệu và đem ra sử dụng trong các tình huống tương tự.
- Học nhờ lấy lời cố vấn (advice taking) : là phương pháp học nhờ thông tin cố vấn heuristic.
- Học mẫu từ các ví dụ (concept learning) : là phương pháp học từ các mẫu ví dụ hoặc các mẫu bằng chứng. Nó được chia thành bốn loại phương pháp sau :
    - Học giám sát (supervised learning) : Cho một tập hợp những ví dụ là những giả thiết đầu vào/đầu ra của một số đối tượng nào đấy. Tìm luật để tiên đoán được dữ liệu đầu ra của đối tượng tương ứng với các giả thiết đầu vào mới.
    - Học theo phân nhóm (clustering learning) : Cho một tập những ví dụ, nhưng không xác định rõ chúng cụ thể là thuộc nhóm nào. Công việc là phải phân nhóm những ví dụ vào những nhóm cụ thể.

- Học củng cố (reinforcement learning) : Đây là phương pháp học những kỹ năng vận động. Chúng ta phải thử mọi khả năng có thể của một vấn đề, từ đó rút ra kết quả là thất bại hay thành công. Phương pháp học này giúp cho ta qua thực hành chọn lựa những hành động, những cách thử theo một con đường nào đó để đạt được nhiều kết quả thành công nhất.
- Học chức năng (function learning) :Cho một tập hợp những ví dụ là dữ liệu đầu vào/đầu ra. Tìm một hàm có chức năng biểu diễn được những mối quan hệ giữa chúng. Vd : Học để chẩn đoán bệnh, chúng ta phải xác định được một hàm chức năng diễn tả mối quan hệ giữa những kết quả kiểm tra trong phòng thí nghiệm với việc phân loại bệnh.

- Học nhờ lập luận tương tự (learning by analogy) : là phương pháp học bằng cách sử dụng quy trình đã giải một bài toán nào đó để giải các bài toán tương tự.

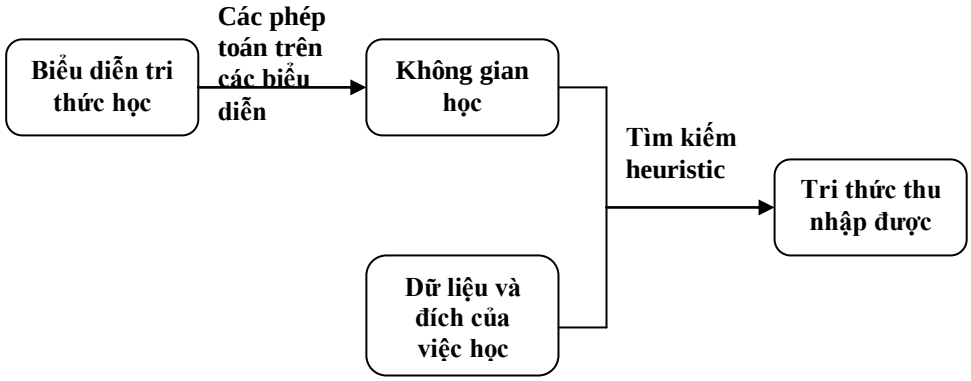
Các phương pháp học máy đã thành công trong một số lĩnh vực với những ứng dụng đa dạng như sau :

- Ước định được mức độ rủi ro của cho vay
- Phát hiện ra được sự gian lận thẻ tín dụng
- Liệt kê danh mục những hình ảnh thiên văn học
- Hướng một xe hơi tự động di chuyển được…

II. KHUNG LÀM VIỆC CHO VIỆC HỌC

Để thiết kế một chương trình cho máy có khả năng học, cần chú ý đến một số thành phần như biểu diễn tri thức học, các phép toán trên các biểu diễn, không gian học, dữ liệu học, các đích của việc học, tìm kiếm heuristic.

Tham khảo sơ đồ dưới đây :



Hình 4.1

1. Dữ liệu và đích của công việc học

Điều mà cần xác định đầu tiên trong vấn đề học đó là đích của việc học và dữ liệu nó cho phép được học.

- Trong chế độ học giám sát, các giải thuật học mẫu sử dụng dữ liệu của việc học đó là tập các mẫu huấn luyện. Có hai loại mẫu huấn luyện : mẫu huấn luyện dương và mẫu huấn luyện âm. Đích của việc học là suy diễn một định nghĩa tổng quát sao cho nó sẽ cho phép người học nhận dạng được các mẫu huấn luyện tương lai của lớp.

- Trong chế độ học không giám sát, các giải thuật học sử dụng dữ liệu từ các mẫu huấn luyện không phân lớp và đích của việc học là khám phá ra lớp cung cấp một vài thông tin bổ ích cho người học.

2. Biểu diễn tri thức học

Sử dụng phương pháp biểu diễn tri thức nhờ logic vị từ và phương pháp biểu diễn tri thức nhờ khung.

3. Tập các phép toán

Cho tập các mẫu huấn luyện, người học phải xây dựng luật học sao cho thoả mãn các đích của việc học. Luật học này có thể là một trong các phép toán cụ thể như phép toán tổng quát hoá hoặc đặc trưng hoá các biểu thức ký hiệu, phép toán hiệu chỉnh các trọng số trong một mạng neuron.

4. Không gian học

Tri thức được biểu diễn dưới dạng ngôn ngữ logic vị từ hoặc khung được gọi là ngôn ngữ biểu diễn kết hợp với các luật học định nghĩa một không gian gọi là không gian học. Qua không gian này người học tiến hành tìm kiếm để tìm ra mẫu mong muốn.

5. Tìm kiếm heuristic

Là giải thuật học tìm kiếm với thông tin đánh giá heuristic cho phép các chương trình học đảm bảo hướng tìm kiếm và thứ tự tìm kiếm, cũng như cách sử dụng các mẫu huấn luyện hiện có và các heuristic để tìm kiếm có hiệu quả cao.

IV. MỘT SỐ GIẢI THUẬT HỌC

1. Phép toán tổng quát hoá

Để định nghĩa một không gian học thường sử dụng các phép toán tổng quát hoá và đặc trưng hoá.

- Trong việc học máy, phép toán tổng quát hoá thường được sử dụng là phép toán thay thế các hằng số với các biến số, phép toán bổ sung thêm biểu thức giới từ **hoặc**, phép toán loại bỏ biểu thức liên từ **và**, phép toán thay thế theo bản chất cha của nó trong hệ phân cấp.

Vd 4.1 : Chẳng hạn cho biểu diễn : **màu(đĩa_CD, bạc)**

Ta có thể biểu diễn tổng quát hoá hơn bằng việc thay thế hằng số **đĩa_CD** với biến số X, ta sẽ được biểu diễn : màu(X, bạc)

Vd 4.2 : Cho biểu diễn : **hình(X, tròn) ■ đường_kính(X, Y) ■ màu(X, bạc)**.

Ta loại bỏ một liên từ **và** đi kèm với biểu thức **đường_kính(X, Y)** khỏi biểu diễn, ta được một biểu diễn tổng quát hoá hơn :

hình(X, tròn) ■ màu(X, bạc).

Vd 4.3 : Cho biểu diễn : **màu(X, bạc)**

Ta có thể thay thế bản chất theo cha của nó trong hệ phân cấp đó là biến **màu_nguyên_thuỷ** thuộc về cấp màu bạc, khi đó ta sẽ thay thế **màu_nguyên_thuỷ** với **bạc** để biểu diễn tổng quát hoá hơn :

màu(X,màu_nguyên_thuỷ)

Vd 4.4 : Cho biểu diễn : **hình(X, tròn) ■ đường_kính(X, Y) ■ màu(X, bạc)**

Nếu bổ sung thêm một giới từ **hoặc** với biểu thức **màu(X, đỏ)** vào biểu diễn, sẽ được một biểu diễn tổng quát hoá hơn :

hình(X, tròn) ■ đường_kính(X, Y) ■ màu(X, bạc) ■ màu(X, đỏ)

2. Giải thuật học hướng đặc trưng đến tổng quát hoá

Giải thuật học hướng đặc trưng đến tổng quát hoá được sử dụng để tìm kiếm mẫu mong muốn học trong không gian học.

Dữ liệu huấn luyện được sử dụng trong giải thuật gồm hai tập mẫu huấn luyện P và N, với P là tập các mẫu huấn luyện dương, N là tập các mẫu huấn luyện âm. Giải thuật cũng được trang bị bằng một danh sách S chứa các mẫu đích của việc học.

Ban đầu cho S chứa một mẫu đặc trưng nhất trong không gian học .

Sau đó giải thuật sẽ tổng quát hoá S cho đến khi nó tìm thấy một mẫu mong muốn thì dừng.

Begin

Đầu tiên cho S chứa mẫu huấn luyện dương thứ nhất

N là tập các mẫu huấn luyện âm

For mỗi mẫu huấn luyện dương p

Begin

For ■s ■ S, nếu s không hợp với p, thay thế s với các thành phần tổng quát đặc trưng hoá nhất của nó mà hợp với p

Huỷ bỏ từ S tất cả các đích tổng quát hơn một vài đích khác trong S

Huỷ bỏ từ S tất cả các đích mà hợp với một mẫu huấn luyện âm được giám sát trước đó trong N

End

For ■ mẫu huấn luyện âm n

Begin

Huỷ bỏ tất cả các thành viên của S mà hợp với n

Cộng n vào N để kiểm tra các đích tương lai cho việc tổng quát hoá

End

End

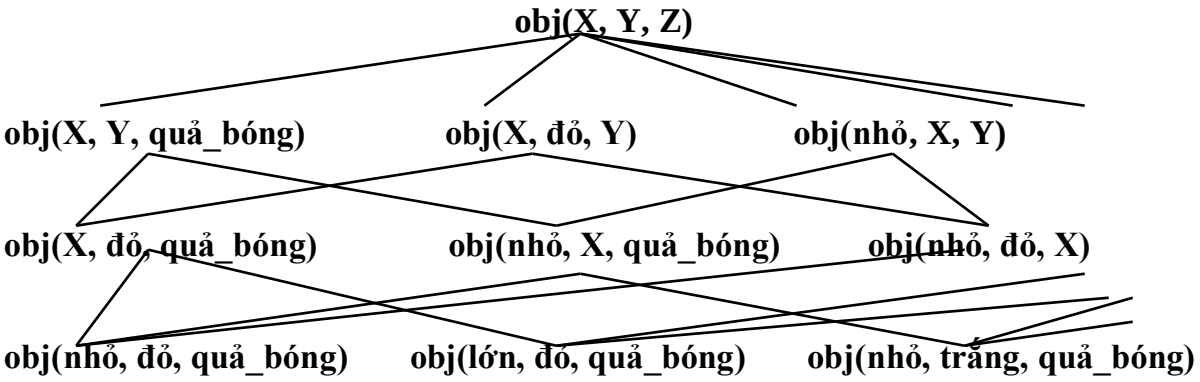
Vd 4.5: Sử dụng giải thuật học hướng đặc trưng đến tổng quát hoá để học nhận dạng đối tượng là quả bóng.

Dữ liệu huấn luyện cho giải thuật học để học nhận dạng quả bóng gồm có tập các mẫu huấn luyện dương P và tập các mẫu huấn luyện âm N được xây dựng như sau:

P = {obj(nhỏ, đỏ, quả_bóng), obj(lớn, đỏ, quả_bóng), obj(nhỏ, trắng, quả_bóng), obj(lớn, trắng, quả_bóng), obj(nhỏ, xanh, quả_bóng), obj(lớn, xanh, quả_bóng)}

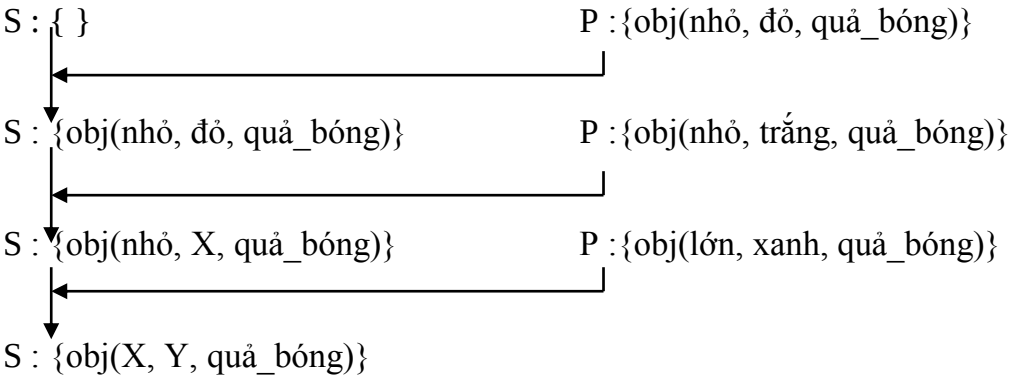
N = {obj(nhỏ, đỏ, thước_kẻ), obj(lớn, đỏ, thước_kẻ), obj(nhỏ, trắng, thước_kẻ), obj(lớn, trắng, thước_kẻ), obj(nhỏ, xanh, thước_kẻ), obj(lớn, xanh, thước_kẻ),

obj(nhỏ, đỏ, khối_trụ), obj(lớn, đỏ, khối_trụ), obj(nhỏ, trắng, khối_trụ), obj(lớn, trắng, khối_trụ), obj(nhỏ, xanh, khối_trụ), obj(lớn, xanh, khối_trụ)}



Hình 4.2

Quá trình học của giải thuật học hướng đặc trưng đến tổng quát hoá tìm kiếm qua không gian học Hình 4.2 một mẫu mong muốn về quả bóng như Hình 4.3



Hình 4.3

3. Giải thuật học hướng tổng quát đến đặc trưng hoá

Giải thuật học hướng tổng quát đến đặc trưng hoá sử dụng các tập mẫu huấn luyện dương P và âm N tìm kiếm qua không gian học để tìm ra một mẫu mong muốn. Giải thuật được trang bị một danh sách G chứa các mẫu tổng quát hoá đặc trưng cực đại_là mẫu bao trùm tất cả các mẫu huấn luyện dương và không bao trùm các mẫu huấn luyện âm.

Ban đầu cho G chứa một mẫu tổng quát hoá nhất trong không gian học.

Sau đó giải thuật đặc trưng hoá G cho đến khi nó tìm thấy một mẫu mong muốn thì dừng.

Begin

Ban đầu cho G chứa mẫu tổng quát nhất trong không gian học

P là tập các mẫu huấn luyện dương

For mỗi mẫu huấn luyện âm n

Begin

For mỗi g ■ G mà hợp với n, thay thế g với các thành phần đặc trưng tổng quát hoá nhất của nó mà không hợp với n

Huỷ bỏ từ G tất cả các đích đặc trưng hơn một vài đích khác trong G

Huỷ bỏ từ G tất cả các đích mà không hợp với một vài mẫu huấn luyện dương trong P

End

For mỗi mẫu huấn luyện dương p

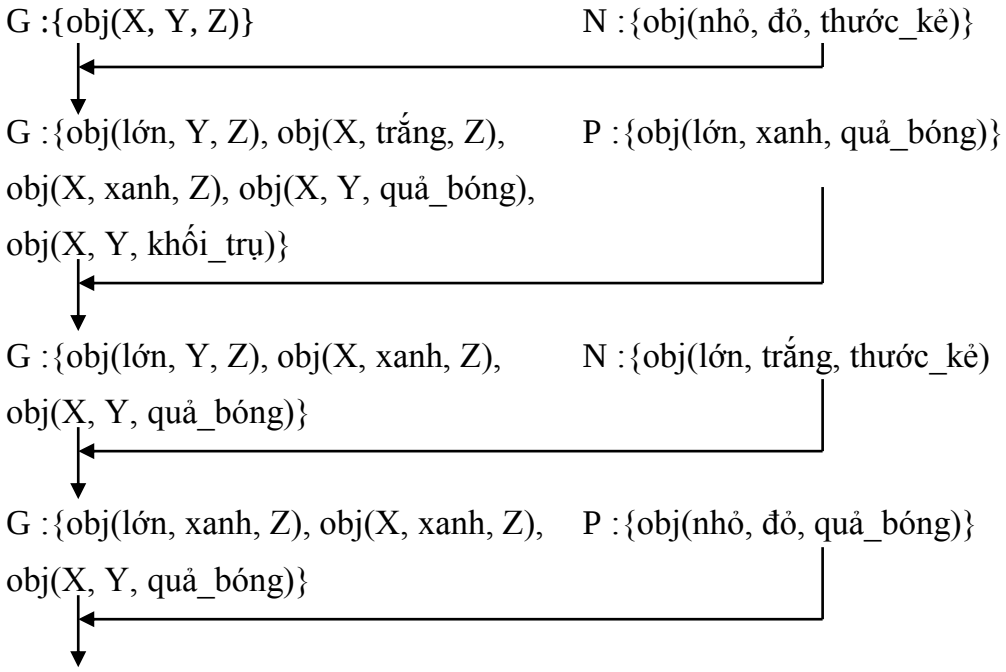
Begin

Huỷ bỏ từ G tất cả các đích mà không hợp với p ; cộng p vào P

End

End

Vd 4.6 : Sử dụng dữ liệu huấn luyện của ví dụ 4.5, quá trình học của giải thuật hướng tổng quát đến đặc trưng hoá tìm kiếm qua không gian học Hình 4.2 cho kết quả như sau :



Hình 4.4

4. Giải thuật học loại bỏ ứng cử viên

Giải thuật loại bỏ ứng cử viên sử dụng phép toán tổ hợp hai giải thuật hướng đặc trưng hoá đến tổng quát hoá và tổng quát hoá đến đặc trưng hoá.

Giải thuật sử dụng các tập mẫu huấn luyện dương P và âm N tìm kiếm qua không gian học để tìm ra một mẫu mong muốn

Giải thuật được trang bị bằng hai danh sách G và S, với G chứa các ứng viên mẫu tổng quát hoá cực đại và S chứa các ứng viên mẫu đặc trưng hoá cực đại.

Giải thuật đặc trưng hoá G và tổng quát hoá S cho đến khi chúng hội tụ đến mẫu đích.

Begin

Ban đầu cho G chứa mẫu tổng quát hoá nhất trong không gian học

Và cho S chứa mẫu huấn luyện dương thứ nhất

For mỗi mẫu huấn luyện dương mới p

Begin

Huỷ bỏ tất cả các thành viên của G mà không hợp với p

For ■s ■S, nếu s không hợp với p, thay thế s với các thành phần tổng quát đặc trưng nhất của nó mà hợp với p

Huỷ bỏ từ S bất kỳ đích nào tổng quát hoá hơn một vài đích khác trong S

Huỷ bỏ từ S bất kỳ đích nào không đặc trưng hơn một vài đích khác trong G

End

For mỗi mẫu huấn luyện âm mới n

Begin

Huỷ bỏ tất cả các thành viên của S hợp với n

For mỗi g■G mà hợp với n, thay thế g với các thành phần đặc trưng tổng quát hoá nhất mà không hợp với n

Huỷ bỏ từ G bất kỳ đích nào đặc trưng hơn một vài đích khác trong G

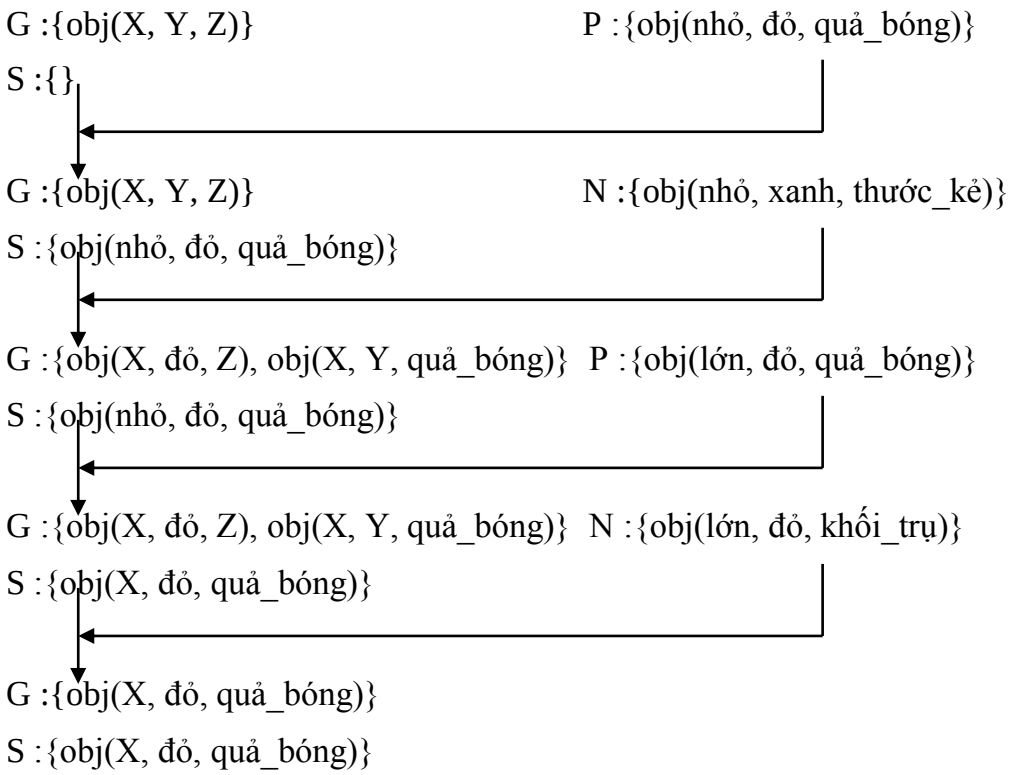
Huỷ bỏ từ G bất kỳ đích nào đặc trưng hơn một vài đích khác trong S

Nếu G = S và cả hai chỉ chứa một mẫu duy nhất, thì lúc này giải thuật đã tìm thấy mẫu đó là mẫu tương thích với tất cả dữ liệu và giải thuật dừng

Nếu G và S trở thành rỗng, thì không có mẫu bao trùm tất cả các mẫu huấn luyện dương và không có mẫu của các mẫu huấn luyện âm.

End

Vd 4.7 : Sử dụng giải thuật học loại bỏ ứng cử viên để học nhận dạng quả bóng màu đỏ sử dụng các tập mẫu huấn luyện P và N đã cho trong ví dụ 4.5. Thông qua quá trình học qua không gian học Hình 4.2, ta sẽ nhận được kết quả như Hình 4.5 dưới đây



Hình 4.5

V. BÀI TẬP

Bài 1 : Cho miền các đối tượng có các thuộc tính và các giá trị như sau :

da = {trắng, ngăm đen} ;

màu_tóc = {vàng, hạt dẻ, nâu} ;

dáng_người = {mảnh khảnh, béo, trung bình} ;

a. Sử dụng logic vị từ để biểu diễn các đối tượng này.

b. Sử dụng phép toán tổng quát hoá để định nghĩa không gian học của các đối tượng.

c. Xây dựng các tập mẫu huấn luyện dương P và âm N để học nhận dạng đối tượng là người có dáng_người mảnh khảnh sử dụng cả 3 loại giải thuật đã được thảo luận trên.

Bài 2 : Cho miền của các đối tượng có các thuộc tính và các giá trị như sau :

có = {lông, mỏ, cánh, động_cơ}

là = {chim, máy_bay}

tên = {đại_bàng, boeing}

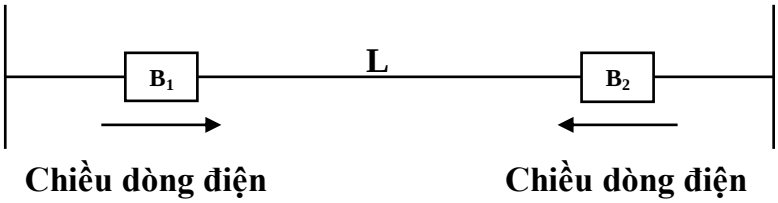
Hãy sử dụng logic vị từ để biểu diễn các đối tượng và xây dựng các tập mẫu huấn luyện để học nhận dạng đối tượng là đại_bàng sử dụng giải thuật học loại bỏ ứng viên.

CHƯƠNG V:

VÀI ỨNG DỤNG TRÍ TUỆ NHÂN TẠO

I. ỨNG DỤNG TRÍ TUỆ NHÂN TẠO ĐỂ PHÂN TÍCH BẢO VỆ HỆ THỐNG NĂNG LƯỢNG ĐIỆN

Cho một hệ thống đơn giản chỉ có một đường dẫn L được bảo vệ bởi hai máy cắt B_1 và B_2 như Hình 5.1 dưới đây :



Hình 5.1

Máy cắt B1 bảo vệ đường dẫn L hướng theo chiều dòng điện qua đường dẫn L từ trái sang phải và B2 bảo vệ đường dẫn L hướng theo chiều dòng điện chảy qua đường dẫn L từ phải sang trái.

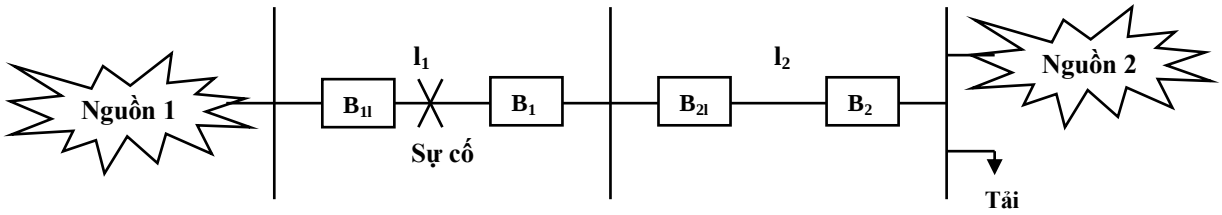
Khi có sự cố xảy ra trên đường dẫn L, một trong hai máy cắt hoặc cả hai phải hoạt động để ngắt dòng nếu chúng có nguồn.

Xây dựng mô hình hình học của một hệ thống năng lượng điện đơn giản như dưới đây :

- Hệ gồm :
    1. Các nguồn 1 & 2 là nguồn cung cấp năng lượng
    2. Các đường dẫn
    3. Tải
    4. Các máy cắt
- Các đường dẫn nối với nguồn và tải.
- Mỗi đường dẫn được bảo vệ bởi 2 máy cắt điện ở hai đầu cuối

Mục tiêu của việc phân tích bảo vệ hệ thống là một khi có sự cố xảy ra trên đường dẫn thì các máy cắt bảo vệ đường dẫn đó phải hoạt động ngắt dòng điện để tránh sự cố lan truyền đến các đường dẫn khác.

Trường hợp nếu các máy cắt bảo vệ đường dẫn không hoạt động thì ta phải cần đến các máy cắt dự phòng hoạt động đó là các máy cắt bảo vệ các đường dẫn khác trong cùng hệ thống đó.



Hình 5.2

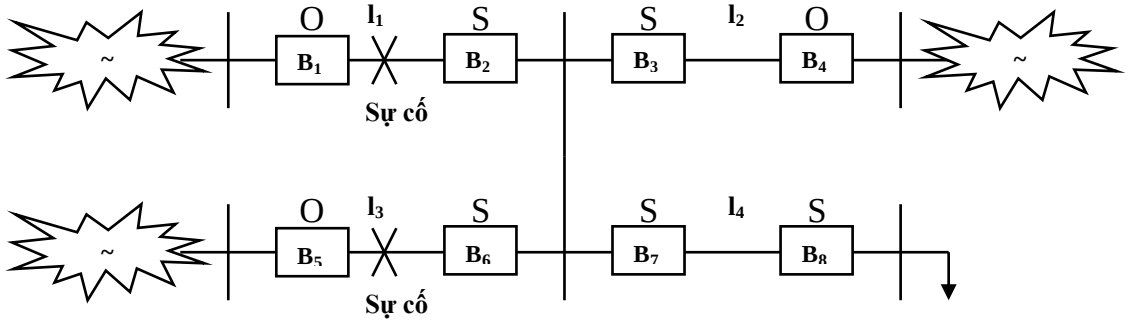
Từ Hình 5.2, ta có Nguồn 1, 2 là nguồn cung cấp. B_{1l}, B_{1r}, B_{2l}, B_{2r} là các máy cắt bảo vệ các đường dẫn l_1 và l_2 ; tải là nơi tải điện của hệ thống.

Tiến trình phân tích bảo vệ hệ thống năng lượng điện như sau :

1. Xác định vị trí có sự cố xảy ra trên hệ thống
2. Xác định các trạng thái hoạt động (O) và không hoạt động hay phục vụ (S) của các máy cắt trên hệ thống.
3. Chỉ rõ các thành phần của hệ thống
4. Xác định các máy cắt dự phòng cho máy cắt không hoạt động

Thông thường để thiết kế một hệ thống thông minh phân tích bảo vệ hệ thống năng lượng điện, yêu cầu đặt ra là hệ thống đó chỉ có một sự cố xảy ra trên đường dẫn l_x và chỉ có một máy cắt không hoạt động trên đường dẫn đó_khi đó sẽ có các máy cắt dự phòng hoạt động thay cho máy cắt đó.

Vd 5.1: Cho hệ thống năng lượng điện xây dựng như sơ đồ hình dưới đây:



Hình 5.3

Với: ~ là nguồn cung cấp
O là trạng thái hoạt động
S là trạng thái phục vụ (không hoạt động)
l_1, l_2, l_3, l_4 là tên các đường dẫn
B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8 là tên các máy cắt

Yêu cầu: Thiết kế một hệ thống trí tuệ nhân tạo sao cho nó có khả năng phân tích, bảo vệ hệ thống năng lượng điện (sơ đồ cho trên) với các mục tiêu sau:

1. Báo cáo khả năng sự cố xảy ra trên mỗi đường dẫn l_x
2. Báo cáo máy cắt hoạt động chính xác trên đường dẫn l_x
3. Báo cáo máy cắt không hoạt động trên đường dẫn l_x
4. Báo cáo các máy cắt dự phòng cho máy cắt không hoạt động trên đường dẫn l_x

Muốn thiết kế được như vậy đầu tiên chúng ta cần xây dựng hai thành phần cơ bản của một hệ thống trí tuệ nhân tạo:

- Thành phần cơ sở dữ liệu hay tri thức mô tả
- Thành phần các luật vận hành cơ sở dữ liệu còn gọi là tri thức thủ tục.

Hai thành phần này tạo thành một cơ sở tri thức trong hệ thống.

Để xây dựng **cơ sở dữ liệu** cho hệ thống năng lượng điện được mô tả ở hình 5.3, ta cần xây dựng các vị từ tổng quát, đó là:

- Vị từ bảo vệ đường dẫn l_x
- Vị từ liên thông giữa hai máy cắt qua thanh góp
- Vị từ máy cắt nối trực tiếp với nguồn
- Vị từ mô tả trạng thái hoạt động của máy cắt

Các vị từ này được thiết kế dưới dạng ngôn ngữ logic vị từ và chúng được sử dụng để biểu diễn các sự kiện hiện có của hệ thống như sau:

1. Vị từ bảo vệ đường dẫn l_x có 2 máy cắt B_y và B_z được xây dựng:

   protected_by(l_x,B_y,B_z)

   Áp dụng trong hệ thống, vị từ này dùng để biểu diễn các sự kiện như:

   protected_by(l_1,B_1,B_2)
   protected_by(l_2,B_3,B_4)
   protected_by(l_3,B_5,B_6)
   protected_by(l_4,B_7,B_8)

2. Vị từ liên thông giữa 2 máy cắt B_x và B_y qua thanh góp được định nghĩa

   connect(B_x,B_y)

   Vị từ này biểu diễn các sự kiện liên thông hiện có giữa các máy cắt như sau:

   connect(B_2,B_3)
   connect(B_2,B_6)
   connect(B_2,B_7)
   connect(B_3,B_6)
   connect(B_3,B_7)
   connect(B_6,B_7)

3. Vị từ mô tả máy cắt nối trực tiếp với nguồn

   generation(B)

   Vị từ này biểu diễn các sự kiện hiện có của các máy cắt nối trực tiếp với các nguồn năng lượng: generation(B_1)
   generation(B_4)
   generation(B_5)

4. Vị từ mô tả trạng thái hoạt động của các máy cắt

   operate(B)

   Trong hệ thống, các sự kiện hiện có của các máy cắt được biểu diễn áp dụng vị từ này:

   operate(B_1)
   operate(B_4)
   operate(B_5)

Để hình thành **các luật vận hành cơ sở dữ liệu**_thành phần quan trọng thứ hai của một hệ thống trí tuệ nhân tạo để phân tích bảo vệ hệ thống năng lượng điện cần tiến hành các bước xác định sau:

- Xác định các luật:
  - Luật xác định liên thông giữa 2 máy cắt
  - Luật xác định quan hệ giữa 2 máy cắt cùng bảo vệ một đường dẫn
  - Luật xác định máy cắt có nguồn trực tiếp và có nguồn gián tiếp
  - Luật xác định máy cắt không hoạt động thì có các máy cắt dự phòng khác hoạt động
  - Luật xác định máy cắt dự phòng không hoạt động
  - Luật xác định máy cắt mất nguồn

- Luật thông báo máy cắt hoạt động
- Luật thông báo máy cắt không hoạt động
- Luật thông báo trạng thái hoạt động của các máy cắt dự phòng

- Xây dựng các luật dưới dạng ngôn ngữ logic vị từ:
  - Luật xác định sự liên thông giữa 2 máy cắt:
    if connect(By,Bz) then connection(By,Bz)
    if connect(Bz,By) then connection(By,Bz)
  - Luật xác định quan hệ giữa 2 máy cắt cùng bảo vệ một đường dẫn, nếu biết tên máy cắt này -> sẽ biết tên máy cắt kia:
    if protected_by(lx,By,Bz) then other_breaker(By,Bz)
    if protected_by(lx,Bz,By) then other_breaker(By,Bz)
  - Luật xác định máy cắt có nguồn trực tiếp
    if generation(B) then has_gen(B)
  - Luật xác định máy cắt có nguồn gián tiếp nhờ sự liên thông giữa 2 máy cắt
    if connection(B,By) and other_breaker(By,Bz) and has_gen(Bz) then has_gen(B)
  - Luật xác định các máy cắt dự phòng cho máy cắt không hoạt động
    if not(generation(By)) and connection(By,Bt) and other_breaker(Bt,Bz) and has_gen(Bz) then back_up(By,Bz)
  - Luật xác định máy cắt dự phòng không hoạt động
    if back_up(By,Bz) and not(operate(Bz)) then back_up_did_not_work(By,Bz)
  - Luật xác định máy cắt mất nguồn
    if not(has_gen(By)) then no_source_comming(By)
    if has_gen(By) and operate(By) then no_source_comming(By)
    if back_up(By,Bz) and not(back_up_did_not_work(By,Bz)) then no_source_comming(By)
  - Luật thông báo máy cắt hoạt động chính xác như máy cắt dự phòng
    if back_up(B, By) and operate(By) and write("breaker") and write("operated correctly as a backup breaker") then printbackup(B)
  - Luật thông báo máy cắt hoạt động chính xác
    if has_gen(B) and operate(B) and write("breaker") and write(B) and write("operated correctly") then printout(B)
  - Luật thông báo máy cắt không hoạt động
    if has_gen(B) and not(operate(B)) and write("breaker") and write(B) and write("malfunctioned") and not(printbackup(B)) then printout(B)

Vd 5.2: Với hệ thống tri thức mô tả và tri thức thủ tục vừa xây dựng ở trên áp dụng cho hệ thống năng lượng điện cho ở hình 5.3. Hãy chứng minh rằng các máy cắt B4 và B5 là các máy cắt dự phòng cho máy cắt B2 khi có sự cố xảy ra trên đường dẫn l1.

Có thể sử dụng cả hai chiến lược tìm kiếm suy diễn tiến (hướng từ dữ liệu đến đích) và chiến lược tìm kiếm suy diễn lùi (hướng từ đích lùi về dữ liệu).

Cách 1: Sử dụng chiến lược suy diễn tiến

- Bắt đầu từ dữ liệu: máy cắt B2 không nối trực tiếp với nguồn, biểu diễn bằng vị từ: **■(generation(B2))** có trị đúng
- Máy cắt B2 và B3 liên thông qua thanh góp được biểu diễn bằng vị từ: **connection(B2,B3)** có trị đúng

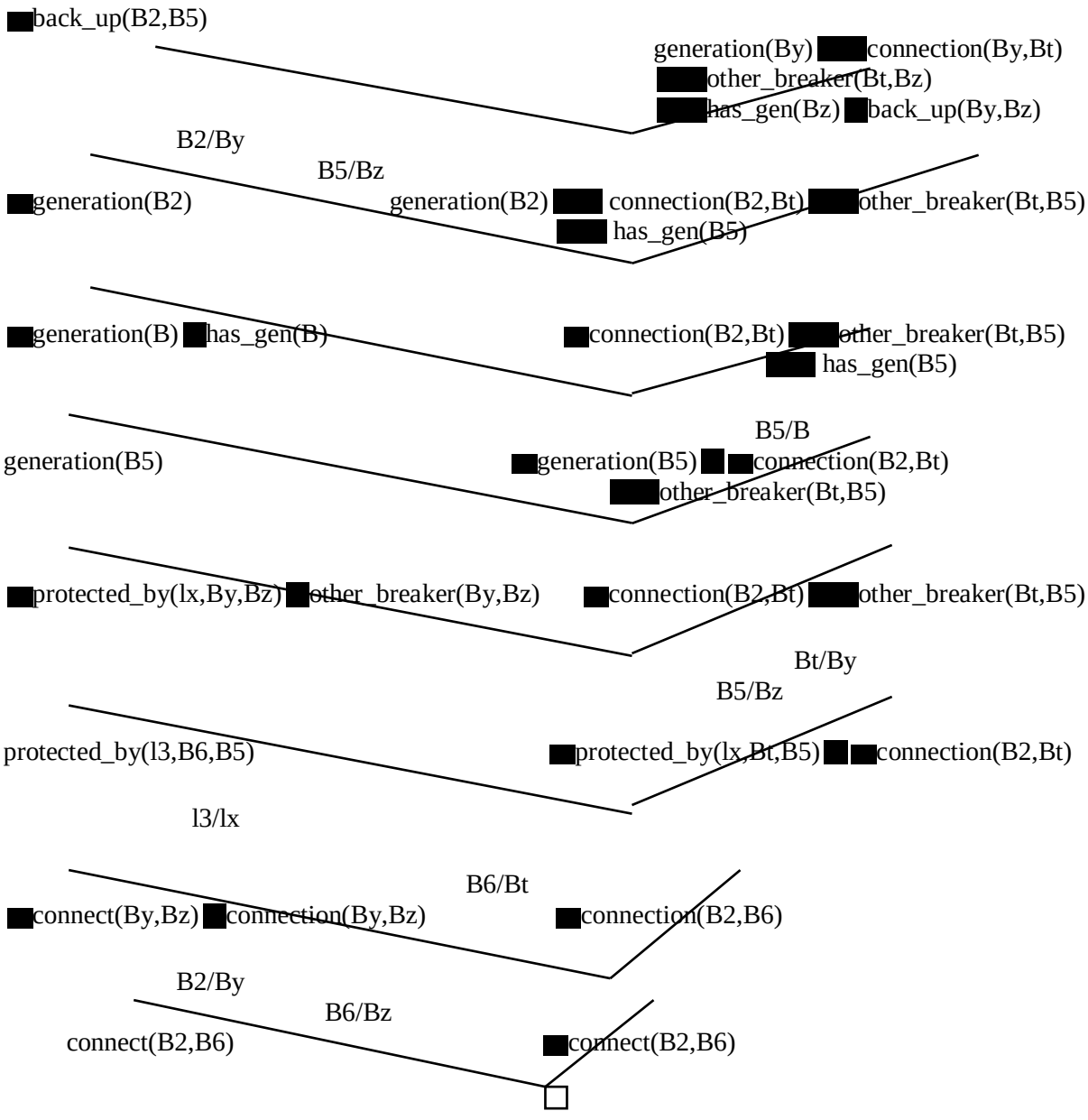
- Máy cắt B3 và B4 cùng bảo vệ đường dẫn l2, nếu biết B3 thì biết B4 và ngược lại được biểu diễn bằng vị từ: **other_breaker(B3,B4)** có trị đúng
- Máy cắt B4 có nguồn (nối trực tiếp với nguồn) được biểu diễn bằng vị từ: **has_gen(B4)** có trị đúng

Mà theo trên ta có luật xác định các máy cắt dự phòng cho máy cắt không hoạt động: if **(generation(B2))** and **connection(B2,B3)** and **other_breaker(B3,B4)** and **has_gen(B4)** then **back_up(B2,B4)**

Tất cả các vị từ ở vế bên trái của luật đều có trị đúng, vì thế vế bên phải của luật là **back_up(B2,B4)** phải có trị đúng hay nói cách khác máy cắt B4 là máy cắt dự phòng cho máy cắt B2 nếu có sự cố xảy ra trên đường dẫn l1. Tương tự, ta có thể chứng minh máy cắt B5 là máy cắt dự phòng cho máy cắt B2.

Cách 2: Sử dụng chiến lược suy diễn lùi (phương pháp hợp giải):

- Giả sử máy cắt B5 không phải là máy cắt dự phòng cho máy cắt B2, ta có vị từ: **(back_up(B2,B5))**



Hình 5.4

Cơ sở tri thức được sử dụng để chứng minh máy cắt B5 là máy cắt dự phòng cho máy cắt B2 được liệt kê như sau:

1. ■(generation(B2)
2. generation(B5)
3. connect(B2,B6)
4. protected_by(l3,B6,B5)
5. operate(B5)
6. connect(By,Bz) → connection(By,Bz)
7. protected_by(lx,By,Bz) → other_breaker(By,Bz)
8. generation(B) → has_gen(B)
9. ■(generation(By))■connection(By,Bt)■other_breaker(Bt,Bz) ■has_gen(Bz) → back_up(By,Bz)

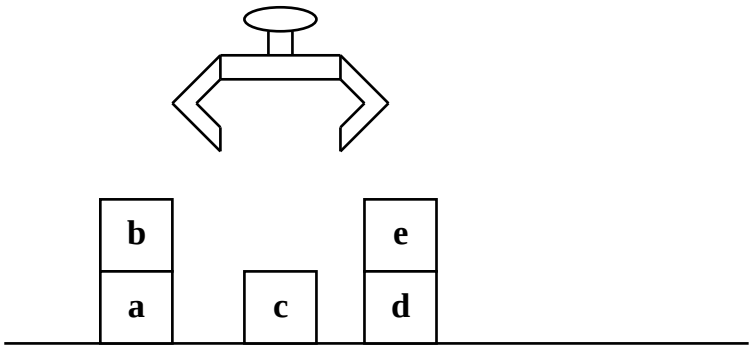
Hệ các tiên đề này được chuyển sang dạng mệnh đề:

1. ■(generation(B2)
2. generation(B5)
3. connect(B2,B6)
4. protected_by(l3,B6,B5)
5. operate(B5)
6. ■connect(By,Bz)■connection(By,Bz)
7. ■protected_by(lx,By,Bz)■other_breaker(By,Bz)
8. ■generation(B)■has_gen(B)
9. generation(By)■■connection(By,Bt)■■other_breaker(Bt,Bz) ■■ has_gen(Bz)■back_up(By,Bz)
10. ■back_up(B2,B5)

Quá trình chứng minh máy cắt B5 là máy cắt dự phòng cho máy cắt B2 được mô tả rõ qua sơ đồ Hình 5.4(trên).

II. PHƯƠNG ÁN TRONG CÁC HỆ THỐNG TRÍ TUỆ NHÂN TẠO

Phương án là một ứng dụng trí tuệ nhân tạo, có nhiều ứng dụng trong công nghiệp sản xuất. Chẳng hạn như phương án điều khiển quá trình tự động của cánh tay robot. Quan sát thế giới khối như hình vẽ dưới đây :



Trạng thái đầu

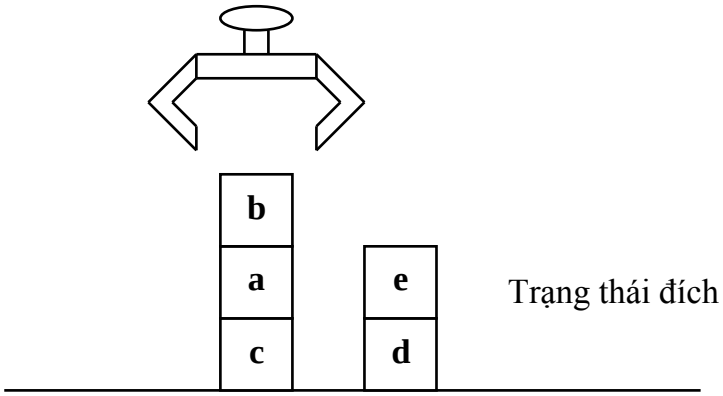
Hình 5.5

Trên một mặt bàn, đặt các khối a, b, c, d và e có cùng kích thước.

Yêu cầu là cánh tay robot có khả năng nhặt các khối từ vị trí này đến đặt tại vị trí khác như Hình 5.6 dưới đây:

Để di chuyển các khối, cánh tay robot phải thực hiện các thao tác như: xác định vị trí của một khối, nhặt một khối lên từ mặt bàn, đặt một khối xuống mặt bàn, lấy một khối từ đỉnh của một khối khác, đặt một khối lên đỉnh của một khối khác. Xây dựng các vị từ biểu diễn các thao tác trên:

1. **goto(X,Y,Z)**: đi đến vị trí được mô tả bằng toạ độ X, Y, Z.



Hình 5.6

2. **pickup(X)**: nhặt khối X lên từ mặt bàn. Điều kiện khối X phải nằm trên mặt bàn, không có vật gì trên khối X và lòng bàn tay robot không cầm giữ bất cứ vật gì.
3. **putdown(X)**: đặt khối X xuống mặt bàn. Điều kiện là lòng bàn tay robot đang cầm giữ khối X.
4. **takeoff(X,Y)**: lấy khối X từ đỉnh của khối Y. Với điều kiện là khối X phải nằm trên khối Y, không có vật gì trên khối X và lòng bàn tay robot không cầm giữ bất cứ vật gì.
5. **puton(X,Y)**: đặt khối X lên đỉnh của khối Y. Các điều kiện là lòng bàn tay robot đang cầm giữ khối X và không có khối nào nằm trên khối Y.

Có thể biểu diễn trạng thái của thế giới khối sử dụng tập các vị từ và vị từ quan hệ như sau:

location(A,X,Y,Z): khối A ở tại vị trí có toạ độ X, Y, Z

on(X,Y): khối X nằm trên khối Y

clear(X): không có vật gì trên khối X

hold(X): lòng bàn tay robot đang cầm giữ khối X

hold( ) : lòng bàn tay robot không cầm giữ bất cứ vật gì

ontable(X): khối X nằm trên mặt bàn

Vd 5.3: Sử dụng các vị từ đã xây dựng ở trên để biểu diễn trạng thái ban đầu của thế giới khối Hình 5.5.

Trạng thái ban đầu của thế giới khối Hình 5.5 được biểu diễn bằng các vị từ:

ontable(a)	on(b,a)	hold( )	clear(b)
ontable(c)	on(e,d)		clear(c)
ontable(d)			clear(e)

Dựa vào hệ thống vị từ biểu diễn trạng thái ban đầu của thế giới khối trên ta thiết kế bộ các luật để khẳng định 3 trường hợp sau:

- Khối không có bất kỳ khối khác nằm trên nó
- Khối đang có trên mặt bàn
- Bàn tay robot chưa cầm giữ vật gì

Luật 1: Nếu khối X sạch thì không tồn tại bất kỳ khối Y nào sao cho Y nằm trên đỉnh của khối X

(∀X) (clear(X) → ¬(∃Y)(on(Y,X)))

Luật 2: Nếu một khối nằm trên bàn thì nó không nằm trên đỉnh của một khối bất kỳ nào khác

(∀Y) (∀X) (¬on(Y,X) ∨ ontable(Y))

Luật 3: Lòng bàn tay robot không cầm giữ bất cứ vật gì nếu và chỉ nếu không có bất kỳ khối X nào sao cho bàn tay robot đang cầm giữ

(∀X) hold( ) ↔ ¬(hold(X))

Từ 3 luật xác định trạng thái hiện có của thế giới khối trên, ta có thể thiết kế bộ các luật 4, 5, 6, 7 là các luật vận hành để thay đổi trạng thái trong không gian của bài toán dưới dạng A → (B → C). Dạng suy diễn kiểu này có nghĩa là A cho phép suy ra trạng thái mới B chỉ khi nào điều kiện C là đúng.

Luật 4: Nếu nhặt một khối X bất kỳ lên từ mặt bàn thì lòng bàn tay robot phải cầm giữ khối đó chỉ khi nào trước đó lòng bàn tay robot là rỗng và không có bất kỳ khối nào khác trên đỉnh của nó.

(∀X) (pickup(X) → (hold(X) → (hold( ) ∧clear(X))))

Luật 5: Nếu đặt một khối X bất kỳ xuống mặt bàn thì bàn tay robot phải rỗng và khối đó phải nằm trên mặt bàn và không có bất kỳ khối nào khác nằm trên đỉnh của nó chỉ khi nào trước đó bàn tay robot đã cầm giữ khối đó.

(∀X) (putdown(X) → (hold( ) ∧ontable(X) ∧clear(X) → ¬hold(X)))

Luật 6: Nếu đặt khối X lên đỉnh khối Y thì khối X phải nằm trên đỉnh khối Y và bàn tay robot phải rỗng và không có bất kỳ khối nào khác nằm trên đỉnh của nó chỉ khi nào trước đó không có khối nào nằm trên đỉnh của Y và bàn tay robot đang cầm giữ khối X

(∀Y) (∀X) (puton(X,Y) →((on(X,Y) ∧hold( ) ∧clear(X))
→(clear(Y) ∧hold(X))))

Luật 7: Nếu lấy khối X từ đỉnh của khối Y thì không có khối nào nằm trên đỉnh của khối Y và bàn tay robot đang cầm giữ khối X chỉ khi nào trước đó khối X nằm trên khối Y và không có bất kỳ khối nào nằm trên đỉnh của khối X và bàn tay robot không cầm giữ bất kỳ vật gì.

(∀Y) (∀X) (takeoff(X,Y) →((clear(Y) ∧hold(X))
→(on(X,Y) ∧ (clear(X) ∧hold( )))

Tiếp theo ta thiết kế bộ các luật xác định một số trạng thái không bị thay đổi còn gọi là luật khung (hay quan hệ khung) khi một trong bốn luật vận hành thay đổi trạng thái được đưa ra ứng dụng. Chẳng hạn ta xét các luật xác định vị từ ontable, on, clear không bị thay đổi trạng thái khi hai luật puton và takeoff được đưa ra ứng dụng.

Luật 8: Nếu lấy một khối X bất kỳ từ đỉnh của khối Y thì khối Z vẫn nằm trên mặt bàn nếu trước đó Z đã có ở trên mặt bàn.

(∀X) (∀Y) (∀Z) (takeoff(X,Y) ∧ (ontable(Z) → ontable(Z)))

Luật 9: Nếu đặt khối X bất kỳ lên đỉnh khối Y thì khối Z vẫn nằm trên mặt bàn nếu trước đó Z đã có ở trên bàn

(∀X) (∀Y) (∀Z) (puton(X,Y) ∧ (ontable(Z) → ontable(Z)))

Luật 10: Nếu lấy một khối X bất kỳ từ đỉnh của khối Y thì khối Z vẫn nằm trên đỉnh khối F nếu trước đó Z đã nằm trên F.

(∀X) (∀Y) (∀Z) (∀F) (takeoff(X,Y) ∧ (on(Z,F) → on(Z,F)))

Luật 11: Nếu đặt khối X bất kỳ lên đỉnh khối Y thì khối Z vẫn nằm trên đỉnh khối F nếu trước đó Z đã nằm trên F.

(∀X) (∀Y) (∀Z) (∀F) (puton(X,Y) ∧ (on(Z,F) → on(Z,F)))

Luật 12: Nếu lấy một khối X bất kỳ từ đỉnh của khối Y thì sẽ không tồn tại bất cứ vật gì trên khối Z nếu trước đó khối Z sạch.

(∀X) (∀Y) (∀Z) (takeoff(X,Y) ∧ (clear(Z) → clear(Z)))

Luật 13: Nếu đặt khối X bất kỳ lên đỉnh khối Y thì sẽ không tồn tại bất cứ vật gì trên khối Z nếu trước đó khối Z sạch.

(∀X) (∀Y) (∀Z) (puton(X,Y) ∧ (clear(Z) → clear(Z)))

Tương tự chúng ta có thể thiết kế một số luật khung khác để xác định một số vị từ không bị thay đổi trạng thái nếu các luật vận hành được áp dụng.

Tập các luật gồm các luật vận hành thay đổi trạng thái và luật khung định nghĩa một không gian gọi là không gian trạng thái tìm kiếm của bài toán thế giới khối.

Phương án có thể được xem như một không gian trạng thái tìm kiếm. Các đặc thù của bài toán phương án là:

- Kỹ thuật tìm kiếm trên đồ thị
- Biểu diễn tri thức nhờ logic vị từ
- Các phép toán suy diễn thay đổi trạng thái

Bảng tam giác:

Bảng tam giác được tìm ra vào năm 1972 (bởi Fikes và Nilsson). Đó thực chất là một cấu trúc dữ liệu để tổ chức dãy các tác động của một phương án. Mục đích là để phân loại phương án và mô tả luật tương tác của phương án.

Bảng tam giác có tổng số ô là

$$(x + 1)^2/2 + x/2 = (x(x + 3)/2 + 1)$$

với x là số luật được đưa ra ứng dụng O_1, O_2, …, O_x

- Mỗi ô trong bảng có thể rỗng hoặc chứa các trạng thái.

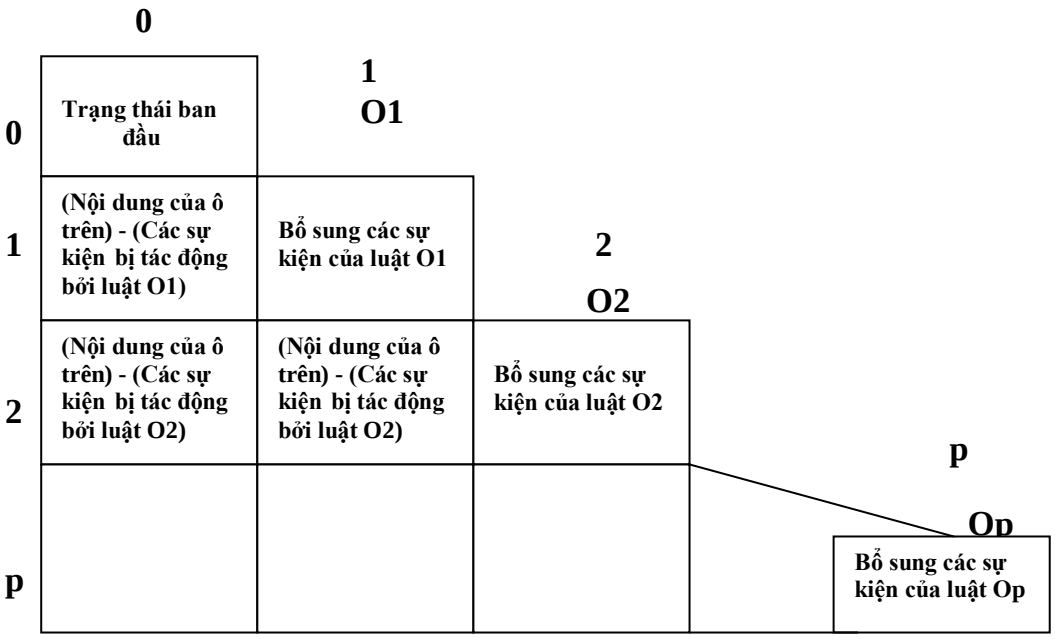
- Số luật x tương ứng với số cột của bảng.

- Giả sử nếu có c cột thì luật thứ O_1 tương ứng với c=1, cột thứ c tương ứng với luật thứ O_c .

- Ô có chỉ số cột c = 0, chỉ số hàng r = 0 của bảng chứa trạng thái ban đầu của phương án.

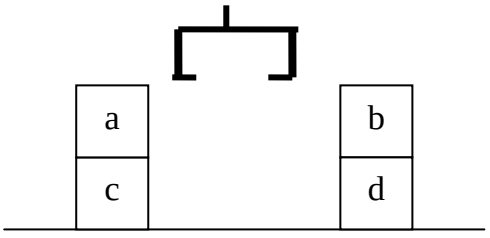
Sơ đồ cấu trúc tổng quát của bảng được tổ chức theo trình tự như hình 5.7 dưới đây:

☞ Chú ý: Khi xây dựng phải bắt đầu từ phương án, từ đó mới xây dựng bảng tam giác tương ứng.

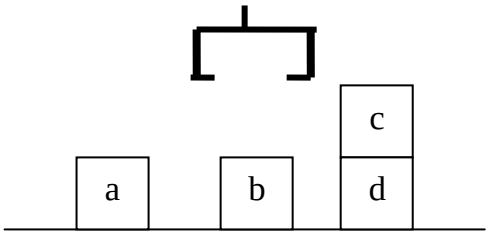


Hình 5.7

Vd 5.4: Hãy xây dựng phương án và bảng tam giác tương ứng với phương án cho thế giới khối như dưới đây:

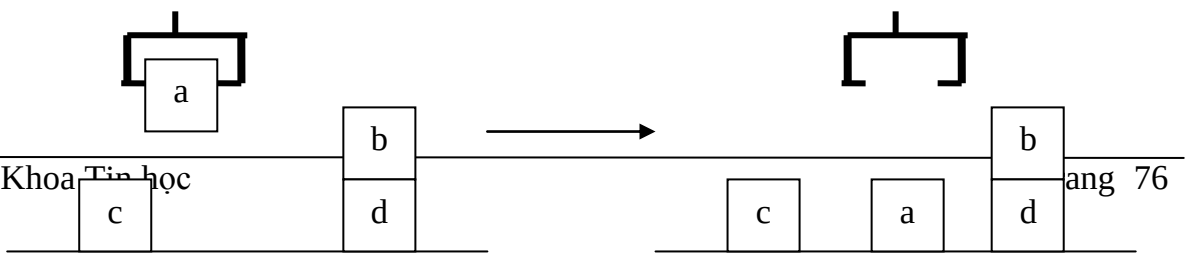


Trạng thái ban đầu

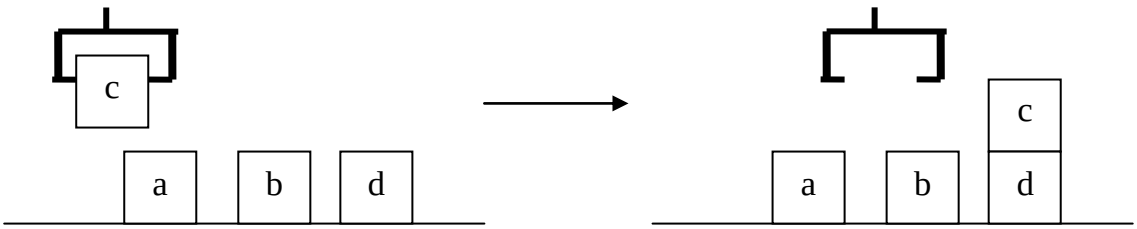


Trạng thái đích

+ Phương án được mô tả như sau:



takeoff(a,c) putdown(a)



pickup(c) puton(c,d)

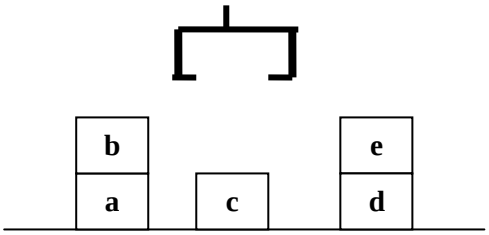
+ Bảng tam giác tương ứng cho phương án thế giới khối trên :

on(a,c) **on(b,d)** **ontable(c)** **ontable(d)** **clear(a)** **clear(b)** **hold( )**	1 **takeoff(a,c)**					
on(b,d) **ontable(c)** **ontable(d)** **clear(b)**	**hold(a)** **clear(c)**	2 **putdown(a)**				
on(b,d) **ontable(c)** **ontable(d)** **clear(b)**	**clear(c)**	**hold( )** **ontable(a)** **clear(a)**	3 **takeoff(b,d)**			
ontable(c) **ontable(d)**	**clear(c)**	**ontable(a)** **clear(a)**	**hold(b)** **clear(d)**	4 **putdown(b)**		
ontable(c) **ontable(d)**	**clear(c)**	**ontable(a)** **clear(a)**	**clear(d)**	**hold( )** **clear(b)** **ontable(b)**	5 **pickup(c)**	
ontable(d)		**ontable(a)** **clear(a)**	**clear(d)**	**clear(b)** **ontable(b)**	**hold(c)**	6 **puton(c,d)**
ontable(d)		**ontable(a)** **clear(a)**		**clear(b)** **ontable(b)**	**hold( )**	**hold( )** **on(c,d)** **clear(c)**

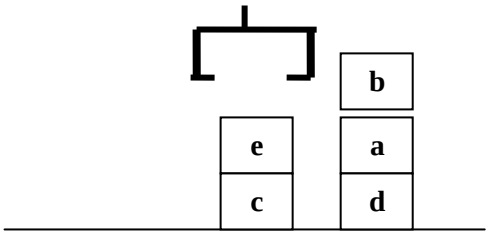
Qua bảng ta nhận thấy sau khi thực hiện luật 6, thì đạt đến trạng thái đích với giá trị trong các ô của bảng thể hiện đầy đủ trạng thái này: ontable(a), clear(a); ontable(b), clear(b); on(c,d), clear(c), ontable(d); hold( ).

III. BÀI TẬP

Bài 1 : Cho thế giới khối (Hình dưới)



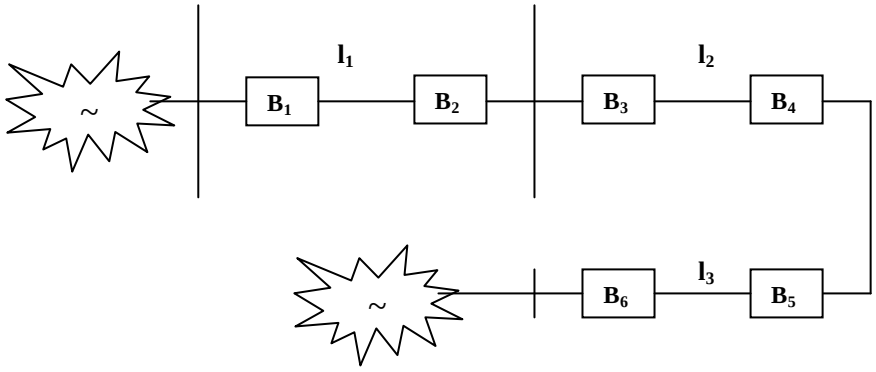
Trạng thái đầu



Trạng thái đích

Hãy xây dựng phương án để robot di chuyển các khối từ trạng thái đầu đến trạng thái đích và bảng tam giác tương ứng với phương án này.

Bài 2 : Cho hệ thống năng lượng điện như hình vẽ :



Các trạng thái của các máy cắt được liệt kê ở bảng : (O : hoạt động ; S : không hoạt động (phục vụ))

Máy cắt	Trường hợp 1	Trường hợp 2	Trường hợp 3
1	S	S	O
2	S	S	S
3	S	O	O
4	S	S	S
5	O	S	S
6	O	O	S

a. Tìm các máy cắt có nguồn ứng với mỗi trường hợp

b. Với từng trường hợp hãy xác định các máy cắt hoạt động và máy cắt dự phòng hoạt động cho máy cắt không hoạt động khi có sự cố trên mỗi đường dẫn.

CHƯƠNG VI:

XỬ LÝ TRI THỨC KHÔNG CHẮC CHẮN TRONG CÁC HỆ THỐNG TRÍ TUỆ NHÂN TẠO

I. LÝ GIẢI VỚI SỰ KHÔNG CHẮC CHẮN

Các sự kiện của bài toán mà chúng ta đã gặp trước đây được giả sử là tuyệt đối đúng hoặc tuyệt đối sai. Dạng tri thức này được gọi là tri thức chắc chắn . Cách xử lý nó tuân theo các phép toán của logic rõ (đúng và sai tương ứng với hai chữ số 1 và 0). Nó hoàn toàn không phù hợp với tri thức không chắc chắn mà chúng ta sắp thảo luận dưới đây.

Ngược lại việc giả sử rằng các sự kiện của bài toán chưa hẳn tuyệt đối đúng hoặc tuyệt đối sai thuộc về nhóm tri thức không chắc chắn. Việc lý giải tri thức không chắc chắn phải kèm theo số đo chắc chắn trong biểu diễn cũng như trong luật suy diễn. Số đo chắc chắn còn gọi là độ tin cậy của biểu diễn và độ tin cậy của luật suy diễn.

Vd 6.1: Xét phát biểu: "Mưa dường như không to lắm".

Giả sử phát biểu này có độ tin cậy khoảng 50% hay 0.50 thì độ chắc chắn của phát biểu chỉ đúng khoảng 50%.

Nếu ta xem xét luật dạng: p → q

Trong tri thức chắc chắn, độ tin cậy của p là con(p) = 1
độ tin cậy của suy diễn là con(p → q) = 1
Từ đó suy ra độ tin cậy của q là con(q) = 1.

Trong tri thức không chắc chắn, độ tin cậy của p là con(p) <1
độ tin cậy của suy diễn là con(p → q) <1
Do đó độ tin cậy của q phải là con(q) <1

Có hai phương pháp xử lý tri thức không chắc chắn trong các hệ thống trí tuệ nhân tạo đó là phương pháp xử lý tri thức không chắc chắn sử dụng xác suất thống kê và phương pháp xử lý không chắc chắn sử dụng logic mờ.

II. XỬ LÝ TRI THỨC KHÔNG CHẮC CHẮN SỬ DỤNG XÁC SUẤT THỐNG KÊ

1. Xác suất điều kiện _ các nguyên lý căn bản

Giả sử có một cái hộp chứa nhiều quả bóng, mỗi quả bóng có đánh dấu trên nó để giúp ta đếm số lần thực nghiệm.

- Một số quả bóng được đánh dấu chữ cái "A"
- Một số quả bóng được đánh dấu chữ cái "B"

- Một số quả bóng được đánh dấu hai chữ cái "A" và "B"
- Một số quả bóng không được đánh dấu bất cứ ký hiệu nào

Quá trình tiến hành thí nghiệm:

- Lấy một quả bóng từ hộp và sau đó lại bỏ nó trở lại vào hộp
- Đếm số lần lặp lại của các quả bóng có đánh dấu chữ cái A, chữ cái B và hai chữ cái A B.

Gọi : + t1 : số lần lấy ra của các quả bóng có đánh dấu chữ cái A

+ t2 : số lần lấy ra của các quả bóng có đánh dấu chữ cái B

+ t3 : số lần lấy ra của các quả bóng có đánh dấu cả hai chữ cái AB

+ t4 : số lần lấy ra của các quả bóng không có đánh dấu

+ t : tổng số lần của các quả bóng được lấy ra

Khi đó :

Xác suất lấy ra của các quả bóng	Ký hiệu đánh dấu
P(A) = t1/t	A
P(B) = t2/t	B
P(A và B) = t3/t	A và B

Xác suất điều kiện của A đối với B được định nghĩa là: P(A\B) = t3/t2

Dựa vào các công thức ở bảng trên → P(A\B) = P(A và B)/P(B) (1)

Vậy có thể phát biểu : xác suất điều kiện của A đối với B bằng xác suất của A và B chia cho xác suất của B.

(1) → P(A và B) = P(A\B) * P(B) (1')

Tương tự, công thức cơ bản xác suất điều kiện của B đối với A là:

→ P(B\A) = P(A và B)/P(A) (2)

(2) → P(A và B) = P(B\A) * P(A) (2')

(1') và (2') → P(A) * P(B\A) = P(B) * P(A\B) (luật Bayes)

Bên cạnh đó ta còn có các công thức:

P(A hoặc B) = P(A) + P(B) – P(A và B) (luật and-or (và-hoặc))

P(A) = P(A\B) * P(B) + P(A\not B) * P(not B) (luật gán phức hợp)

2. Lý giải xác suất chính xác dưới điều kiện không chắc chắn

Ở đây chúng ta thấy có sự mâu thuẫn : Làm cách nào nó có thể chính xác nếu nó là không chắc chắn ?

Để thực hiện được điều này, trong một tình huống không chắc chắn, các mảnh thông tin phải có các số đo chắc chắn đó chính là các giá trị xác suất của chúng. Các

giá trị xác suất này là nguồn cung cấp cho các suy diễn để xác định các giá trị xác suất chính xác của các kết luận.

Xem xét luật đơn giản có dạng: if (A) then (B)

Trường hợp 1: Nếu ta xét luật trong một tình huống chắc chắn:

- Khi đó số đo chắc chắn của tiền điều kiện A là 1, ứng với giá trị xác suất là:

$$P(A) = 1 \quad (3)$$

- Số đo chắc chắn của suy diễn A → B là 1, ứng với giá trị xác suất của nó là:

$$P(A \rightarrow B) = P(B\backslash A) = 1 \quad (4)$$

- Từ (3) và (4) : số đo chắc chắn của kết luận B là 1, ứng với giá trị xác suất là:

$$P(B) = 1;$$

Trường hợp 2: Nếu ta xét luật trong một tình huống không chắc chắn với :

- Xác suất của A là P(A) = 0.9
- Xác suất của suy diễn là P(B\A) = 0.95
- Làm thế nào để tính số đo chắc chắn cho kết luận B? Số đo chắc chắn hay xác suất của kết luận B được xác định bằng công thức luật gán phức hợp ở trên :

$$P(B) = P(B\backslash A) * P(A) + P(B\backslash not\ A) * P(not\ A) \quad (5)$$

Mà P(A) = 0.9, nên P(not A) = 0.1, thay vào luật (5) ở trên ta có:

$$P(B) = 0.95 * 0.9 + P(B\backslash not\ A) * 0.1$$

$$P(B) = 0.855 + P(B\backslash not\ A) * 0.1$$

Giả sử $P(B\backslash not\ A) = 0$

Vậy xác suất chính xác của B là : P(B) = 0.855

3. Lý giải xấp xỉ

Xem xét luật dạng đơn giản **if (A) then (B)**

Trong tình huống không chắc chắn, ta có thể cho

- Số đo chắc chắn của bằng chứng tương đương với giá trị xác suất của nó

Số đo chắc chắn của bằng chứng A = ct(A) ■ P(A)

- Số đo chắc chắn của suy diễn tương đương với giá trị xác suất của suy diễn

Số đo chắc chắn của suy diễn = ct(i) ■ P(B\A)

Cách lý giải xấp xỉ đó là số đo chắc chắn của kết luận B được tính bằng tích của số đo chắc chắn của bằng chứng A và số đo chắc chắn của suy diễn :

ct(kết luận) = ct(bằng chứng) * ct(suy diễn)

ct(B) = ct(A) * ct(i)

Tương tự, xét luật dạng: **if (A1 và A2) then (B)**

Số đo chắc chắn cho kết luận B được tính bằng công thức đã nêu trên :

ct(kết luận) = ct(bằng chứng) * ct(suy diễn)

với số đo chắc chắn bằng chứng A được cho xấp xỉ như sau :

$$ct(A) = ct(A1 \text{ and } A2) = \min(ct(A1), ct(A2))$$

Tương tự, nếu luật có dạng: **if(A1 hoặc A2) then (B)** thì số đo chắc chắn cho kết luận B cũng được tính bằng công thức:

$$ct(\text{kết luận}) = ct(\text{bằng chứng}) * ct(\text{suy diễn})$$

với số đo chắc chắn bằng chứng A được xấp xỉ bằng :

$$ct(A) = ct(A1 \text{ or } A2) = \max(ct(A1), ct(A2))$$

Giả sử luật có dạng như sau:

Luật 1: **if (A1) then (B)** ct(kết luận) = ct1

Luật 2: **if (A2) then (B)** ct(kết luận) = ct2

Nếu muốn tính tổng số đo chắc chắn của kết luận B trong hệ thống ta phải thực hiện như sau:

Tổng số đo chắc chắn của B = (số đo chắc chắn của B từ luật 1) + (số đo chắc chắn của B từ luật 2) – (số đo chắc chắn của B từ luật 1) * (số đo chắc chắn của B từ luật 2).

$$ctotal = ct1 + ct2 - ct1 * ct2$$

Nếu luật có dạng : if (A1 và (not A2)) then (B) thì số đo chắc chắn của A2 được tính bằng một trong 2 cách sau:

Tính số đo chắc chắn cho (not A2) bằng cách đổi dấu:

$$ct(\text{not } A) = -ct(A)$$

Tính số đo chắc chắn cho (not A2) bằng cách sử dụng công thức xác suất:

$$P(\text{not } A) = 1 - P(A)$$

III. XỬ LÝ TRI THỨC KHÔNG CHẮC CHẮN SỬ DỤNG LOGIC MỜ (FUZZY LOGIC)

Logic mờ là một trong những phương pháp xử lý tri thức không chắc chắn trong các hệ thống AI.

1. Khái niệm tập mờ và các phép toán trên tập mờ

a. Tập rõ (crisp set) : Cho A là một tập hợp trong không gian U, x là các phần tử của U. A được gọi là tập rõ trong U nếu A được định nghĩa bằng hàm đặc tính $X_A(x)$ của nó sao cho:

$$X_A(x) = \begin{cases} 1 & \text{nếu } x \in A \\ 0 & \text{nếu } x \notin A \end{cases}$$

b. Tập mờ (fuzzy set) : Cho F là một tập hợp trong không gian U, x là các phần tử của U. F được gọi là tập mờ trong U nếu F được định nghĩa bằng hàm liên thuộc $\mu_F(x)$ của nó sao cho $\mu_F(x)$ có thể lấy giá trị bất kỳ nào trong khoảng [0,1].

- Nếu U là không gian liên tục, tập mờ F trong U có thể được biểu diễn bằng: $F = \int_U \mu_F(x)/x$

Với ∫ là toán tử hợp cho hàm liên tục $\mu_F(x)$; ký hiệu / là toán tử kết hợp giữa giá trị liên thuộc với giá trị rõ của nó.

- Nếu U là không gian rời rạc, tập mờ F trong U có thể được biểu diễn bằng: $F = \sum \mu_F(x)/x$

Với $\sum$ là toán tử hợp cho hàm rời rạc $\mu_F(x)$

- Tập mờ F còn có thể được biểu diễn theo cách khác trong không gian rời rạc U như sau:

$$F = \mu_F(x_1)/x_1 + \mu_F(x_2)/x_2 + \mu_F(x_3)/x_3 + \ldots + \mu_F(x_i)/x_i + \ldots + \mu_F(x_N)/x_N$$

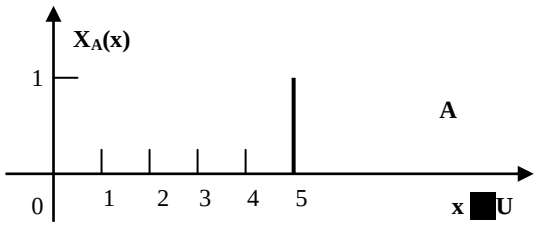
Với ký hiệu + là toán tử hợp; ký hiệu / là toán tử kết nối giữa giá trị liên thuộc ứng với giá trị rõ của nó; $\mu_F(x_i)$ là giá trị liên thuộc hay là số đo chắc chắn của phần tử x_i trong tập F.

Vd 6.2: Cho U là không gian các số thực và A là tập các số thực lớn hoặc bằng 5.

Nếu A là tập rõ trong U thì A được định nghĩa bằng hàm đặc tính của nó như sau:

$$X_A(x) = \begin{cases} 1 & x \geq 5 \\ 0 & x < 5 \end{cases}$$

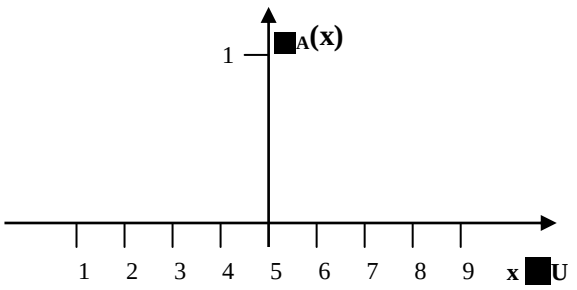
và A được biểu diễn ở Hình 6.1 bên



Hình 6.1

Nếu A là tập mờ trong U thì A được định nghĩa bằng hàm liên thuộc của nó:

$$\mu_A(x) = \frac{1}{(1 + 10(x - 5)^2)}$$



Hình 6.2

c. Các phép toán trên tập mờ:

- Phép toán bằng nhau:

Giả sử A và B là hai tập mờ trong không gian U, A và B được gọi là bằng nhau nếu và chỉ nếu : $\mu_A(x) = \mu_B(x) \quad \forall x \in U$

- Phép toán hợp của hai tập mờ:

Hai tập mờ A và B với các hàm liên thuộc tương ứng $\mu_A(x)$ và $\mu_B(x)$ trong không gian U hợp nhau sinh ra một tập mờ mà hàm liên thuộc của nó $\mu_A(x) \cup \mu_B(x)$ được định nghĩa bởi:

$$\mu_A(x) \cup \mu_B(x) = \max\{\mu_A(x), \mu_B(x)\} \quad \forall x \in U$$

- Phép toán giao của hai tập mờ:

Hai tập mờ A và B với các hàm liên thuộc tương ứng $\mu_A(x)$ và $\mu_B(x)$ trong không gian U giao nhau sinh ra một tập mờ mà hàm liên thuộc của nó $\mu_A(x) \cap \mu_B(x)$ được định nghĩa bởi:

$$\mu_A(x) \cap \mu_B(x) = \min\{\mu_A(x), \mu_B(x)\} \quad \forall x \in U$$

- Phép toán bù của tập mờ:

Nếu A là tập mờ trong không gian U với hàm liên thuộc $\mu_A(x)$ thì phần bù của tập mờ A là một tập mờ A' trong không gian U mà hàm liên thuộc của nó được định nghĩa :

$$\mu_{A'}(x) = 1 - \mu_A(x) \qquad \forall x \in U$$

- Tích cartesian (cartesian product):

Tích cartesian của các tập mờ $A_1, A_2, \ldots, A_n$ trong các không gian $U_1, U_2, \ldots, U_n$ là một tập mờ trong không gian tích $U_1 \times U_2 \times U_3 \times \ldots \times U_n$ với hàm liên thuộc của nó được định nghĩa:

$$\mu_{A1 \times A2 \times \ldots \times An}(x) = \min\{\mu_{A1}(x_1), \mu_{A2}(x_2), \ldots, \mu_{An}(x_n)\} \quad \forall x_1, x_2, \ldots, x_n \in U$$

- Tích đại số (algebraic product):

Tích đại số của hai tập mờ A và B với các hàm liên thuộc $\mu_A(x)$ và $\mu_B(x)$ là một tập mờ mà hàm liên thuộc của nó $\mu_{A \cdot B}(x)$ được định nghĩa:

$$\mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x) \qquad \forall x \in U$$

2. Quan hệ mờ và các phép toán trên các quan hệ mờ

a. Không gian tích: cho $x \in X$ và $y \in Y$, không gian tích của X và Y được định nghĩa:

$$X \times Y = \{(x, y) / x \in X, y \in Y\}$$

b. Quan hệ rõ: cho R là tập con của không gian tích $X \times Y$, R được gọi là quan hệ rõ nếu R được định nghĩa bằng hàm đặc tính của nó sao cho:

$$X_R(x, y) = \begin{cases} 1 & \text{nếu } (x, y) \in R \\ 0 & \text{nếu } (x, y) \notin R \end{cases}$$

c. Quan hệ mờ: cho R là tập con của không gian tích $X \times Y$, R được gọi là quan hệ mờ giữa hai không gian X và Y nếu R được định nghĩa bằng hàm liên thuộc của nó sao cho $\mu_R(x, y)$ có thể lấy giá trị bất kỳ nào trong khoảng [0, 1].

d. Các phép toán trên các quan hệ mờ: Nếu P và Q là hai quan hệ mờ trên không gian tích $X \times Y$ và $Y \times Z$ thì quan hệ mờ R trên không gian tích $X \times Z$ là sự

hợp thành của hai quan hệ mờ P và Q, được viết R = P∘Q. Toán tử hợp thành mờ gồm ba loại thông dụng: max-min, max-product, min-max.

Toán tử hợp thành	Hàm liên thuộc của quan hệ mờ R
max-min	$\mu_R(x, z) = \mu_{P \circ Q}(x, z) = \max \min[\mu_P(x, y), \mu_Q(y, z)]$
min-max	$\mu_R(x, z) = \mu_{P \circ Q}(x, z) = \min \max[\mu_P(x, y), \mu_Q(y, z)]$
max-product	$\mu_R(x, z) = \mu_P \mu_Q(x, z) = \max[\mu_P(x, y) \mu_Q(y, z)]$

Vd 6.3: Trong lĩnh vực trí tuệ nhân tạo, chương trình thiết kế chia ra làm 4 môn học tự chọn: lý thuyết mờ (Fuzzy Theory), điều khiển mờ (Fuzzy Control), mạng neuron nhân tạo (Neuron Network) và hệ chuyên gia (Expert System) ứng với các tính chất của chúng là : theory, application, hardware, programme. Có 3 sinh viên Tony, Stephany, và Jack muốn chọn một môn học mình ưa thích nhất trong số các môn học nêu trên.

Yêu cầu: Hãy sử dụng các quan hệ mờ để giúp họ đưa ra quyết định đúng đắn.

- Gọi X = {T, S, J} là tập chứa tên các sinh viên, Y = {y1, y2, y3, y4} là tập chứa tên các tính chất môn học theo thứ tự đó, Z = {FT, FC, NN, ES} là tập chứa tên các môn học.

- Nếu các quan hệ mờ của P và Q được cho trong các không gian tích X×Y và Y×Z như sau:

$$P(x,y) = \begin{array}{c} \\ T \\ S \\ J \end{array} \begin{array}{c} \begin{array}{cccc} y1 & y2 & y3 & y4 \end{array} \\ \begin{bmatrix} .2 & 1.0 & 0.8 & 0.1 \\ .0 & 0.1 & 0.0 & 0.5 \\ .5 & 0.9 & 0.5 & 1.0 \end{bmatrix} \end{array}$$

$$Q(y,z) = \begin{array}{c} \\ y1 \\ y2 \\ y3 \\ y4 \end{array} \begin{array}{c} \begin{array}{cccc} FT & FC & NN & ES \end{array} \\ \begin{bmatrix} .0 & 0.5 & 0.6 & 0.1 \\ .2 & 1.0 & 0.8 & 0.8 \\ .0 & 0.3 & 0.7 & 0.0 \\ .1 & 0.5 & 0.8 & 1.0 \end{bmatrix} \end{array}$$

- Ma trận quan hệ R(x,z) được tính sử dụng phép toán max-min như sau:

$$R = P \circ Q = \begin{array}{c} \\ T \\ S \\ J \end{array} \begin{array}{c} \begin{array}{cccc} FT & FC & NN & ES \end{array} \\ \begin{bmatrix} .2 & 1.0 & 0.8 & 0.8 \\ .0 & 0.5 & 0.6 & 0.5 \\ .5 & 0.9 & 0.8 & 1.0 \end{bmatrix} \end{array}$$

- Từ ma trận quan hệ trên ta rút ra kết luận: Tony thích nhất là môn học điều khiển mờ, Stephany thích nhất là môn lý thuyết mờ và Jack thích nhất là môn hệ chuyên gia.

3. Các phương pháp mờ hoá và giải mờ

a. Phương pháp mờ hoá : (fuzzification) Mờ hoá là quá trình làm mờ một đại lượng rõ. Chúng ta sẽ nghiên cứu một vài phương pháp mờ hoá điển hình.

- **Phương pháp mờ hoá trực giác :** Là phương pháp dựa trên kinh nghiệm và sự hiểu biết của con người để phát triển các hàm liên thuộc chuyển đổi các đại lượng rõ sang các đại lượng mờ.

- **Phương pháp mờ hoá suy diễn :** Là phương pháp dựa trên luật để phát triển các hàm liên thuộc.

Vd 6.4: Sử dụng phương pháp mờ hoá suy diễn để phát triển các hàm liên thuộc xấp xỉ các hình tam giác. Gọi [redacted] là ba góc trong của hình tam giác, với điều kiện [redacted]0, và cho U là không gian góc của các hình tam giác:

U = {([redacted]) \ [redacted]0; [redacted]180}

Ta định nghĩa các tập mờ của các loại tam giác như sau:

I (isosceles)	: xấp xỉ tam giác cân
E (equilateral)	: xấp xỉ tam giác đều
R (right)	: xấp xỉ tam giác vuông
IR (isosceles-right)	: xấp xỉ tam giác vuông cân
O (other)	: các loại tam giác khác

Nếu I là tập mờ của tam giác cân, thì hàm liên thuộc của nó có thể được định nghĩa là: $\mu_I([redacted]) = 1 - (1/60^0)\min([redacted])$.

Nếu R là tập mờ của tam giác vuông, thì hàm liên thuộc của nó có thể được định nghĩa là: $\mu_R([redacted]) = 1 - (1/90^0)|A[redacted]90^0|$

Nếu IR là tập mờ của tam giác vuông cân, thì hàm liên thuộc của nó có thể suy diễn được nhờ luật giao của hai tập mờ I và R là: IR = I[redacted]R

$$\mu_{IR}([redacted]) = \min(\mu_I([redacted]), \mu_R([redacted]))$$

$$= 1 - \max(1/60^0 \min([redacted]), 1/90^0 |A[redacted]90^0|)$$

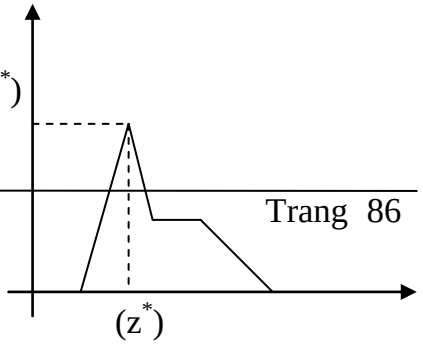
Nếu E là tập mờ của tam giác đều thì hàm liên thuộc của nó có thể được định nghĩa là: $\mu_E([redacted]) = 1 - 1/180^0([redacted])$

Vậy đối với trường hợp cho tất cả các loại tam giác khác, hàm liên thuộc của chúng có thể suy diễn được nhờ luật giao của các phép bù của tam giác cân, vuông và đều là: O = (not I)[redacted](not R)[redacted](not E)

$$\mu_O([redacted]) = \min(1 - \mu_I([redacted]), 1 - \mu_R([redacted]), 1 - \mu_E([redacted]))$$

$$= 1/180^0 \min(3([redacted], 3([redacted]), 2|A[redacted]90^0|, [redacted])$$

b. Phương pháp giải mờ : (defuzzification) Giải mờ là quá trình biến đổi từ các đại lượng mờ sang các đại lượng rõ. Đầu ra của một hệ thống mờ là sự hợp nhau của nhiều tập mờ đầu ra trên cùng biến đầu ra. Chúng ta sẽ nghiên cứu một số phương pháp giải mờ như sau:



- **Phương pháp giải mờ cực đại :**

Phương pháp giải mờ cực đại còn gọi là phương pháp giải mờ độ cao nghĩa là lấy giá trị rõ tại điểm cực đại của tập mờ đầu ra hệ thống.

- **Phương pháp giải mờ điểm trọng tâm :**

Phương pháp giải mờ điểm trọng tâm hay còn gọi là phương pháp trọng tâm vùng, tức là lấy giá trị rõ tại điểm trọng tâm vùng hợp nhau của nhiều tập mờ đầu ra. Phương pháp được cho bởi biểu thức đại số:

$$z^* = \frac{\int \mu(z) z dz}{\int \mu(z) dz}$$

với ∫ là dấu tích phân đại số; z là biến ngôn ngữ đầu ra; z^* là giá trị rõ lấy được.

Phương pháp giải mờ điểm trọng tâm cũng được cho bởi biểu thức đại số dưới dạng rời rạc :

$$z^* = \frac{\sum \mu(z) z}{\sum \mu(z)}$$

với ∑ là dấu tổng đại số.

4. Logic mờ (fuzzy logic)

a. Logic rõ : Là logic hai chữ số 0 và 1. T: x ∈ X → [0, 1]

Cho hai đề xuất I và J, các giá trị chân lý của các đề xuất này được cho như sau:

if y ∈ A, T(I) = 1; else T(I) = 0

if y ∈ B, T(J) = 1; else T(J) = 0

Phép hợp: I ∪J: y ∈ A hoặc y ∈ B → T(I ∪J) = max(T(I),T(J))

Phép giao: I ∩J: y ∈ A và y ∈ B → T(I ∩J) = min(T(I),T(J))

Phép phủ định: if T(I) = 1, then T(¬I) = 0

if T(I) = 0, then T(¬I) = 1

Phép kéo theo: (I → J): y ∈ A hoặc y ∈ B → T(I → J) = T(¬I ∪J)

Phép tương đương: (I ↔ J) : T(I ↔ J) = $\begin{cases} 1 & \text{cho } T(I) = T(J) \\ 0 & \text{cho } T(I) \neq T(J) \end{cases}$

b. Logic mờ : Là sự mở rộng của logic rõ hay còn gọi là logic nhiều chữ số, vì giá trị chân lý của một đề xuất trong không gian có thể lấy giá trị bất kỳ trong khoảng [0, 1] mà không bị giới hạn bởi hai chữ số 0 và 1. Giả sử đề xuất I trong tập mờ A, thì giá trị chân lý của đề xuất I được ký hiệu là T(I), có thể được cho bởi T(I) = $\mu_A(x)$ với 0 ≤ $\mu_A(x)$ ≤ 1

Nếu định nghĩa giá trị chân lý của đề xuất I trong tập A đó là:

$$T_N(I) = \{0,\ldots,i/(N-1),\ldots,1\}$$

với N là số nguyên và 0 ■ i ■ N.

Trong logic rõ: N = 2 (hai chữ số 0 và 1) và i = 0, do đó $T_2(I) = \{0,1\}$

Trong logic mờ: N = nhiều chữ số cụ thể như sau:

N = 3, i = 1, thì T3(I) = {0, 1/2, 1} ứng với {False, May-be, True}

N = 4, i = 1, 2, thì T4(I) = {0, 1/3, 2/3, 1} ứng với {False, Almost-False, Almost-True, True}

N = 5, i = 1, 2, 3, thì T5(I) = {0, 1/4, 2/4, 3/4, 1} ứng với {False, Almost-False, May-be, Almost-True, True}.

Giống như trong logic rõ, logic mờ có các phép toán logic như sau:

Phép phủ định: T(not I) = T(■I) = 1 – T(I)

Phép hợp: I ■J: y is A or B ■ T(I ■J) = max(T(I), T(J))

Phép giao: I ■J: y is A and B ■ T(I ■J) = min(T(I), T(J))

Phép kéo theo:

I ■ J: y is A then y is B T(I ■ J) = T(not I ■J) = max(T(not I), T(J))

c. Lý giải xấp xỉ mờ : (cách xác định tập mờ đầu ra B')

Nếu luật có dạng **if x is A then y is B**, tương đương với quan hệ mờ

R = (A■B)■(notA■Y) và hàm liên thuộc của quan hệ này được định nghĩa bởi:

$$\blacksquare(x,y) = \max\left|\blacksquare(x)\blacksquare(y),(1\blacksquare(x))\right|$$

Do đó nếu biết tập mờ A', ta có thể xác định tập mờ đầu ra B' sử dụng công thức:

$$B' = A'_oR = A'_o((A\blacksquare B)\blacksquare(notA\blacksquare Y))$$

Nếu luật có dạng if x is A then y is B else y is C thì nó tương đương với quan hệ mờ R = (A■B)■(notA■C) và hàm liên thuộc của quan hệ này được cho bởi:

$$\blacksquare(x,y) = \max\left|\blacksquare(x)\blacksquare(y),(1\blacksquare(x)\blacksquare(y)\right|$$

Tương tự, nếu biết được tiền điều kiện A', ta có thể tính được tập mờ B' thông qua công thức:

$$B' = A'_oR = A'_o((A\blacksquare B)\blacksquare(notA\blacksquare C))$$

với $_o$ là toán tử hợp thành max-min hoặc max-product.

Vd 6.5: Để phát triển bộ điều khiển nhiệt độ ta xây dựng luật điều khiển nhiệt độ trong phòng có dạng if x is A then y is B, với A là tập mờ đầu vào biểu diễn nhiệt độ nóng trong ^{0}F và B là tập mờ đầu ra biểu diễn tốc độ của quạt gió mở tối đa. A và B được định nghĩa:

$$A = \{0.0/60 + 0.1/70 + 0.7/80 + 0.9/90 + 1.0/100\}$$

$$B = \{0.0/0 + 0.2/1 + 0.5/2 + 0.9/3 + 1.0/4\}$$

a. Hãy xác định quan hệ mờ R: A → B

b. Hãy xác định tập mờ đầu ra B' nếu A' là tập mờ đầu vào biểu diễn nhiệt độ hơi nóng là {0.0/60 + 0.2/70 + 1.0/80 + 1.0/90 + 1.0/100} dùng phép hợp thành max-min.

Quá trình thực hiện:

a. Theo trên ta có R = (A→B)∩(notA→Y)

$$A \to B = \begin{bmatrix} .0 \\ .1 \\ .7 \\ .9 \\ .0 \end{bmatrix} \begin{bmatrix} .0 & 0.2 & 0.5 & 0.9 & 1.0 \end{bmatrix} = \begin{bmatrix} .0 & 0.0 & 0.0 & 0.0 & 0.0 \\ .0 & 0.1 & 0.1 & 0.1 & 0.1 \\ .0 & 0.2 & 0.5 & 0.7 & 0.7 \\ .0 & 0.2 & 0.5 & 0.9 & 0.9 \\ .0 & 0.2 & 0.5 & 0.9 & 1.0 \end{bmatrix}$$

$$notA \to Y = \begin{bmatrix} .0 & 1.0 & 1.0 & 1.0 & 1.0 \\ .9 & 0.9 & 0.9 & 0.9 & 0.9 \\ .3 & 0.3 & 0.3 & 0.3 & 0.3 \\ .1 & 0.1 & 0.1 & 0.1 & 0.1 \\ .0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\to R(x,y) = \begin{bmatrix} .0 & 1.0 & 1.0 & 1.0 & 1.0 \\ .9 & 0.9 & 0.9 & 0.9 & 0.9 \\ .3 & 0.3 & 0.5 & 0.7 & 0.7 \\ .1 & 0.2 & 0.5 & 0.9 & 0.0 \\ .0 & 0.2 & 0.5 & 0.9 & 1.0 \end{bmatrix}$$

b. Xác định B' nếu có đầu vào A' sử dụng công thức: $B' = A' \circ_0 R$.

$$B' = \begin{bmatrix} .0 & 0.2 & 1.0 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} .0 & 1.0 & 1.0 & 1.0 & 1.0 \\ .9 & 0.9 & 0.9 & 0.9 & 0.9 \\ .3 & 0.3 & 0.5 & 0.7 & 0.7 \\ .1 & 0.2 & 0.5 & 0.9 & 0.0 \\ .0 & 0.2 & 0.5 & 0.9 & 1.0 \end{bmatrix}$$

$$\xrightarrow{\text{max-min}} B' = \begin{bmatrix} .3 & 0.3 & 0.5 & 0.9 & 1.0 \end{bmatrix}$$

hay B' = {0.3/0 + 0.3/1 + 0.5/2 + 0.9/3 + 1.0/4}

d. Cơ sở tri thức mờ : (fuzzy knowledge base)

Cơ sở tri thức mờ được thiết kế dưới dạng n luật mờ if-then tổng quát như sau:

if $A_{11}\ldots A_{1i}\ldots A_{1n}$ then $B_{11}\ldots B_{1j}\ldots B_{1m}$

if $A_{21}\ldots A_{2i}\ldots A_{2n}$ then $B_{21}\ldots B_{2j}\ldots B_{2m}$

……

if $A_{k1}\ldots A_{ki}\ldots A_{kn}$ then $B_{k1}\ldots B_{kj}\ldots B_{km}$

……

if $A_{N1}…A_{Ni}…A_{Nn}$ then $B_{N1}...B_{Nj}…B_{Nm}$

Nếu là hệ có nhiều đầu vào và nhiều đầu ra, khi m ■ 2 và n ■ 2 và nếu là hệ có nhiều đầu vào và một đầu ra khi m = 1 và n ■ 2.

Nếu luật có dạng: if x is A_1 and A_2…and A_L then y is B_S,

Ta có thể viết lại dạng:

if A_S then B_S

với $A_S = A_1 ■ A_2 ■ … ■ A_L$

và hàm liên thuộc của nó được cho bởi:

$■A_S(x) = \min[■A_1(x), ■A_2(x),…, ■A_L(x)]$

Nếu luật có dạng: if x is A_1 or A_2…or A_L then y is B_S

Ta có thể viết lại dạng:

if A_S then B_S

với $A_S = A_1 ■ A_2 ■ … ■ A_L$

và hàm liên thuộc của nó được cho bởi:

$■A_S(x) = \max[■A_1(x), ■A_2(x),…, ■A_L(x)]$

IV. ỨNG DỤNG CỦA LOGIC MỜ VÀO LÝ THUYẾT ĐỒ THỊ

Lý thuyết đồ thị có những ứng dụng rộng rãi đối với những vấn đề trong phân tích hệ thống, giao thông đi lại, kinh tế…Tuy nhiên, trong một số trường hợp những khía cạnh của việc vận dụng lý thuyết đồ thị có thể không chắc chắn. Chẳng hạn như thời gian di chuyển của các phương tiện hoặc sức chứa các phương tiện đi lại trên một mạng lưới đường giao thông không thể được xác định một cách chính xác. Trong những trường hợp này tất nhiên chúng ta phải đối mặt với những vấn đề không chắc chắn sử dụng phương pháp của logic mờ như đã được giới thiệu ở trên. Phần này sẽ trình bày một phân loại của các đồ thị mờ (fuzzy graph) cung cấp hàng loạt những kiểu khác nhau của "fuzziness" có thể trong đồ thị.

a. Đồ thị mờ (fuzzy graphs)

Trong trường hợp này chúng ta chỉ xét đồ thị có hướng và ở đây nhắc lại một số khái niệm về đồ thị (không phân biệt giữa đồ thị rõ-crisp graphs và đồ thị mờ-fuzzy graphs)

Một đồ thị G bao gồm tập đỉnh V và tập cạnh E

$$G = (V,E)$$

$$\text{với } V = \{v1, v2,…, vnV\}$$

$$E = \{e1, e2,…, enE\}$$

với n_V là số đỉnh, n_E là số cạnh, mỗi cạnh có một head(h) và một tail(t)

$$h_i = \text{head}(e_i)$$

$$t_i = \text{tail}(ei)$$

Trong đồ thị có trọng số, mỗi cạnh đều có trọng số w (được gọi là chiều dài cạnh-length)

$$w_i = W(ei)$$

Một đường đi P (path) là một chuỗi tiếp nối các cạnh

$$P = (e_{i1}, e_{i2},\ldots e_{in})$$

ở đó head của mỗi cạnh này bằng tail của mỗi cạnh tiếp theo

$$h_{ik} = t_{ik+1} \text{ với } k = 1,\ldots,(n-1)$$

Đầu của đường đi là h_{in} và đuôi của đường đi là t_{i1}

$$h_{in} = \text{head}(P)$$

$$t_{i1} = \text{tail}(P)$$

Nếu đồ thị có trọng số thì đường đi có chiều dài là tổng các trọng số của các cạnh trên đường đi

$$l_P \text{ } \blacksquare ength(P) \blacksquare \blacksquare_{e_k}$$

Để tạo điều kiện cho việc diễn đạt những kiểu của đồ thị mờ được thảo luận trong phần tiếp theo ta quy định như sau:

$$A = A_1 \backslash \blacksquare + A_2 \backslash \blacksquare + A_3 \backslash \blacksquare \ldots$$

điều này có nghĩa là : $\blacksquare(x) \blacksquare \begin{cases} \max\{\blacksquare\} & \text{khi } x = A_i \\ 0 & \text{khi } x \neq A_i \end{cases}$

b. Phân loại đồ thị mờ

Bây giờ chúng ta sẽ phân loại một số kiểu mờ hoá trong đồ thị.

- **Loại I: Tập mờ của đồ thị rõ**

Một loại tầm thường của đồ thị mờ phát sinh từ việc xem xét một tập mờ G của những đồ thị rõ G_i:

$$G = G_1 \backslash \blacksquare + G_2 \backslash \blacksquare + G_3 \backslash \blacksquare + \ldots + G_{nG} \backslash \blacksquare_G$$

Loại mờ hoá kiểu này thực sự không được quan tâm trừ khi những đồ thị G_i có một vài đỉnh hoặc cạnh tương đồng . Thậm chí khi xảy ra điều này thì việc phân tích vẫn gặp nhiều khó khăn trừ khi sự tương đồng này có một cấu trúc chuẩn bình thường

Minh hoạ loại I: Bạn muốn có một vài sự thay đổi hệ thống điện trong nhà bạn. Tuy nhiên nhà thiết kế đã làm lẫn lộn các hồ sơ của tất cả các ngôi nhà trong khu chung cư mà bạn đang ở. Do đó bạn có thể nhận được một bản copy của những hệ thống điện cho tất cả các ngôi nhà trong khu chung cư của bạn nhưng sẽ không có cách nào để phân biệt bản thiết kế hệ thống điện nào tương ứng với ngôi nhà nào.

Trường hợp được quan tâm nhiều nhất chúng ta gọi là loại I' xảy ra khi mỗi đồ thị rõ G_i có cùng tập đỉnh

$$V = V_1 = V_2 = V_3 = \ldots = V_{nG}$$

Minh hoạ loại I': Bạn được nhận hai bản đồ là hai bản kế hoạch thiết kế đường đi bằng ô tô ngắn nhất từ thành phố này sang thành phố khác. Hai bản đồ này được vẽ trong hai giai đoạn khác nhau và vì vậy có mạng lưới đường đi khác nhau. Nhưng thật không may là trên cả hai bản đồ đều không có bất cứ một dấu hiệu nào cho thấy bản đồ nào được thiết kế gần đây nhất.

- **Loại II: Tập đỉnh rõ và tập cạnh mờ**

Điều này xảy ra khi một đồ thị có các đỉnh được biết nhưng không biết các cạnh. Trong trường hợp này tập đỉnh là rõ còn tập cạnh là mờ

$$V = \{v_1, v_2,\ldots, v_{nV}\}$$

$$E = e_1\backslash\blacksquare + e_2\backslash\blacksquare + \ldots + e_{nE}\backslash\blacksquare$$

ở đây mỗi một cạnh e_i là rõ (nó sẽ có head, tail và trọng số xác định)

Minh hoạ loại II: Bạn phải lên kế hoạch tìm đường đi ngắn nhất bằng ô tô từ thành phố này sang thành phố khác. Nhưng thật không may là có một số đường đang nằm trong diện nâng cấp sửa chữa và sẽ bị phong toả trong suốt thời gian thi công mà chúng ta lại không biết chắc chắn đấy là những con đường nào.

- **Loại III: Tập đỉnh và tập cạnh rõ, kết nối mờ**

Trái với đồ thị mờ loại II, đồ thị loại này có tập cạnh và tập đỉnh đã biết, nhưng lại không biết sự kết nối cạnh. Như vậy ở đây cả tập đỉnh và tập cạnh là rõ nhưng chính bản thân các cạnh lại có head và tail mờ.

$$V = \{v_1, v_2,\ldots, v_{nV}\}$$

$$E = \{e_1, e_2,\ldots,e_{nE}\}$$

$$h_i = h_{i,1}\backslash\blacksquare + h_{i,2}\backslash\blacksquare + \ldots + h_{i,nV}\backslash\blacksquare \text{ với } i = 1,\ldots,n_E$$

$$t_i = t_{i,1}\backslash\blacksquare + t_{i,2}\backslash\blacksquare + \ldots + t_{i,nV}\backslash\blacksquare \text{ với } i = 1,\ldots,n_E$$

Minh hoạ loại III: Bạn phải lên kế hoạch tìm đường đi ngắn nhất bằng ô tô từ thành phố này sang thành phố khác. Một số đường đi có liên quan trong kế hoạch của bạn lại phải cần sử dụng kết hợp phương tiện phà để băng qua một số con sông. Nhưng thật không may là kế hoạch khởi hành và dừng của những chuyến phà lại không rõ ràng.

- **Loại IV: Tập đỉnh mờ và tập cạnh rõ**

Thuộc dạng tương đồng với đồ thị mờ loại II, điều này xảy ra khi một đồ thị có tập đỉnh không được xác định nhưng lại biết rõ tập cạnh. Trong trường hợp này tập đỉnh là mờ và tập cạnh là rõ

$$V = v_1\backslash\blacksquare + v_2\backslash\blacksquare + \ldots + v_{nV}\backslash\blacksquare$$

$$E = \{e_1, e_2,\ldots,e_{nE}\}$$

Minh hoạ loại IV: Bạn sẽ báo cáo tại một vài hội nghị liên tiếp và cần phải xác định lộ trình chuyến đi như thế nào để đến các hội nghị với chi phí tiết kiệm nhất. Tuy nhiên ban tổ chức hội nghị vẫn chưa tiết lộ địa điểm tương ứng để tổ chức các hội nghị.

- **Loại V: Đồ thị rõ với trọng số mờ**

Đây là một trong những loại được quan tâm nhiều nhất. Đồ thị mờ loại này xảy ra khi đồ thị có tập đỉnh và tập cạnh được biết nhưng trọng số trên mỗi cạnh thì không được biết vì vậy chỉ có trọng số là mờ.

$$w_i = w_{i,1} \backslash \blacksquare_{\mu 1} + w_{i,2} \backslash \blacksquare_{\mu 2} + \ldots$$

Minh hoạ loại V: Bạn phải lên kế hoạch xác định lộ trình đường đi nhanh nhất bằng ô tô từ thành phố này sang thành phố khác, nhưng thật không may là bản đồ không cho biết khoảng thời gian cụ thể nên bạn sẽ không được biết chính xác sẽ mất bao lâu để đi hết bất cứ một con đường cụ thể nào.

V. BÀI TẬP

Bài 1: Cho A là tập mờ biểu diễn vận tốc motor trung bình và B là tập mờ biểu diễn điện áp bình thường.

A = {0.3/20 + 0.6/30 + 0.8/40 + 1/50 + 0.7/60 + 0.4/70}

B = {0.0/1 + 0.3/2 + 0.8/3 + 1/4 + 0.7/5 + 0.4/6 + 0.2/7}

Cho luật if A then B.

a. Tìm quan hệ R: A ■ B

b. Hãy xác định tập mờ B' nếu biết tập mờ A' với A' = {0.4/20 + 0.7/30 + 1/40 + 0.6/50 + 0.3/60 + 0.1/70} dùng toán tử max-min.

Bài 2: Cho tập các luật suy diễn trong môi trường không chắc chắn như sau:

Luật 1: if (a1) and (a2) then (b2)	ct(suy diễn) = 0.8
Luật 2: if (not a3) or (a4) then (b1)	ct(suy diễn) = 1.0
Luật 3: if (a5) or (b1) then (b3)	ct(suy diễn) = 0.7
Luật 4: if (a6) then (b4)	ct(suy diễn) = 0.9
Luật 5: if (a7) and (a8) then (b5)	ct(suy diễn) = 0.8
Luật 6: if (a9) then (b5)	ct(suy diễn) = 0.9
Luật 7: if (b2) then (b6)	ct(suy diễn) = 0.9
Luật 8: if (b3) then (b6)	ct(suy diễn) = 0.9
Luật 9: if (b4) and (b5) then (b6)	ct(suy diễn) = 0.85

với ct(a1) = 0.2; ct(a2) = 0.2; ct(a3) = -0.2; ct(a4) = 0.7; ct(a5) = 0.5; ct(a6) = 0.8; ct(a7) = 0.8; ct(a8) = 0.8; ct(a9) = 0.7.

a. Hãy biểu diễn mạng suy diễn bằng đồ thị

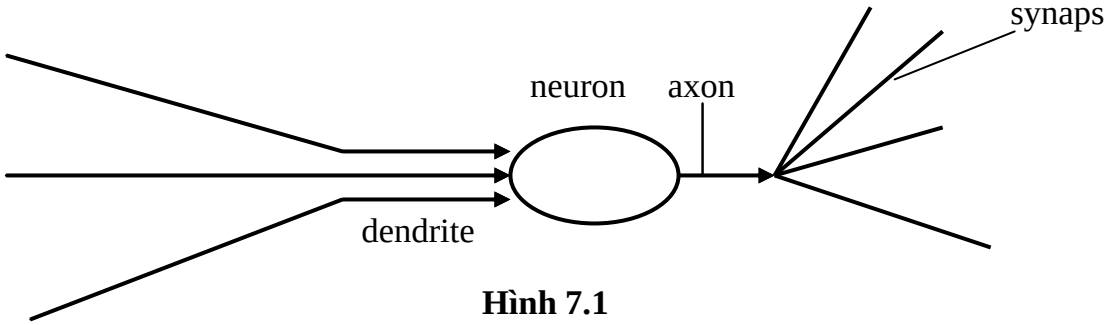
b. Tính các số đo chắc chắn cho các kết luận b2, b3, b4, b5, b6.

CHƯƠNG VII:

MẠNG NEURON NHÂN TẠO

I. LỊCH SỬ PHÁT TRIỂN

Vào thế kỷ thứ 19, khi neuron sinh học đầu tiên của hệ thần kinh con người bắt đầu được nghiên cứu rộng rãi, Cajal (1892) đã xác định rằng hệ thần kinh gồm có những neuron riêng biệt liên lạc với nhau bằng việc gửi những tín hiệu điện áp xuống những axon dài của chúng mà cuối cùng sẽ phân chia và tiếp xúc với dendrite (vùng tiếp thu) của hàng ngàn những neuron khác, truyền những tín hiệu điện xuyên qua synaps (điểm tiếp xúc với điện trở thay đổi). Minh hoạ hình 7.1



Hình 7.1

Tín hiệu điện áp truyền từ neuron này đến neuron khác gồm có hai loại:

+ Điện áp dương được xem như là tín hiệu kích động (excitory) để kích động neuron gửi tín hiệu đến nhiều neuron khác.

+ Điện áp âm được xem như là tín hiệu ức chế (inhibitory) để ức chế neuron gửi tín hiệu đến những neuron khác.

Khi điện áp là 0 thì không có tín hiệu kết nối giữa hai neuron.

Mô hình kết nối của một hệ neuron con người gồm có đầu ra của neuron này kết nối với đầu vào của nhiều neuron khác hoặc kết nối với các đầu vào của neuron chính nó. Cường độ kết nối synapse xác định lượng tín hiệu truyền đến đầu vào dendrite. Giá trị của cường độ synapse được gọi là trọng số. Trong thời gian hệ tiếp xúc một vài đối tượng, một số phần tử cảm biến bị tác động, cường độ kết nối của một số neuron thích hợp trong hệ sẽ được gia tăng nhằm cung cấp toàn bộ thông tin về đối tượng mà hệ đang tiếp xúc và sau đó đưa ra một số quyết định ở lớp neuron đầu ra để điều khiển một vài phần tử cơ. Quá trình này được gọi là huấn luyện hay quá trình học và cường độ kết nối của một số neuron thích hợp được gia tăng theo trong thời gian hệ đang tiếp xúc đối tượng được gọi là luật học.

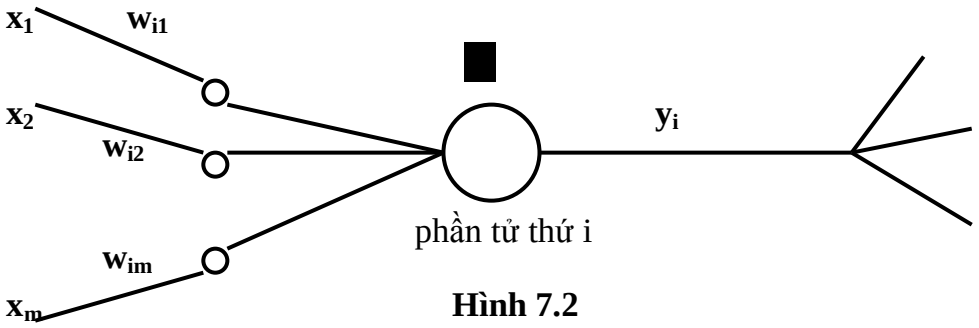
Trong khi chúng ta có thể dễ dàng nghiên cứu chức năng của những neuron riêng lẻ thì lại gặp khó khăn trong việc xác định làm thế nào mà những neuron có thể

làm việc với nhau để đạt được những chức năng bậc cao chẳng hạn như sự nhận thức (perception) và sự hiểu biết (cognition). Tuy nhiên cùng với sự ra đời của máy tính tốc độ cao, cuối cùng người ta cũng có thể xây dựng được những mẫu làm việc của hệ thần kinh cho phép nhà nghiên cứu có thể thử nghiệm trên những hệ thống như vậy để hiểu một cách hoàn chỉnh các thuộc tính và cơ chế hoạt động của nó.

- McCulloch và Pitts (1943) với *binary threshold unit*
- Rosenblatt (1962) với *single-layer perceptron*
- Minsky và Papert (1969) với *multi-layer perceptrons*
- Hopfield (1982) với việc đưa ra quan điểm phân tích mạng dưới dạng *energy function* dẫn đến sự phát triển của các thế hệ tiếp theo
- Ackley, Hinton, & Sejnowski (1985) với *Boltzmann Machine*
- Rumelhart et al (1986) với *backpropagation* - một thủ tục học rất nhanh và với sự ra đời của nó đã thúc đẩy mạng neuron ngày càng phát triển và cung cấp những ứng dụng hữu ích như hiện nay.

II. CÁC THÀNH PHẦN CƠ BẢN CỦA CÁC MẠNG NEURON NHÂN TẠO

Để xây dựng một mạng neuron nhân tạo giống hệ neuron sinh học, năm 1943 McCulloch và Pitts đã đưa ra cấu trúc cơ bản của một neuron thứ i trong mô hình của mạng neuron nhân tạo. Minh hoạ hình 7.2 dưới đây:



Hình 7.2

với : x_j : đầu ra của neuron thứ j hoặc đầu vào từ môi trường bên ngoài

w_{ij} : là trọng số kết nối giữa neuron thứ i và thứ j

: là giá trị ngưỡng của neuron thứ i

y_i : là đầu ra của neuron thứ i được định nghĩa bằng công thức sau:

$$y_i(t + 1) = a\left(x_j(t) \right)$$

Gọi fi : là hàm tổng hợp để tổng hợp tất cả các thông tin đến từ các đầu vào của neuron thứ i được mô tả bằng hàm tuyến tính:

$$f_i = x_j(t)$$

lúc đó ta có đầu ra $y_i(t + 1) = a(f_i)$ là một hàm tác động có thể được mô tả như sau:

$$a(f_i) = \begin{cases} 1 & \text{nếu } f_i \geq 0 \\ 0 & \text{nếu } f_i < 0 \end{cases}$$

Tương tự như mạng neuron sinh học, sẽ có ba trường hợp xảy ra cho trọng số w_{ij} (cường độ kết nối)

+ Nếu $w_{ij} > 0$ tương ứng với tín hiệu truyền kích động
+ Nếu $w_{ij} < 0$ tương ứng với tín hiệu truyền ức chế
+ Nếu $w_{ij} = 0$ không có sự kết nối giữa hai neuron.

Có nhiều kiểu mạng neuron khác nhau nhưng tất cả chúng đều có ba thuộc tính cơ bản:

- Tập các đơn vị (phần tử) xử lý (processing units)
- Tập các mô hình kết nối (connections)
- Thủ tục huấn luyện (training procedure)

Chúng ta sẽ lần lượt tìm hiểu kỹ các thuộc tính này trong những phần tiếp theo

1. Đơn vị xử lý (processing units)

Một mạng neuron chứa một số lượng khổng lồ những đơn vị xử lý (phần tử xử lý) đơn giản tương tự những neuron sinh học trong não. Tất cả những đơn vị xử lý này sẽ hoạt động đồng thời, hỗ trợ song song nhau. Tất cả sự tính toán trong hệ thống được thực hiện bởi những đơn vị xử lý này; không có bất cứ một bộ xử lý nào khác để giám sát hay phối hợp hoạt động của chúng. Mỗi đơn vị xử lý có nhiều đầu vào và một đầu ra. Tại mỗi thời điểm, mỗi đơn vị xử lý sẽ tiến hành thao tác tính toán với một hàm vô hướng của những dữ liệu cục bộ đầu nhập của nó (để thực hiên được điều này một hàm tổng hợp f_i cho mỗi phần tử xử lý thứ i có chức năng tổng hợp các thông tin từ các nguồn bên ngoài hoặc từ nhiều phần tử xử lý khác gửi đến. Hàm này dùng để kết hợp với các đầu vào của mỗi phần tử xử lý thứ i) và truyền kết quả (được gọi là giá trị hoạt hoá – activation value) đến những đơn vị xử lý lân cận của nó. (điều này được thực hiện bằng một hàm tác động hay hàm truyền $a(f_i)$ để kết hợp với đầu ra của mỗi phần tử xử lý thứ i).

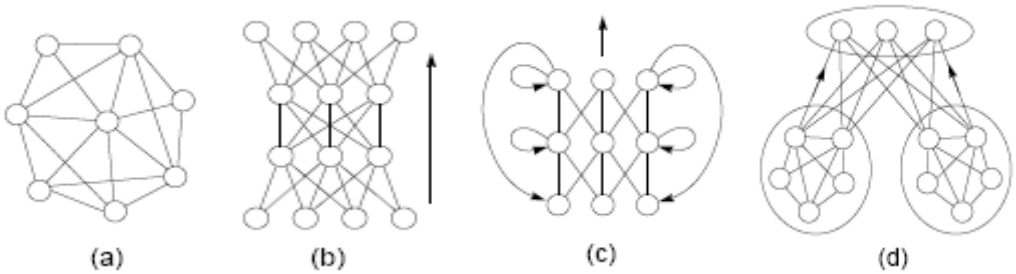
Những đơn vị xử lý hay neuron trong một mạng được chia làm 3 loại tiêu biểu:

- đơn vị nhập (input unit) hay còn gọi là neuron đầu vào: nhận dữ liệu từ môi trường
- đơn vị ẩn (hidden unit) hay neuron ẩn: có thể biến đổi sự biểu diễn dữ liệu bên trong
- đơn vị xuất (output unit) hay neuron đầu ra: có thể trình bày quyết định hoặc điều khiển tín hiệu

2. Mô hình kết nối (connections)

Những đơn vị xử lý trong mạng được tổ chức vào trong những dạng topology được cho dưới đây bằng một tập hợp của những kết nối (connection) hay trọng số (weight) được trình bày dưới dạng những đường kẻ trong biểu đồ. Mỗi trọng số kết nối có một giá trị thực x ($-\blacksquare < x < +\blacksquare$). Giá trị của trọng số kết nối cho biết một đơn vị xử lý ảnh hưởng đến lân cận của nó bao nhiêu.

Giá trị của tất cả các trọng số kết nối định trước phản ứng tính toán của mạng đến bất cứ một mẫu nhập tuỳ ý nào; vì vậy các trọng số kết nối có thể mã hoá bộ nhớ dài hạn (long-term memory) hay kiến thức của mạng. Trọng số có thể thay đổi nhưng có khuynh hướng thay đổi chậm bởi vì kiến thức tích luỹ thay đổi chậm. Điều này tương phản với mẫu hoạt động là những hàm tạm thời của đầu vào hiện hành - một loại của bộ nhớ ngắn hạn (short-term memory)



Hình 7.3: Topology mạng neuron

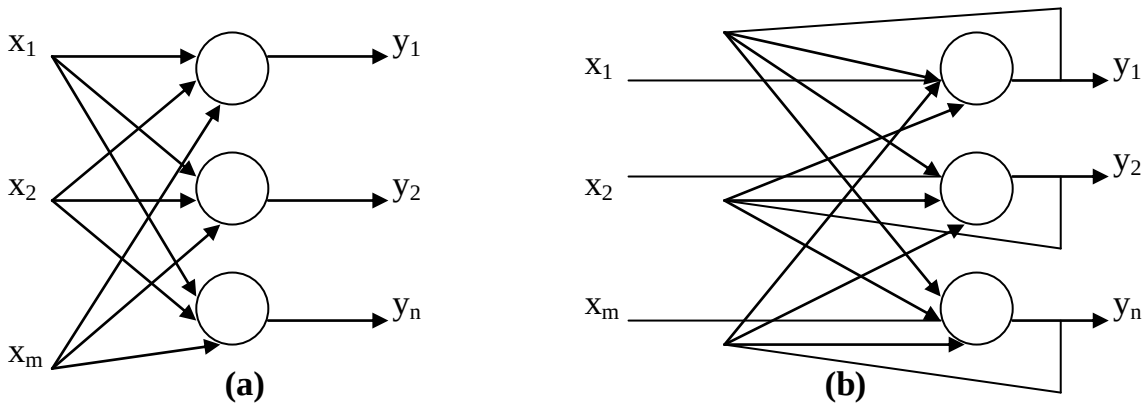
Một mạng có thể được kết nối dưới một trong các dạng topology (phổ biến) trên : (a) unstructured, (b) layered, (c) recurrent, (d) modular. Mỗi loại sẽ thích hợp cho một kiểu ứng dụng cụ thể.

Lưu ý rằng topo mạng loại unstructured có thể chứa vòng kín và vì vậy thực sự nó sẽ được hồi tiếp (recurrent). Nói cách khác mạng loại unstructured sử dụng kết nối hai chiều trong khi những lọai mạng khác sử dụng kết nối một chiều

Hoặc chúng ta có thể xếp các kết nối thành hai dạng chung nhất của mô hình kết nối:

- Mạng nuôi tiến (feedforward network): đầu ra của neuron ở lớp đứng trước chính là đầu vào của các neuron ở lớp đứng sau nó

- Mạng nuôi lùi (feedback network): các đầu ra được định hướng lùi về làm các đầu vào cho các neuron ở cùng lớp hoặc ở lớp đứng trước nó



Hình 7.4

Hình 7.4 (a) là mô hình của mạng neuron nuôi tiến một lớp gồm: lớp các neuron đầu vào từ môi trường bên ngoài, lớp các neuron đầu ra.

Hình 7.4 (b) là mô hình của mạng neuron nuôi lùi một lớp gồm: lớp các neuron đầu vào từ môi trường bên ngoài, lớp các neuron đầu ra. Các đầu vào của mạng chính là các đầu ra thực sự của mạng.

3. Thủ tục huấn luyện (training)

Đây cũng là một thành phần quan trọng của các mạng neuron. Huấn luyện mạng hay còn gọi là việc học cho các mạng theo nghĩa chung nhất có thể được hiểu là làm sao cho thích ứng với những sự kết nối của nó để mạng có thể trình bày sự tính toán mong đợi của nó đối với tất cả các mẫu đầu vào. Tiến trình này bao gồm việc

hiệu chỉnh các trọng số kết nối hay còn gọi là học thông số và việc hiệu chỉnh topo mạng (thêm hoặc xoá một số kết nối trong mạng) còn gọi là học cấu trúc. Ở đây ta chỉ trình bày các phương pháp học thông số.

Giả sử có n phần tử xử lý trong một mạng và mỗi phần tử xử lý có chính xác m trọng số thích nghi. Ma trận trọng số W được định nghĩa như sau:

$$W = \begin{bmatrix} y_1^T \\ y_2^T \\ y_n^T \end{bmatrix} = \begin{bmatrix} y_{11} & w_{12} & w_{1m} \\ y_{21} & w_{22} & w_{2m} \\ y_{n1} & w_{n2} & w_{nm} \end{bmatrix}$$

với : $w_i = (w_{i1}, w_{i2},\ldots, w_{im})^T$, i = 1, 2,…,

n : là vector trọng số của phần tử xử lý thứ i

wij : là trọng số để kết nối phần tử xử lý thứ j và phần tử xử lý thứ i.

Làm thế nào để tìm ra một ma trận trọng số thực sự của mạng xấp xỉ với ma trận mong muốn W đã có trong quá trình xử lý thông tin ?

Luật học thông số tổng quát được phát triển nhằm cập nhật ma trận trọng số sao cho có được một ma trận trọng số thực sự của mạng xấp xỉ với ma trận trọng số mong muốn. Học thông số có thể được chia ra làm ba chế độ học:

- Học giám sát: Trong chế độ này, "người dạy" sẽ cung cấp đích đầu ra cho mỗi mẫu đầu vào và sửa những lỗi của mạng.

- Học nửa giám sát hay học củng cố: Trong chế độ này, "người dạy" chỉ đơn thuần chỉ ra liệu sự đáp lại của mạng đến một mẫu huấn luyện là tốt hay xấu hay không.

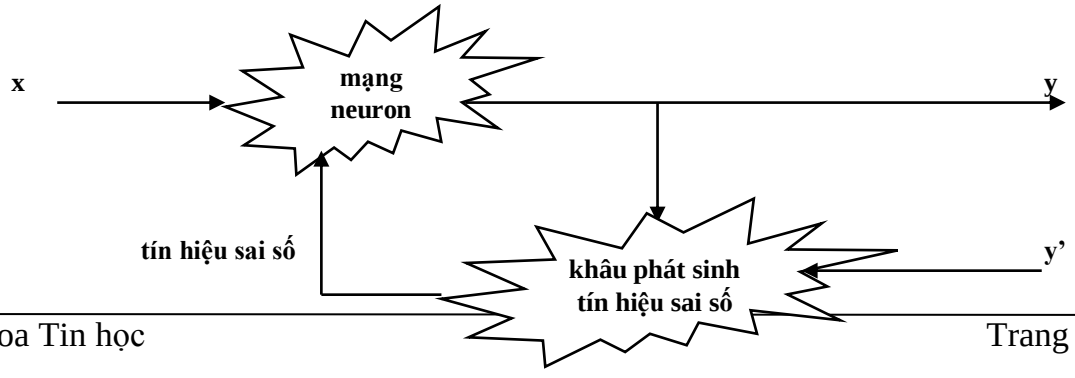
- Học không giám sát: Trong chế độ này, không có mặt người dạy và tự bản thân mạng phải tìm thấy tập các quy tắc trong dữ liệu huấn luyện.

a. Học giám sát : (supervised learning)

Trong cách học này, mạng neuron nhân tạo được cung cấp với một dãy của các cặp đầu vào đầu ra mong muốn: $(x^{(1)}, y'^{(1)})$, $(x^{(2)}, y'^{(2)})$,…, $(x^{(k)}, y'^{(k)})$.

Khi mỗi đầu vào x(k) được đặt vào mạng thì đầu ra mong muốn tương ứng y'(k) cũng được cung cấp đến mạng. Sai lệch giữa đầu ra thật sự y(k) và đầu ra mong muốn y'(k) được giám sát trong công đoạn phát sinh tín hiệu sai số.

Trọng số kết nối giữa các neuron trong mạng sẽ được cập nhật sử dụng tín hiệu sai lệch này sao cho đầu ra thật sự y của mạng sẽ tiến gần đến đầu ra mong muốn y' của mạng

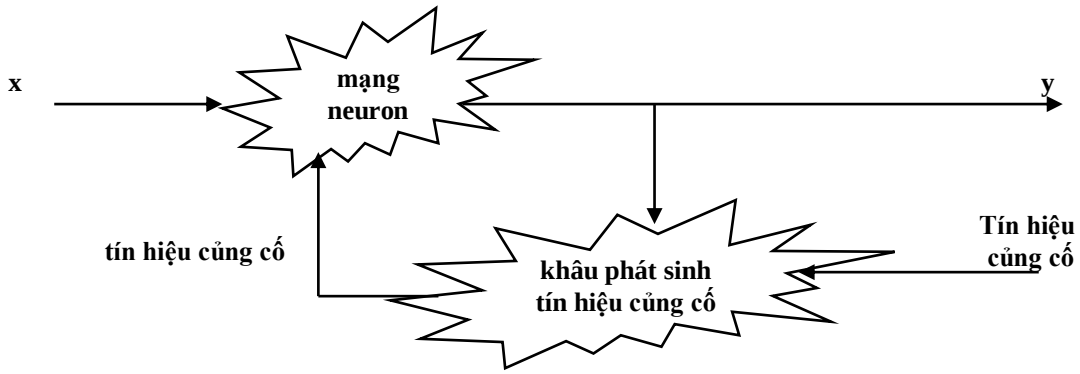


Hình 7.5 *Mô hình cách học giám sát của một mạng neuron nhân tạo*

Học giám sát có thể được áp dụng cho cả hai loại mạng : mạng nuôi tiến và mạng nuôi lùi (hồi tiếp).

b. Học nửa giám sát (semi-supervised learning) **hay học củng cố** (reinforcement learning) **:**

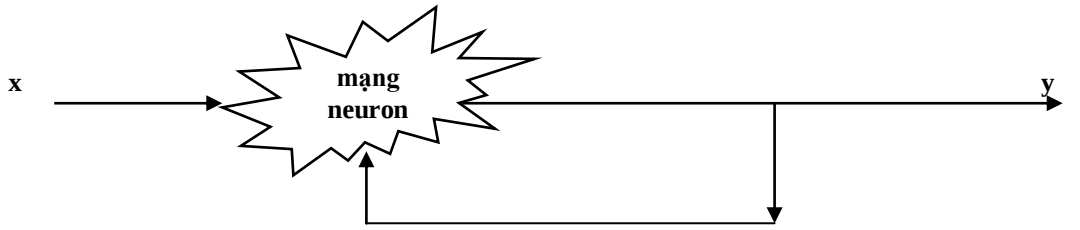
Trong phương pháp học củng cố "người dạy" không cung cấp đích rõ ràng cho đầu ra của mạng. Cụ thể là mạng vẫn được cung cấp các mẫu đầu vào mong muốn nhưng mạng không được cung cấp rõ ràng các mẫu đầu ra mong muốn (ví dụ như mạng chỉ được bảo cho biết rằng đầu ra thật sự hiện có của nó chỉ đúng 50%)mà chỉ đánh giá cách hoạt động của mạng là "tốt" hay "xấu". Có nghĩa là, chỉ có một bit thông tin hồi tiếp xác định đầu ra của mạng đúng hoặc sai. Học dựa trên cơ sở loại thông tin đánh giá này được gọi là học củng cố và thông tin hồi tiếp được gọi là tín hiệu củng cố (reinforcement signal).



Hình 7.6 *Mô hình cách học củng cố của một mạng neuron nhân tạo*

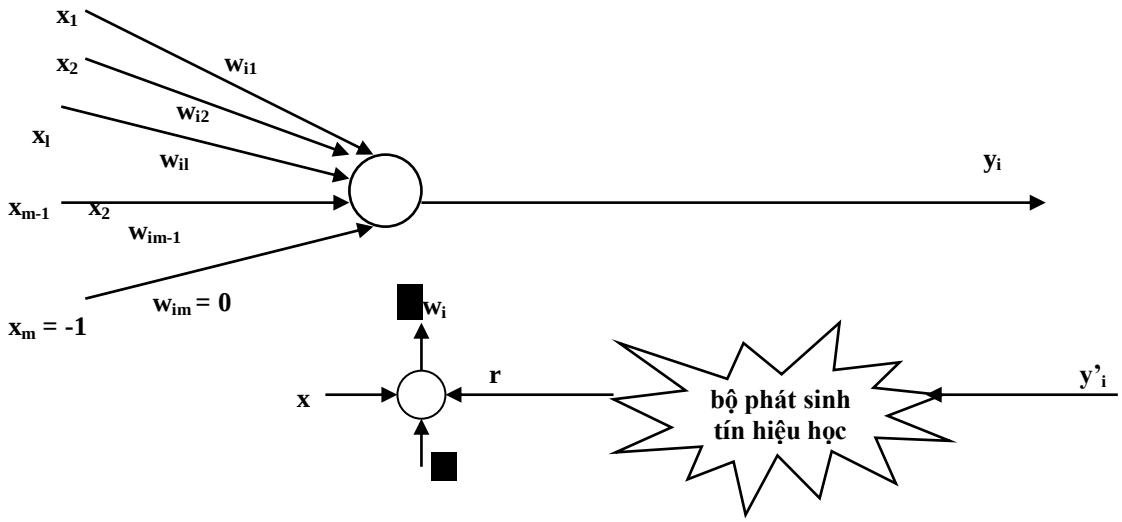
c. Học không giám sát (unsupervised learning) :

Trong cách học không giám sát, không có "người dạy" cung cấp bất kỳ một thông tin hồi tiếp nào. Không có thông tin hồi tiếp từ môi trường để đảm bảo các đầu ra thực sự của mạng đúng hay sai. Nói cách khác trong cách học này, mạng chỉ được cung cấp các mẫu đầu vào và tự điều chỉnh các trọng số kết nối giữa các neuron trong mạng bằng cách sử dụng các mẫu đầu ra thực sự của mạng.



Hình 7.7 *Mô hình cách học không giám sát của một mạng neuron nhân tạo*

Xét một cấu trúc huấn luyện tổng quát cho một phần tử xử lý thứ i trong một mạng neuron nhân tạo được cho dưới đây :



Hình 7.8 *Mô tả luật học thông số tổng quát cho một phần tử xử lý thứ i*

với: đầu vào x_j, j = 1, 2,…, m.

Trong mô hình học tổng quát này, giá trị ngưỡng 0 của neuron thứ i có thể được đưa vào trong việc học. Nó được xem như là một trọng số w_{im} bằng cách gán một giá trị cố định là -1 cho đầu vào x_m. Tín hiệu đầu ra mong muốn y'_i chỉ có mặt trong chế độ học giám sát hoặc trong chế độ học củng cố. Trong hai chế độ học này, các trọng số của phần tử xử lý thứ i được cải tiến sử dụng tín hiệu sai lệch và tín hiệu củng cố (như đã nói ở trên), nhưng trong chế độ học không giám sát, trọng số của phần tử xử lý thứ i được cải tiến chỉ sử dụng tín hiệu đầu ra thực sự của mạng.

Gọi wi(t) là vector trọng số

X(t) là vector các mẫu đầu vào

r là tín hiệu học tại bước học t

Ta có luật học thông số tổng quát trong các mạng neuron nhân tạo được định nghĩa :

$$\Delta w_i(t) = \eta X(t)$$

với $\Delta w_i(t)$: là sự gia tăng của vector trọng số tại bước học

t và η : là số dương, gọi là hằng số học (learning constant) để xác định tốc độ học.

Nên vector trọng số tại bước học (t + 1) có thể được cải tiến bằng công thức:

$$w_i(t + t) = w_i(t) + \eta r X(t)$$

Nếu là chế độ học giám sát, có tín hiệu của giáo viên y'_i, thì tín hiệu học r có thể được biểu diễn bằng một hàm có dạng tổng quát:

$$r = f_r(w_i, X, y'_i) = \text{tín hiệu sai lệch} = y'_i - y_i$$

Nếu là chế độ học củng cố, có tín hiệu của giáo viên y'_i, thì tín hiệu học r có thể được biểu diễn bằng một hàm có dạng tổng quát:

$$r = f_r(w_i, X, y'_i) = \text{tín hiệu củng cố} = y'_i$$

Nếu là chế độ học không giám sát, không có tín hiệu của giáo viên, thì tín hiệu học r có thể được biểu diễn bằng một hàm có dạng tổng quát:

$$r = f_r(w_i, X) = \text{tín hiệu ra thực sự của mạng} = y_i$$

Nếu học trong thời gian liên tục, thì sử dụng phương trình sau để cải tiến vector trọng số tại thời điểm (t + 1)

$$\frac{dw_i(t)}{dt} \quad X(t)$$

Các trọng số được cải tiến tuân theo những tương quan trước và sau, (dựa theo luật học của Hebbian) tín hiệu học r trong luật học tổng quát có thể được thiết lập theo công thức sau:

$$r = a(w^T_i X) = y_i$$

với a( ) là hàm tác động của phần tử xử lý thứ i.

Như vậy sự gia tăng trọng số w_i của vector trọng số sẽ là:

$$w_i \quad (w^T_i X) X \quad X$$

Hay ta có thể nhận thấy các thành phần của vector trọng số được cập nhật với một lượng là :

$$w_{ij} \quad (w^T_i X) x_j \quad x_j, \text{ với } i = 1, 2,\ldots, n; j = 1, 2,\ldots,m$$

Theo trên ta nhận thấy luật học của Hebbian là một luật học không giám sát cho một mạng nuôi tiến

Nếu đề xuất tín hiệu học $r = a(w^T_i X) = y_i = y'_i$ với y'_i là đầu ra mong muốn để điều chỉnh các trọng số, thì các thành phần của vector trọng số sẽ được cập nhật bằng công thức:

$$w_{ij} \quad _i x_j, \text{ với } i = 1, 2, \ldots, n; j = 1, 2,\ldots,m$$

Theo công thức trên ta có thể nhận thấy đây là luật học tổng quát giám sát cho một mạng neuron.

TRÍ TUỆ NHÂN TẠO

Artificial Intelligence (AI)

Nội Dung

- **Chương 1**. Giới thiệuTTNT
- **Chương 2.** Phép tính vị từ
- **Chương 3.** Cấu trúc và chiến lược dùng cho tìm kiếm trên không gian trạng thái (TK-KGTT)
- **Chương 4.** Tìm kiếm heuristic

CHƯƠNG I
GIỚI THIỆU VỀ TRÍ TUỆ NHÂN TẠO

Lịch sử hình thành và phát triển của trí tuệ nhân tạo

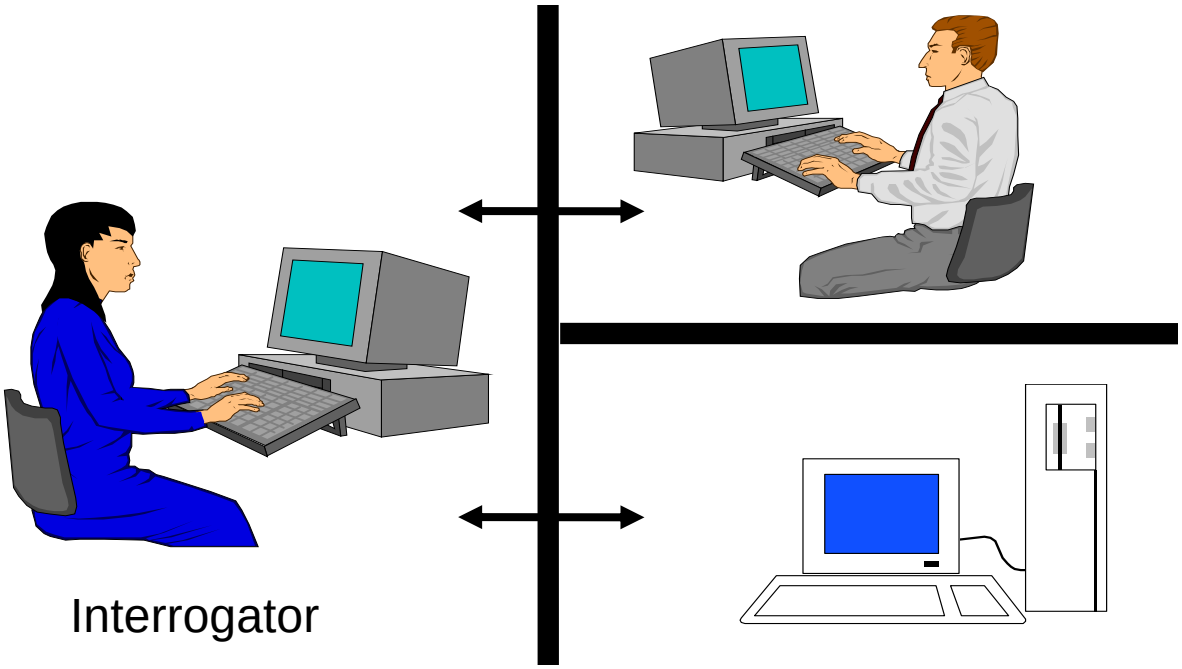
Trí Tuệ Nhân Tạo là gì?

- Là một ngành của khoa học máy tính liên quan đến *sự tự động hóa các hành vi thông minh*.

  Trí tuệ là gì?

- Các câu hỏi chưa có câu trả lời:
  - Liệu trí tuệ có phải là một khả năng duy nhất hay chỉ là một tên gọi cho một tập hợp các hành vi phân biệt và độc lập nhau?
  - Thế nào là khả năng sáng tạo?
  - Thế nào là trực giác?
  - Điều gì diễn ra trong quá trình học?
  - Có thể kết luận ngay về tính trí tuệ từ việc quan sát một hành vi hay không, hay cần phải có biểu hiện của một cơ chế nào đó nằm bên trong?

Turing Test



- Ưu điểm của Turing Test
  - Khái niệm khách quan về trí tuệ
  - Tránh đi những thảo luận về quá trình bên trong và ý thức
  - Loại trừ định kiến thiên vị của người thẩm vấn
- Các ý kiến phản đối

  - Thiên vị các nhiệm vụ giải quyết vấn đề bằng ký hiệu
  - Trói buộc sự thông minh máy tính theo kiểu con người, trong khi con người có:

    + Bộ nhớ giới hạn

    + Có khuynh hướng nhầm lẫn

Tuy nhiên, trắc nghiệm Turing đã cung cấp một cơ sở cho nhiều sơ đồ đánh giá dùng thực sự cho các chương trình TTNT hiện đại.

Các Ứng Dụng của TTNT

1. Trò chơi và các bài toán đố
2. Suy luận và chứng minh định lý tự động
3. Các hệ chuyên gia (các hệ tri thức)
4. Xử lý ngôn ngữ tự nhiên
5. Lập kế hoạch và người máy
6. Máy học
7. Mạng Neuron và giải thuật di truyền
8. …

Một số tổng kết về TTNT

- Sử dụng máy tính vào những *suy luận trên các ký hiệu, nhận dạng qua mẫu, học, và các suy luận khác*…
- Tập trung vào các vấn đề "khó" *không thích hợp với các lời giải mang tính thuật toán*.
- Quan tâm đến các kỹ thuật giải quyết vấn đề sử dụng *các thông tin không chính xác, không đầy đủ, mơ hồ*…
- Cho lời giải 'đủ tốt' *chứ không phải là lời giải chính xác hay tối ưu*.
- Sử dụng những khối lượng lớn tri thức chuyên ngành trong giải quyết vấn đề. *Đây là cơ sở cho các hệ chuyên gia*.
- Sử dụng *tri thức cấp meta* để tăng thêm sự tinh vi cho việc kiểm soát các chiến lược giải quyết vấn đề.
- …

Những vấn đề chưa được giải quyết

- Chương trình chưa tự sinh ra được heuristic
- Chưa có khả năng xử lý song song của con người
- Chưa có khả năng diễn giải một vấn đề theo nhiều phương pháp khác nhau như con người.
- Chưa có khả năng xử lý thông tin trong môi trường liên tục như con người.
- Chưa có khả năng học như con người.
- Chưa có khả năng tự thích nghi với môi trường.

TTNT =
Biểu Diễn + tìm kiếm

TTNT như là sự biểu diễn và tìm kiếm

Sự biểu diễn phải:

- Cung cấp một cơ cấu tự nhiên để thể hiện tri thức/thông tin/ dữ liệu một cách đầy đủ => ***Tính biểu đạt***
- Hỗ trợ việc thực thi một cách hiệu quả việc tìm kiếm đáp án cho một vấn đề => ***Tính hiệu quả***

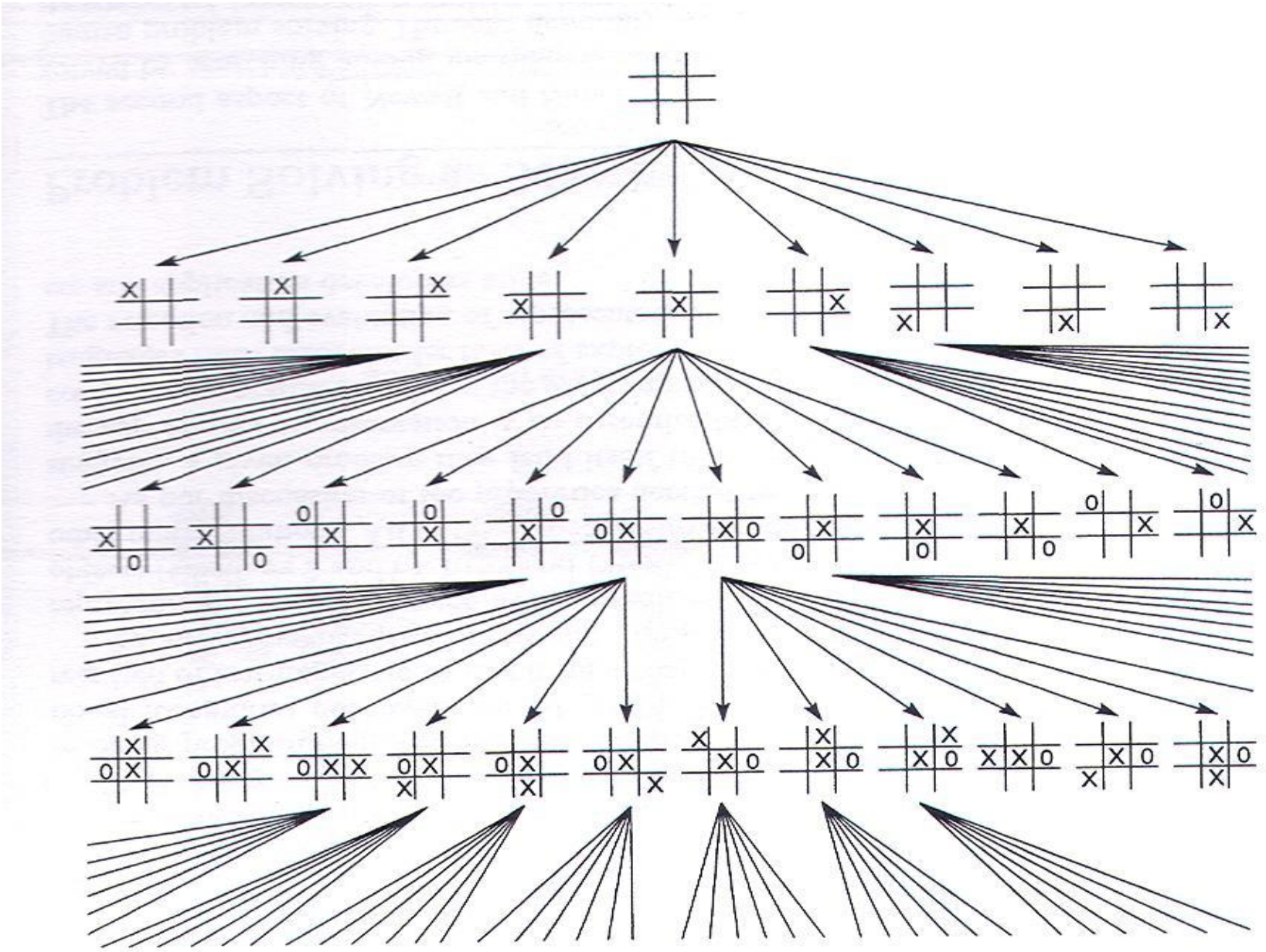
Liệu việc tìm kiếm:

- Có kết thúc không?
- Có chắc chắn sẽ tìm được lời giải không?
- Có chắc chắn sẽ tìm được lời giải tối ưu không?

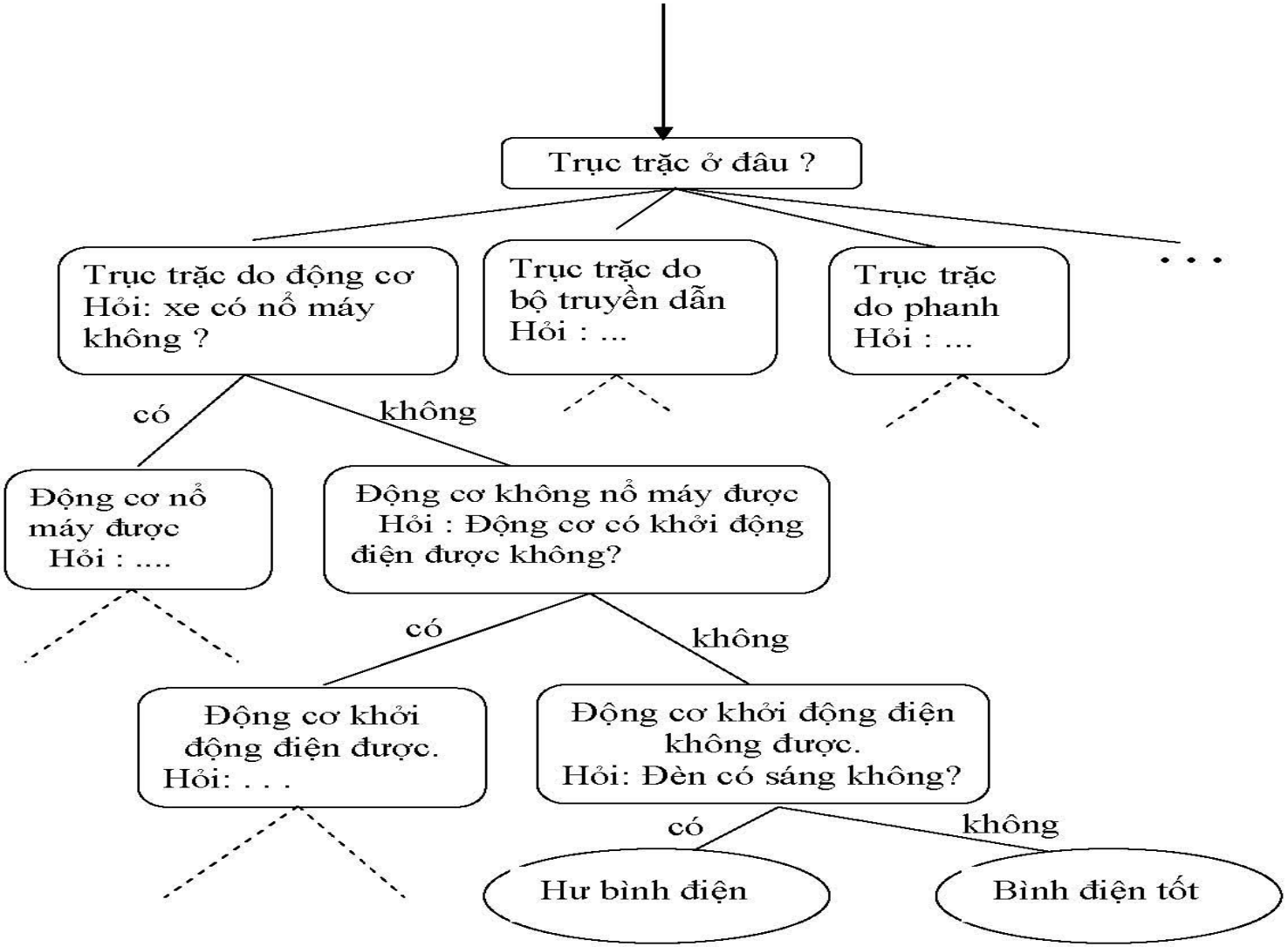
TTNT như là biểu diễn & tìm kiếm

- Giải quyết vấn đề như là sự tìm kiếm lời giải trong một *đồ thị không gian trạng thái*:
  - Nút ~ trạng thái (node ~ state)
  - Liên kết (link)
- Ví dụ:
  - Trò chơi tic-tac-toe
  - Chẩn đoán trục trặc máy móc trong ô tô

KGTT của Trò Chơi Tic-Tac-Toe



Chẩn đoán trục trặc máy móc trong ô tô



Chương 2 – Logic hình thức

- **Logic hình thức**
  - *Logic hình thức = Biểu diễn + suy luận*
  - Logic hình thức như là một cơ chế biểu diễn tri thức
  - Logic hình thức như là tìm kiếm không gian trạng thái trong các đồ thị And/Or
  - Logic hình thức để hình thức hóa các luật heuristic
- **Có hai ngôn ngữ:**
  - Phép tính mệnh đề
  - Phép tính vị từ

Phép tính mệnh đề (1)

- **Mệnh đề:** là một câu khẳng định có thể đúng (true) hoặc sai (false).
- Ví dụ:

Đồng là một kim loại	=>	Đúng
Gỗ là một kim loại	=>	Sai
Hôm nay là thứ Hai	=>	Sai

- Ký hiệu trong phép tính mệnh đề:
  - Ký hiệu mệnh đề: P, Q, R, S,...
  - Ký hiệu chân lý: true, false
  - Các phép toán logic: (hội), (tuyển), (phủ định), (kéo theo) , = (tương đương)

Phép tính mệnh đề (2)

- Định nghĩa ***câu*** trong phép tính mệnh đề:
  - Mỗi ký hiệu mệnh đề (P, Q, …) là một câu.
  - Ký hiệu chân lý (true, false) là một câu.
  - Phủ định của một câu là một câu. (
  - Hội (tuyển (kéo theo (tương đương (=) của hai câu là một câu.
- Ký hiệu ( ), [ ] được dùng để nhóm các ký hiệu vào các biểu thức con.
- Một *biểu thức mệnh đề* được gọi là một *câu* (hay *công thức dạng chuẩn*- WFF) có thể được tạo thành từ những ký hiệu hợp lệ thông qua một dãy các luật trên.

Ví dụ: ((P Q) R) = P R

Ngữ Nghĩa của Phép Tính MĐ

- **Sự thông dịch or sự diễn giải** (Intepretation):
  - Là sự *gán giá trị chân lý* (T / F) cho các câu mệnh đề.
  - Là một sự khẳng định chân lý của các câu mệnh đề.
- Sự thông dịch của một câu kép thường được xác định bằng ***bảng chân lý***:

P	Q	[redacted]	P[redacted]	P[redacted]	P[redacted]	P=Q
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Sự Tương Đương của Phép Tính MĐ

- [redacted] = P
- (P [redacted]) = ([redacted])
- Luật tương phản: (P [redacted]) = ([redacted])
- Luật De Morgan: [redacted]) = ([redacted]), và
  [redacted]) = ([redacted])
- Luật giao hoán: (P [redacted]) = (Q [redacted]), và (P [redacted]) = (Q [redacted])
- Luật kết hợp: ((P [redacted]) [redacted]) = (P [redacted])),
  ((P [redacted]) [redacted]) = (P [redacted]))
- Luật phân phối: P [redacted]) = (P [redacted]) [redacted]),
  P [redacted]) = (P [redacted]) [redacted])

Phép TínhVị Từ (1)

- **Ký hiệu vị từ** là tập hợp gồm các chữ cái, chữ số hay dấu gạch nối (_), và được ***bắt đầu bằng chữ cái***.
  VD: X3, tom_and_jerry
- Ký hiệu vị từ có thể là:
  - **ký hiệu chân lý**: true, false
  - **Hằng**: dùng để chỉ một đối tượng / thuộc tính trong thế giới.
    - Ký hiệu bắt đầu bằng chữ thường: VD: helen, yellow, rain
  - **Biến**: dùng để chỉ một lớp tổng quát các đối tượng / thuộc tính.
    - Ký hiệu bắt đầu bằng chữ HOA VD: X, People, Students
  - **Hàm**: dùng để chỉ một hàm trên các đối tượng.
    - Ký hiệu bắt đầu bằng chữ thường: VD: father, plus
    - Mỗi ký hiệu hàm có n ngôi ,chỉ số lượng các đối số của hàm.
  - **Vị từ**: dùng để định nghĩa một mối quan hệ giữa không hoặc nhiều đối tượng.
    - Ký hiệu vị từ bắt đầu bằng chữ thường. VD: likes, equals, part_of

Phép TínhVị Từ (2)

- **Biểu thức hàm**: là một ký hiệu hàm theo sau bởi n đối số.
  VD: father(david) price(bananas) like(tom, football)
- **Mục** (term): là một hằng, một biến hay một biểu thức hàm
- **Câu sơ cấp**: là một hằng vị từ với n ngôi theo sau bởi n thành phần (mỗi thành phần là một mục) đặt trong dấu (), cách nhau bởi dấu ‘,’ và kết thúc với dấu ‘.’
  - Trị chân lý true, false là các câu sơ cấp.
  - Câu sơ cấp còn được gọi là: biểu thức sơ cấp (atomic expression), nguyên tử (atom) hay mệnh đề (proposition)

  VD: friends(helen, marry). likes(hellen, mary).

  likes(helen, sister(mary)). likes(X, ice-cream).

  Ký hiệu vị từ trong các câu này là friends, likes.

Phép TínhVị Từ (3)

- **Câu**: được tạo ra bằng cách kết hợp các câu sơ cấp sử dụng:
  - **Các phép kết nối logic**: [redacted]
  - **Các lượng tử biến**:
    - **Lượng tử phổ biến** [redacted] dùng để chỉ một câu là đúng với mọi giá trị của biến lượng giá
    - **Lượng tử tồn tại** [redacted] dùng để chỉ một câu là đúng với một số giá trị nào đó của biến lượng giá.

  VD:

  (mọi đứa trẻ đều thích Chocolat)

  - [redacted], thich(dua_tre(X), chocolat).

  (tom có không ít hơn một người bạn)

  - [redacted], friends(Y, tom).

Ngữ Nghĩa của Phép Tính Vị Từ

- **Sự thông dịch** (cách diễn giải) của một tập hợp các câu phép tính vị từ: là một sự gán các thực thể trong miền của vấn đề đang đề cập cho mỗi ký hiệu hằng, biến, vị từ và hàm.
- Giá trị chân lý của một câu sơ cấp được xác định qua sự thông dịch. Đối với các câu không phải là câu sơ cấp, sử dụng bảng chân lý cho cho các phép nối kết, và:
  - Giá trị của câu [redacted]X **<câu>** là true nếu <câu> là T cho tất cả các phép gán có thể được cho X.
  - Giá trị của câu [redacted]X **<câu>** là true nếu tồn tại một phép gán cho X làm cho <câu> có giá trị T.

Phép Tính Vị Từ Bậc Nhất

- **Phép tính vị từ bậc nhất** cho phép các biến tham chiếu đến các đối tượng trong miền của vấn đề đang đề cập nhưng KHÔNG được tham chiếu đến các vị từ và hàm.
- VD không hợp lệ: [redacted]Likes) Likes(helen, ice-cream)
- VD hợp lệ:
  - *Nếu ngày mai trời không mưa, tom sẽ đi biển.*
    - [redacted]eather(tomorrow, rain) [redacted]o(tom, sea)
  - *Tất cả các cầu thủ bóng rổ đều cao.*
    - [redacted]X (basketball_player(X) [redacted]ll(X))
  - *Có người thích coca-cola*
    - [redacted]X person(X) [redacted]ikes(X, coca-cola)
  - *Không ai thích thuế*
    - [redacted]X likes(X, taxes)

Ví dụ về phép tính vị từ

- Cho trước:

  mother(eve,abel) mother(eve, cain)

  father(adam, abel) father(adam,cain)

  [redacted] father(X,Y) [redacted]nother(X,Y) [redacted]arent(X,Y)

  [redacted] parent(Z,X) [redacted]arent(Z,Y) [redacted]bling(X,Y)

- Có thể suy luận:

  parent(eve,abel) parent(eve, cain)

  parent(adam,abel) parent(adam,cain)

  sibling(abel, cain) sibling(cain, abel)

  sibling(abel,abel) sibling(cain,cain) *!không có nghĩa*

Các luật suy diễn

- **Luật Modus Ponens** (MP): P [redacted]

  P

  Q

- **Luật Modus Tolens** (MT): P [redacted]

  [redacted]

  [redacted]

- **Luật triển khai phổ biến** (Universal Instantiation):

  [redacted] P(X)

  a thuộc miền xác định của X

  P(a)

Ví Dụ

"Tất cả mọi người đều chết và Socrates là người, do đó Socrates sẽ chết"

=> [redacted] man(X) [redacted]ortal(X) (1)

man(socrates) (2)

Từ (1),(2) bằng luật UI, ta có:

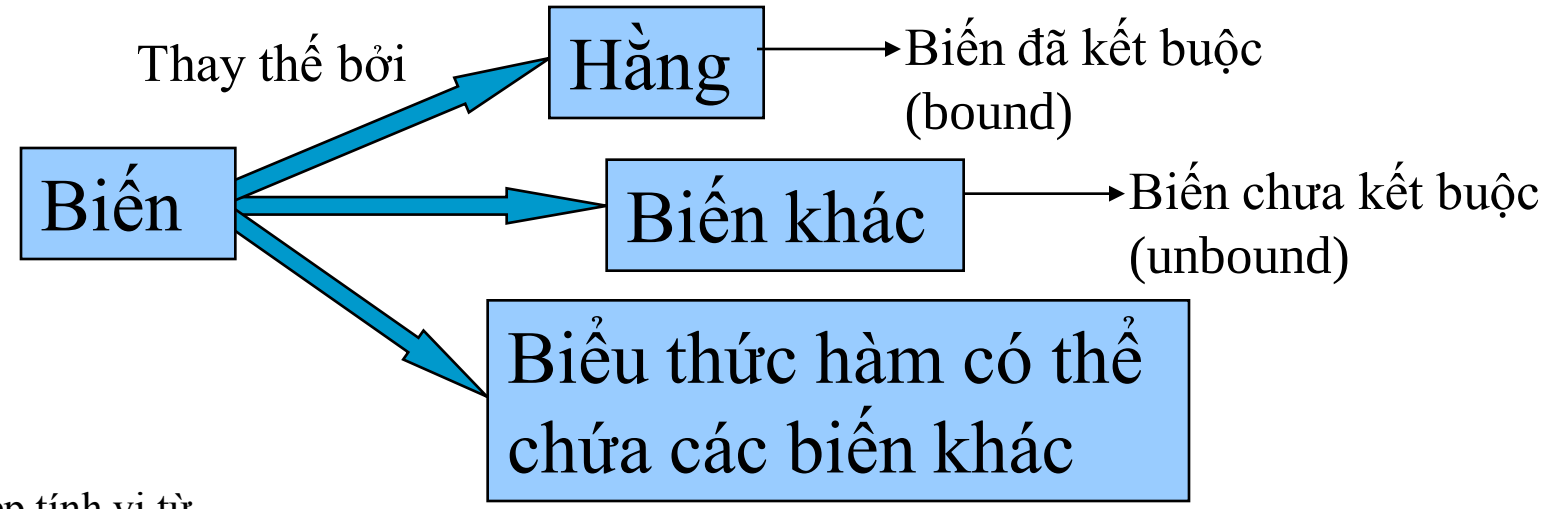
man(socrates) [redacted]ortal(socrates) (3)

Từ (3) và (2) bằng luật MP, ta có:

mortal(bill) (4)

Đối sánh mẫu và phép hợp nhất

- Để áp dụng các luật như MP, một hệ suy diễn phải có khả năng xác định khi nào thì hai biểu thức là ***một*** hay còn gọi là ***đối sánh*** (match).
- **Phép hợp nhất** là một giải thuật dùng để xác định những phép thế (substitution) cần thiết để làm cho hai biểu thức vị từ đối sánh nhau.
- Một biến có thể thay thế bởi một ***mục*** bất kỳ:



"Giải thuật" Đối Sánh Mẫu

1. **Hằng / hằng** đối sánh : chỉ khi chúng giống hệt nhau

   VD: tom không đối sánh với jerry

2. **Hằng a / biến X** đối sánh:

   a. Biến chưa kết buộc: biến trở thành kết buộc với hằng

      => Khi đó ta có phép thế {a/X}

   a. Biến đã kết buộc : xem (1)

3. **Biến X/ biến Y** đối sánh:

   a. Hai biến chưa kết buộc: luôn luôn đối sánh

      => Khi đó ta có phép thế {X/Y}

   a. Một biến kết buộc và một biến chưa kết buộc: xem (2)

   b. Hai biến kết buộc: xem (1)

4. **Biểu thức / biểu thức** đối sánh: chỉ khi các tên hàm hoặc vị từ, số ngôi giống nhau thì áp dụng đối sánh từng đối số một.

   VD: goo(X) - không đối sánh với foo(X) hay goo(X,Y)

   - đối sánh với goo(foo(Y)) với phép thế {foo(Y) / X}

Phạm vi của một biến

- Phạm vi của một biến là một câu.
- Một khi biến đã bị kết buộc, các phép hợp nhất theo sau và các suy luận kế tiếp phải giữ sự kết buộc này

VD:

man(X) => mortal(X)

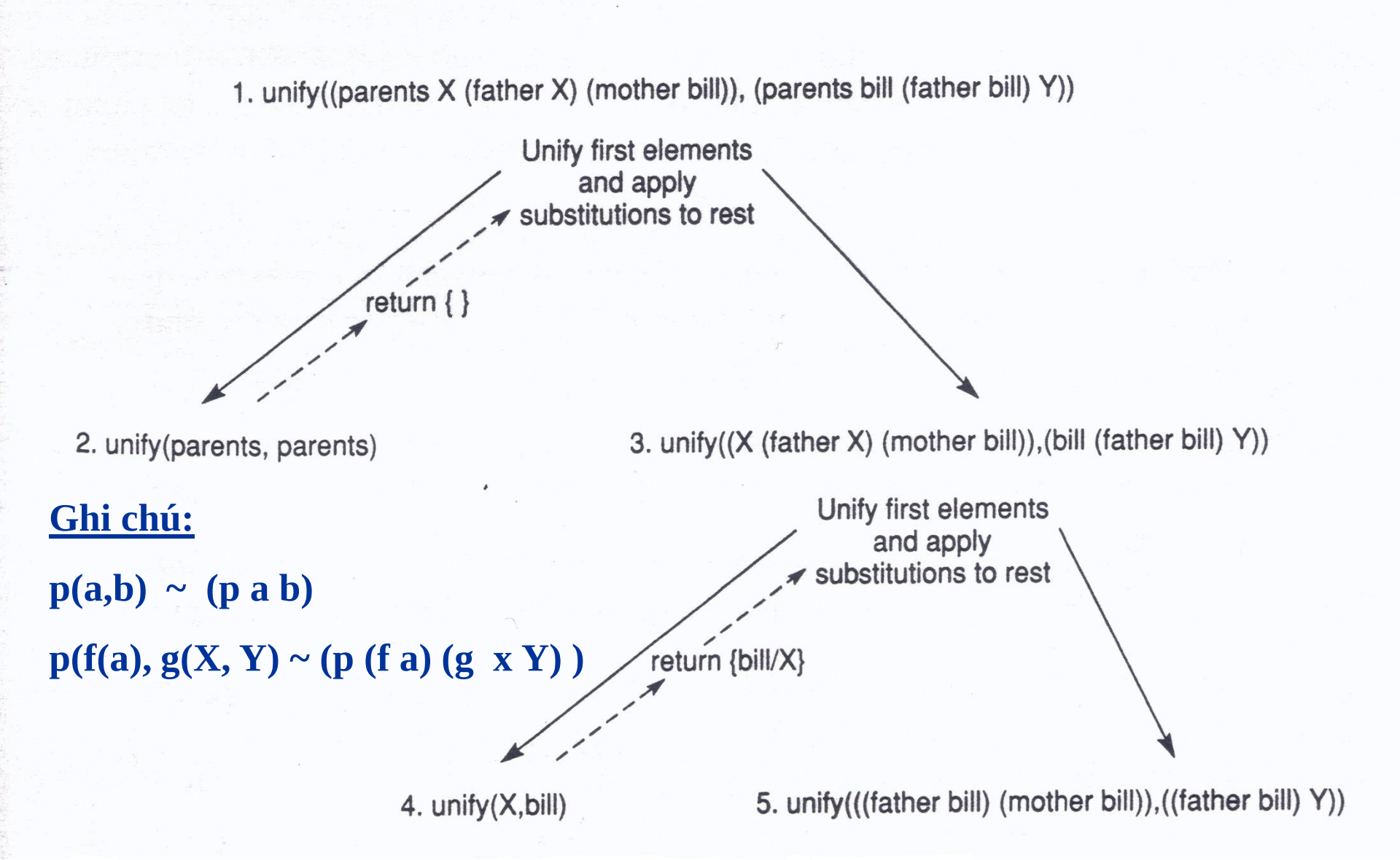
Nếu ta thế X bởi socrates thì ta được:

man(socrates) => mortal(socrates)

Ví dụ: Biểu thức đối sánh

- Hãy xác định xem foo(X,a,goo(Y)) có đối sánh với các biểu thức sau hay không? Nếu có thì cho biết phép thế tương ứng:
  - foo(X,b,foo(Y))
  - foo(fred, a, goo(Z))
  - foo(X,Y)
  - moo(X,a,goo(Y))
  - foo(Z,a,goo(moo(Z)))
  - foo(W,a,goo(jack))
- Cho biết kết quả có được khi hợp nhất p(a, X) với :
  - p(Y,Z) => q(Y,Z)
  - q(W,b) => r(W,b)

Thủ tục hợp nhất "Unify"



Ghi chú:

p(a,b) ~ (p a b)

p(f(a), g(X, Y) ~ (p (f a) (g x Y))

Tích các phép thế hợp nhất (Composition)

- Nếu S và S’ là hai tập hợp phép thế, thì tích của S và S’ được xác định bằng cách áp dụng S’ cho những phần tử của S và bổ sung kết quả này vào S.

  VD: {X/Y, W/X}, {V/X}, {a/V, f(b)/W}

  => {a/Y, f(b)/Z}

Hợp tử tổng quát nhất (Most General Unifier)

- Yêu cầu của giải thuật hợp nhất là hợp tử (unifier) càng tổng quát càng tốt: đó là hợp tử tổng quát nhất tìm thấy cho hai biểu thức.

  VD: Khi hợp nhất p(X) và p(Y):

  [redacted]*ông nên chọn* {fred/X, fred/Y} vì fred không phải là hợp tử tổng quát nhất

  [redacted]*ên chọn* {Z/Y, Z/Y}

Ứng Dụng: Hệ tư vấn tài chính (1)

Hệ tư vấn tài chính hoạt động theo các nguyên tắc sau:

- Các cá nhân không đủ tiền tiết kiệm nên tăng tiền tiết kiệm, bất kể thu nhập là bao nhiêu.
- Các cá nhân có đủ tiền tiết kiệm và đủ thu nhập nên xem xét việc đầu tư vào chứng khoán.
- Các cá nhân với thu nhập thấp nhưng đủ tiền tiết kiệm có thể chia phần thu nhập thêm vào tiết kiệm và chứng khoán.

Với:

- tiết kiệm đủ là 5000$/ người phụ thuộc
- Thu nhập đủ 15000$ + (4000$ / người phụ thuộc)

Ứng Dụng: Hệ tư vấn tài chính (2)

Xâu dựng hệ thống logic với các câu vị từ như sau:

1. savings_account(inadequate) [redacted]vestment(saving)
2. savings_account(adequate) [redacted]come(adequate) [redacted]vestment(stocks)
3. savings_account(adequate) [redacted]come(inadequate) [redacted]
   investment(combination)
4. [redacted] amount_saved(X) [redacted](dependents(Y) [redacted]
   greater(X,minsavings(Y))) [redacted]vings_account(adequate)
5. [redacted] amount_saved(X) [redacted](dependents(Y) [redacted]
   [redacted]reater(X,minsavings(Y))) [redacted]vings_account(inadequate)
6. [redacted] earning(X, steady) [redacted](dependents(Y) [redacted]
   greater(X,minincome(Y))) [redacted]come(adequate)
7. [redacted] earning(X, steady) [redacted](dependents(Y) [redacted]
   [redacted]reater(X,minincome(Y))) [redacted]come(inadequate)
8. [redacted] earning(X, unsteady) [redacted]come(inadequate)

With: minavings(X) = 5000 * X minincome(X)=15000+(4000*X)

Ứng Dụng: Hệ tư vấn tài chính(3)

Một nhà đầu tư với tình trạng như sau:

9. amount_saved(22000)
10. earnings(25000,steady)
11. dependents(3)

[redacted]vestment (?)

Dùng phép hợp nhất và luật Modus Ponens, suy ra:

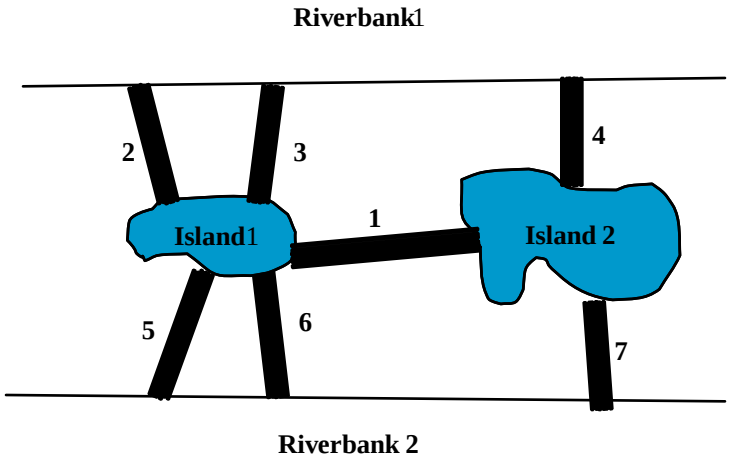
12. income(inadequate)
13. savings_account(adequate)

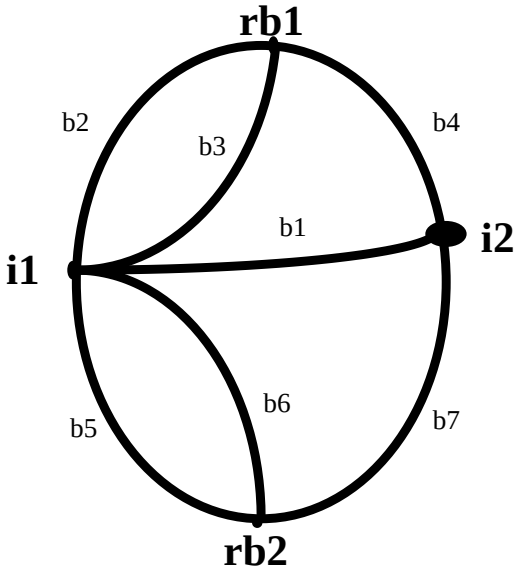
[redacted]vestment (combination)

Bài Tập Chương 2

Chương 3 - Cấu trúc và chiến lược cho TK - KGTT

- Khi biểu diễn một vấn đề như là một đồ thị không gian trạng thái, chúng ta có thể sử dụng lý thuyết đồ thị để phân tích cấu trúc và độ phức tạp của các vấn đề cũng như các thủ tục tìm kiếm.





Hệ thống cầu thành phố Konigsberg và biểu diễn đồ thị tương ứng

Nội dung chương 3

- Định nghĩa Không Gian Trạng Thái
- Các chiến lược tìm kiếm trên không gian trạng thái:
  - TK ***hướng từ dữ liệu*** (data – driven)
  - TK ***hướng từ mục tiêu*** (goal – driven).
- Tìm kiếm trên không gian trạng thái:
  - ***TK rộng*** (breath – first search)
  - ***TK sâu*** (depth – first search)
  - ***TK sâu bằng cách đào sâu nhiều lần*** (depth – first search with iterative deepening)
- Sử dụng không gian trạng thái để biểu diễn suy luận với phép tính vị từ: ***Đồ thị Và/Hoặc*** (And/Or Graph)

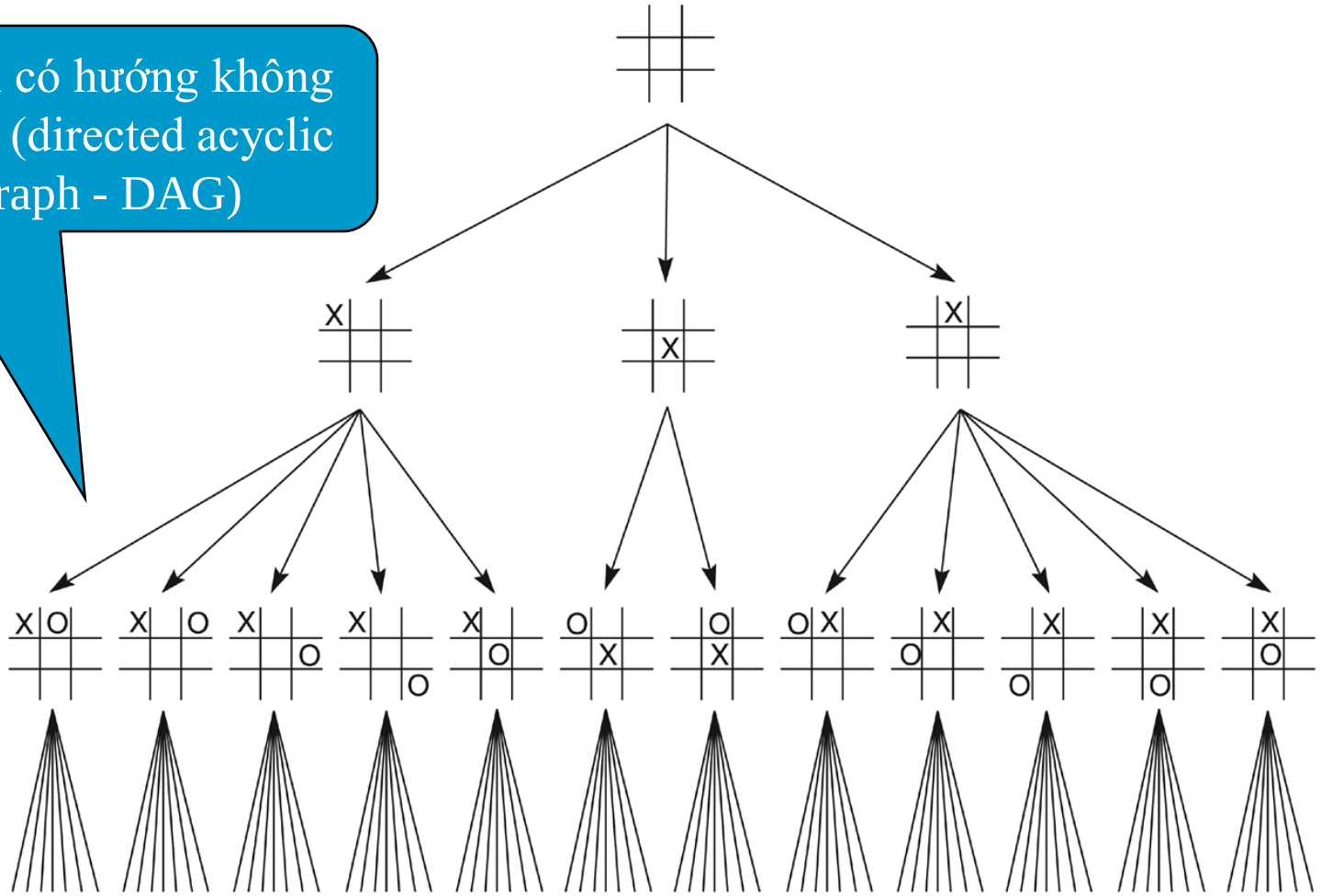
ĐN: KHÔNG GIAN TRẠNG THÁI

Một **KGTT** (state space) là 1 bộ [N, A, S, GD] trong đó:

- **N** (node) là các *nút* hay các *trạng thái* của đồ thị.
- **A** (arc) là tập các *cung* (hay các *liên kết*) giữa các nút.
- **S** (solution) là một tập chứa các *trạng thái ban đầu* của bài toán
- **GD** (Goal Description) là một tập chứa các *trạng thái đích* của bài toán và được mô tả theo một trong hai đặc tính:
  - Đặc tính có thể đo lường được các trạng thái gặp trong quá trình tìm kiếm. VD: Tic-tac-toe, 8-puzzle,…
  - Đặc tính của đường đi được hình thành trong quá trình tìm kiếm. VD: TSP
- ***Đường đi của lời giải*** (solution path) là một con đường đi qua đồ thị này từ một nút thuộc S đến một nút thuộc GD.

Một phần KGTT triển khai trong Tic-tac-toe

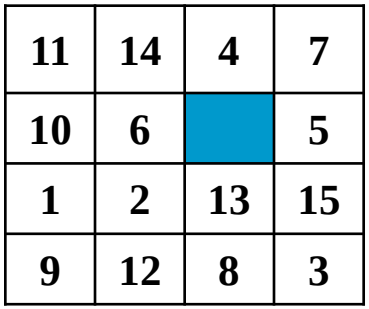
Đồ thị có hướng không lặp lại (directed acyclic graph - DAG)



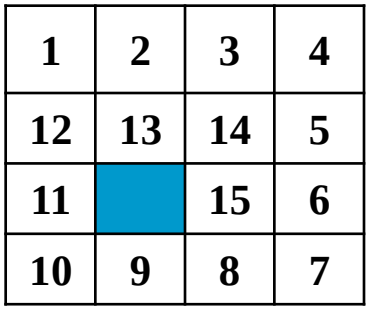
Trò đố 8 ô hay 15 ô

Trạng thái ban đầu Trạng thái đích

- Trò đố 15 ô

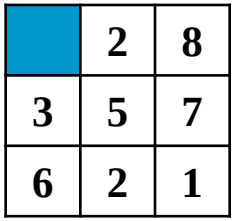


11	14	4	7
10	6		5
1	2	13	15
9	12	8	3

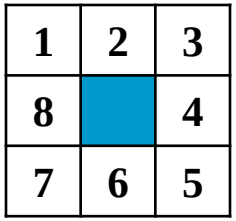


1	2	3	4
12	13	14	5
11		15	6
10	9	8	7

- Trò đố 8 ô



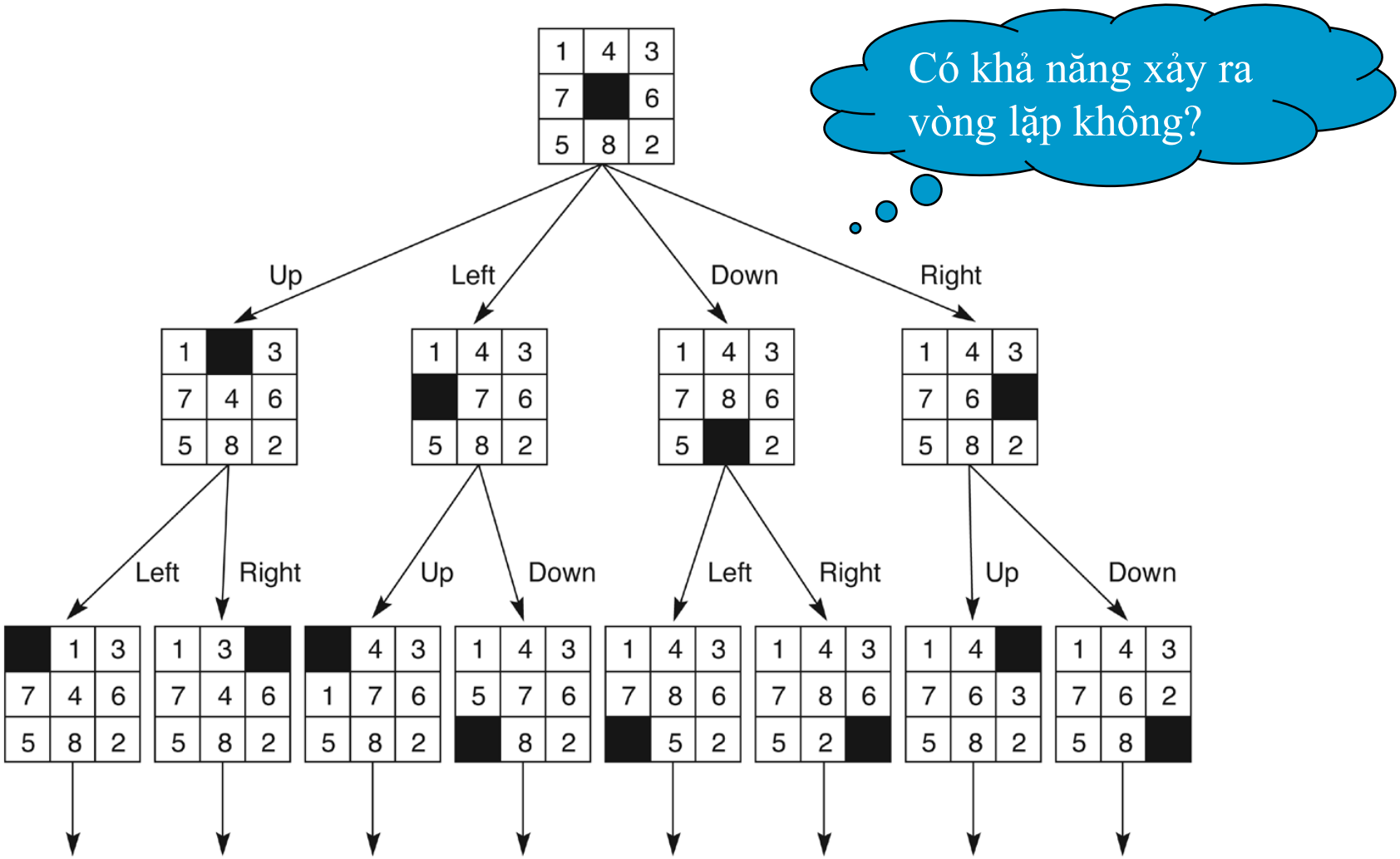
	2	8
3	5	7
6	2	1



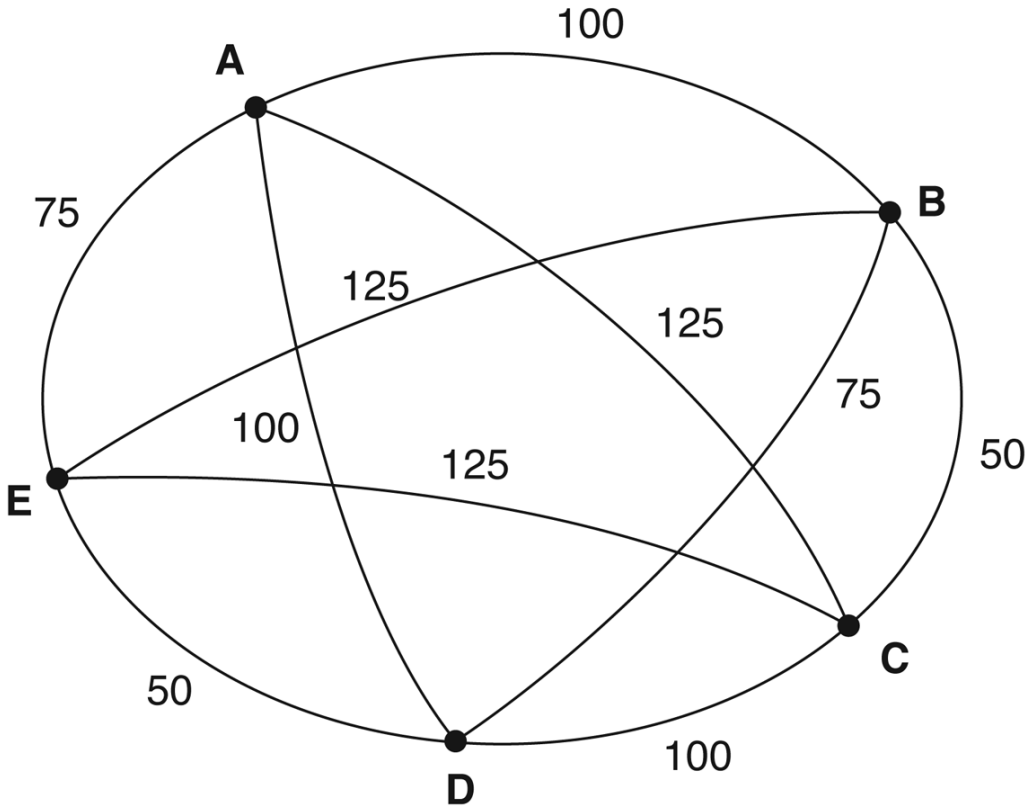
1	2	3
8		4
7	6	5

- Cần biểu diễn KGTT cho bài toán này như thế nào?

KGTT của 8-puzzle sinh ra bằng phép "di chuyển ô trống"

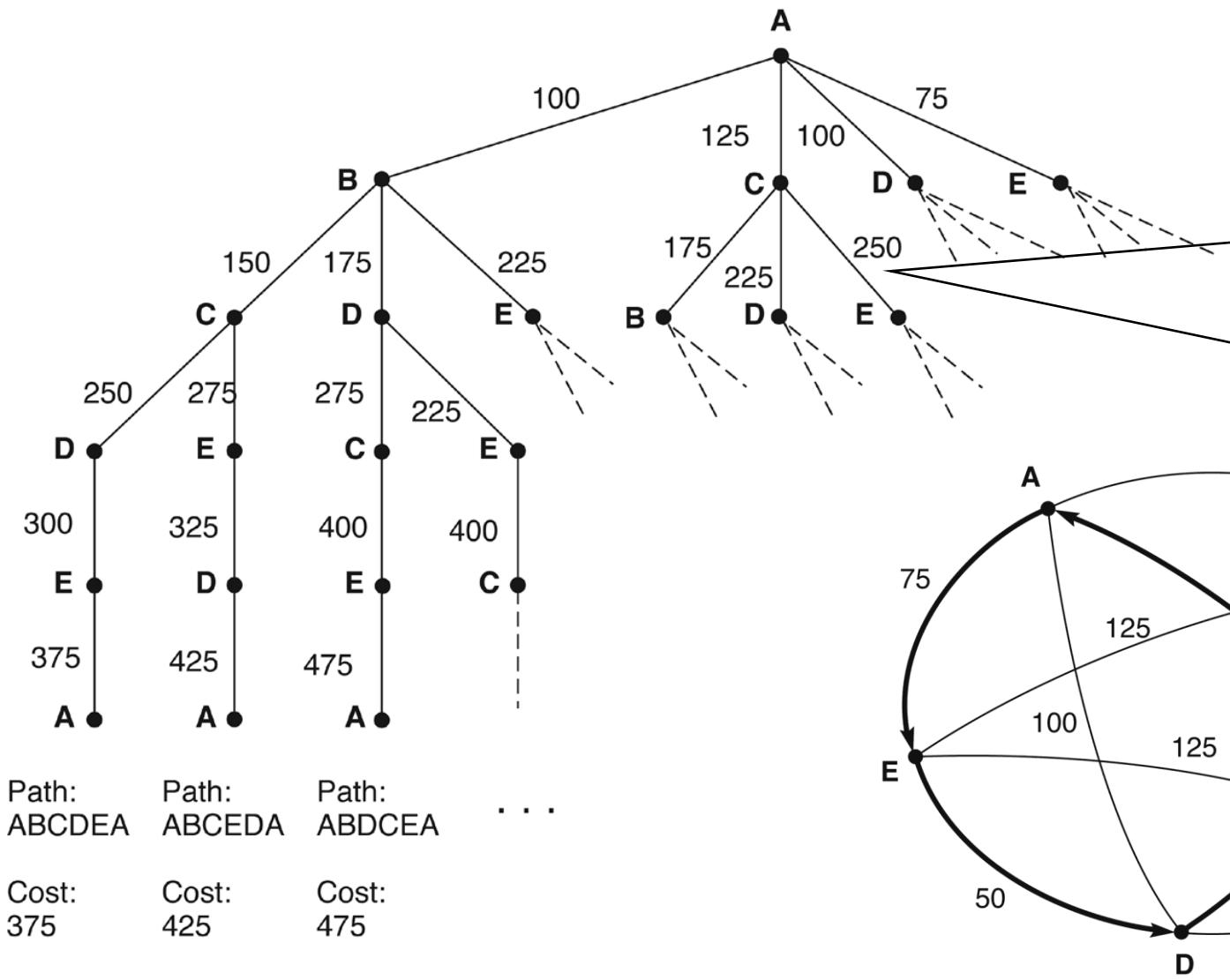


Một ví dụ của bài toán TSP



- Cần biểu diễn KGTT cho bài toán này như thế nào?

KGTT của bài toán TSP



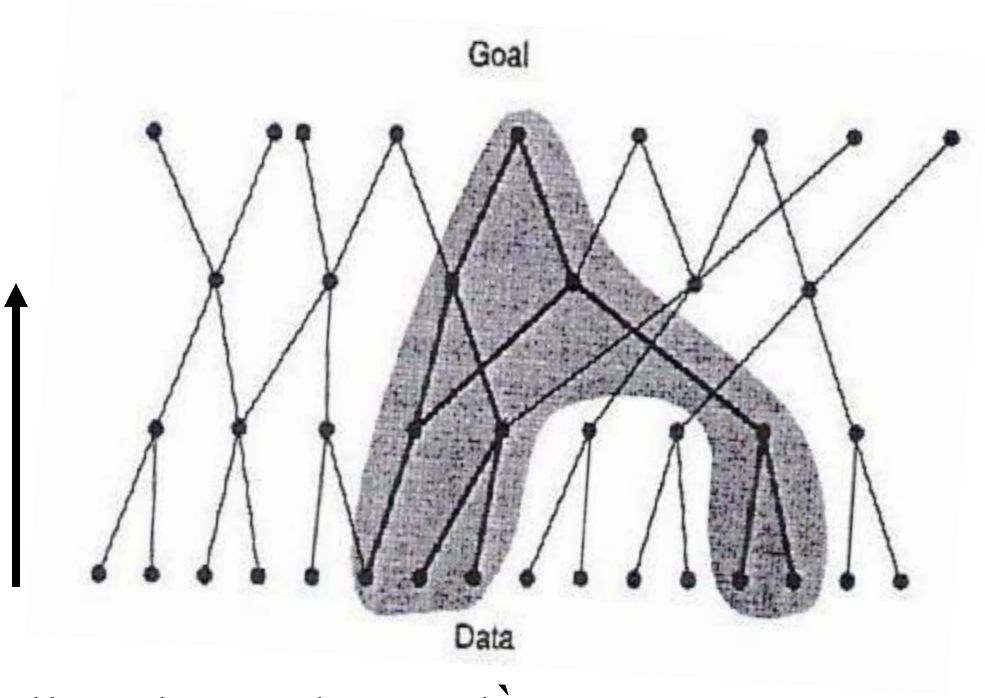
Mỗi cung được đánh dấu bằng tổng giá của con đường từ nút bắt đầu đến nút hiện tại.

Các Chiến Lược cho TK-KGTT

- **TK hướng từ dữ liệu (Data-driven Search)**
  - Suy diễn tiến (forward chaining)
- **TK hướng từ mục tiêu (Goal-driven Search)**
  - Suy diễn lùi (backward chaining)

TK Hướng từ Dữ Liệu

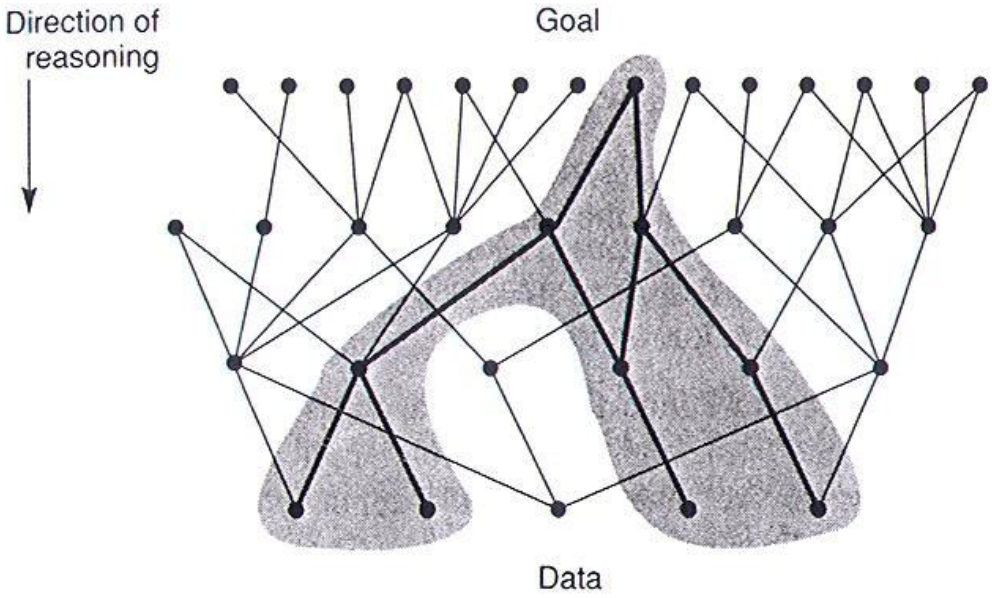
- Việc tìm kiếm đi từ dữ liệu đến mục tiêu



- Thích hợp khi:
  - Tất cả hoặc một phần dữ liệu được cho từ đầu.
  - Có nhiều mục tiêu, nhưng chỉ có một số ít các phép toán có thể áp dụng cho một trạng thái bài toán.
  - Rất khó đưa ra một mục tiêu hoặc giả thuyết ngay lúc đầu.

TK Hướng Từ Mục Tiêu

- Việc tìm kiếm đi từ mục tiêu trở về dữ liệu.



- Thích hợp khi:
  - Có thể đưa ra mục tiêu hoặc giả thuyết ngay lúc đầu.
  - Có nhiều phép toán có thể áp dụng trên 1 trạng thái của bài toán => sự bùng nổ số lượng các trạng thái.
  - Các dữ liệu của bài toán không được cho trước, nhưng hệ thống phải đạt được trong quá trình tìm kiếm.

Các phương pháp tìm kiếm trên đồ thị KGTT:

Phát triển từ giải thuật quay lui **(back – tracking)**

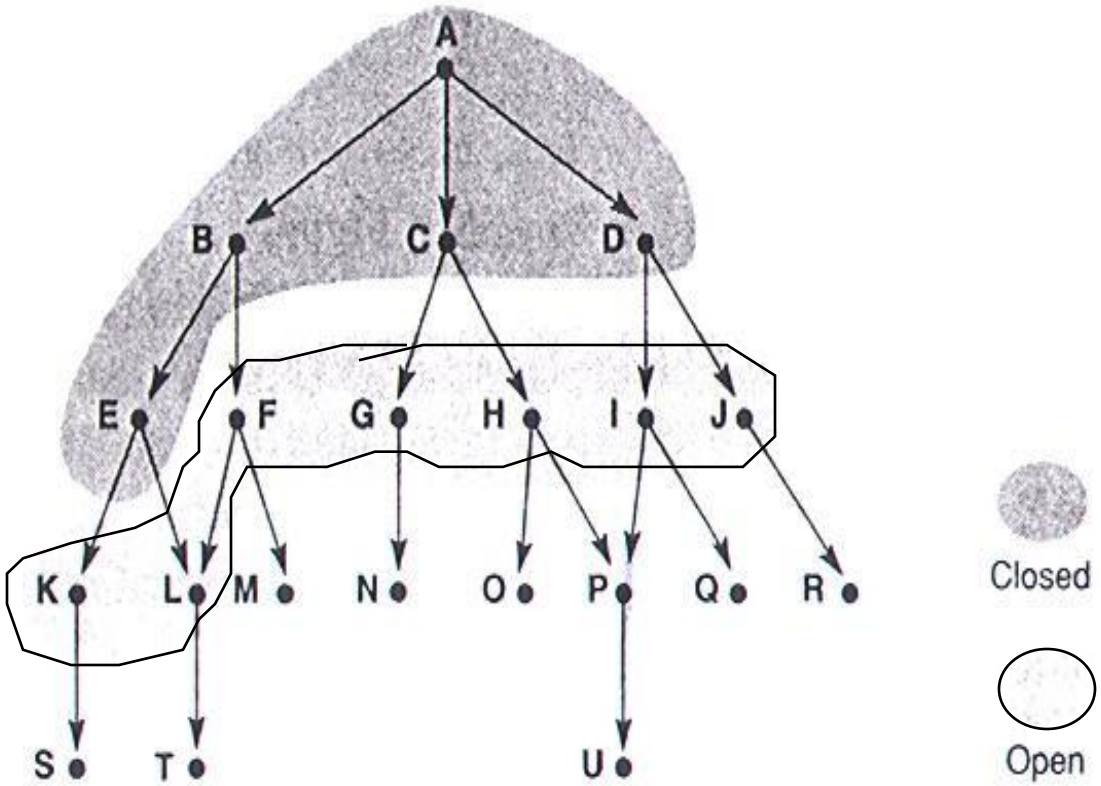
- ***Tìm kiếm rộng*** (breath-first search)
- ***Tìm kiếm sâu*** (depth-first search)
- ***TK sâu bằng cách đào sâu nhiều lần*** (depth-first search with iterative deepening)

Tìm Kiếm Rộng

1. Open = [A]; closed = []
2. Open = [B,C,D];
   closed = [A]
2. Open = [C,D,E,F];
   closed = [B,A]
3. Open = [D,E,F,G,H];
   closed = [C,B,A]
4. Open = [E,F,G,H,I,J];
   closed = [D,C,B,A]
5. Open = [F,G,H,I,J,K,L];
   closed = [E,D,C,B,A]
6. Open = [G,H,I,J,K,L,M];
   (vì L đã có trong open);

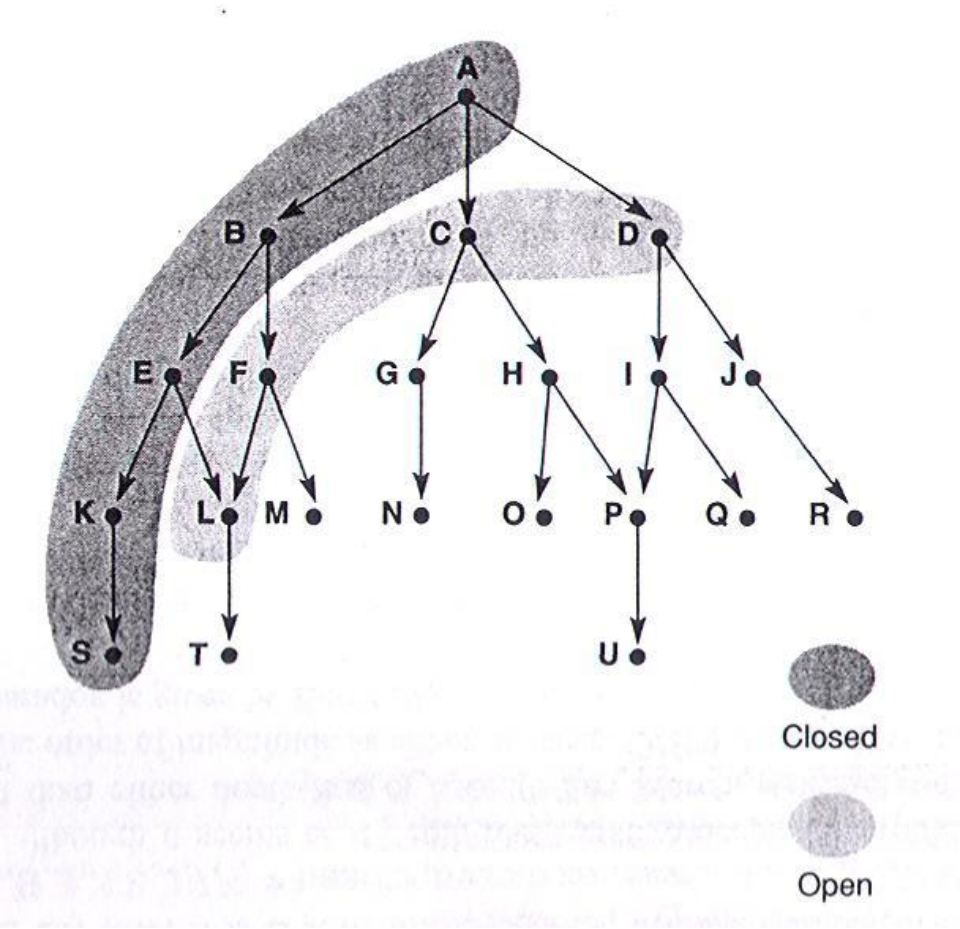
closed = [F,E,D,C,B,A]

…



Tìm kiếm Sâu

1. Open = [A]; closed = []
2. Open = [B,C,D]; closed = [A]
3. Open = [E,F,C,D];closed = [B,A]
4. Open = [K,L,F,C,D];
   closed = [E,B,A]
5. Open = [S,L,F,C,D];
   closed = [K,E,B,A]
6. Open = [L,F,C,D];
   closed = [S,K,E,B,A]
7. Open = [T,F,C,D];
   closed = [L,S,K,E,B,A]
8. Open = [F,C,D];
   closed = [T,L,S,K,E,B,A]
9. …



Tìm Kiếm Sâu hay Rộng? (1)

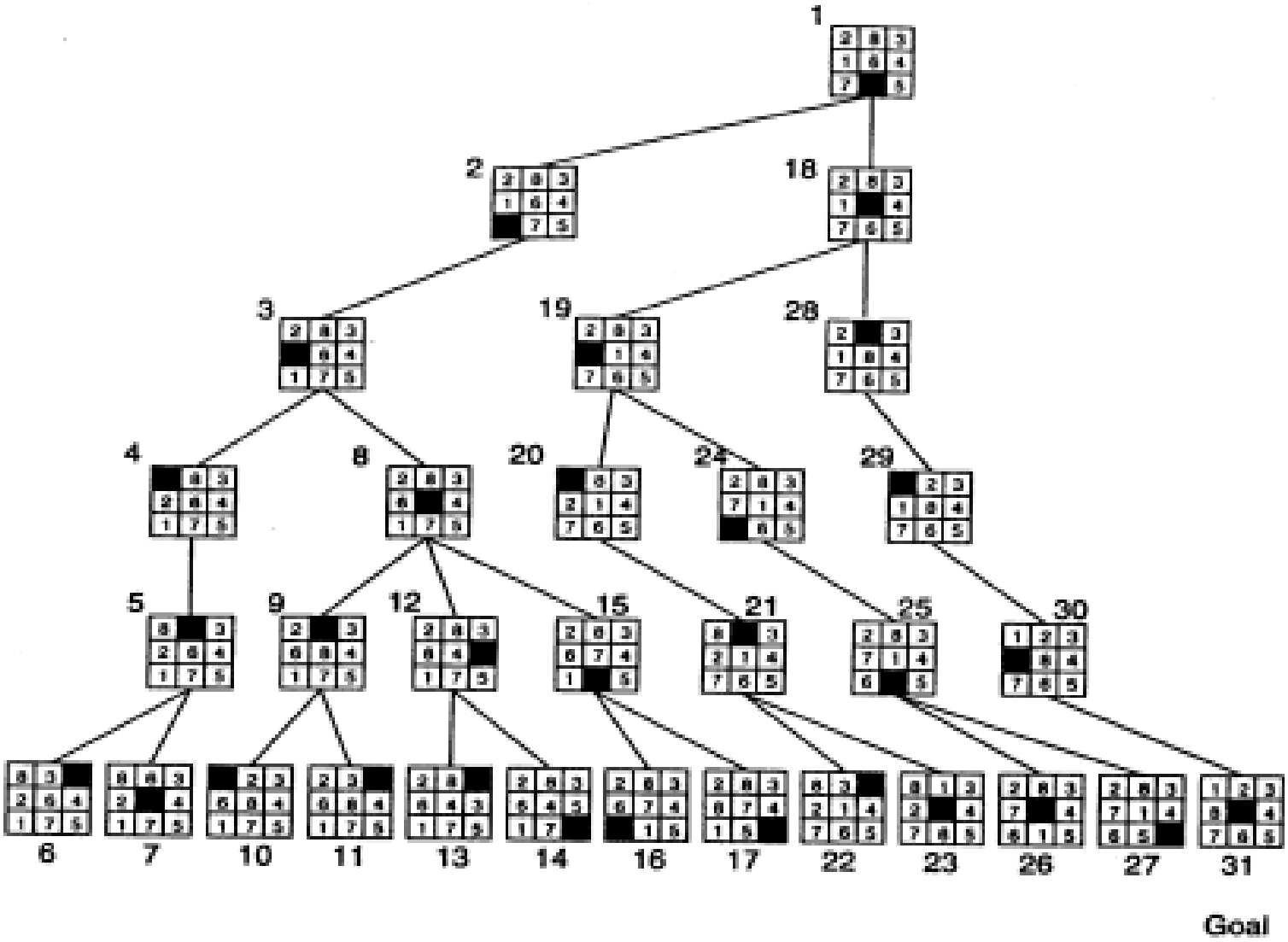
- Có cần thiết tìm một *đường đi ngắn nhất* đến mục tiêu hay không?
- Sự *phân nhánh* của không gian trạng thái
- Tài nguyên về *không gian* và *thời gian* sẵn có
- *Khoảng cách trung bình* của đường dẫn đến trạng thái mục tiêu.
- Yêu cầu đưa ra *tất cả các lời giải* hay chỉ là lời giải tìm được đầu tiên.

Tìm kiếm sâu bằng cách đào sâu nhiều lần (depth-first iterative deepening)

- Độ sâu giới hạn (depth bound): giải thuật TK sâu sẽ quay lui khi trạng thái đang xét đạt đến độ sâu giới hạn đã định.
- TK Sâu bằng cách đào sâu nhiều lần: TK sâu với độ sâu giới hạn là 1, nếu thất bại, nó sẽ lặp lại GT TK sâu với độ sâu là 2,… GT tiếp tục cho đến khi tìm được mục tiêu, mỗi lần lặp lại tăng độ sâu lên 1.
- GT này có độ phức tạp về thời gian cùng bậc với TK Rộng và TK Sâu.

Trò chơi ô đố 8-puzzle

The 8-puzzle searched by a production system with loop detection and depth bound 5



Đồ thị Và/Hoặc

- Sử dụng KGTT để biểu diễn suy luận với phép tính vị từ
- Là phương pháp *qui bài toán về các bài toán con*.
- Một tập hợp các mệnh đề / câu vị từ tạo thành một đồ thị Và/Hoặc (And/Or graph) hay siêu đồ thị (hypergraph).
- Trong đồ thị Và/Hoặc:
  - Các nút AND biểu thị sự phân chia bài toán, tất cả các bài toán con phải được chứng minh là đúng.
  - Các nút OR biểu thị các chiến lược giải quyết bài toán khác nhau, chỉ cần chứng minh một chiến lược đúng là đủ
- Có thể áp dụng TK theo kiểu hướng từ dữ liệu hay từ mục tiêu.
- Trong giải thuật cần ghi nhận diễn tiến của quá trình.

Ví dụ Đồ thị Và/Hoặc

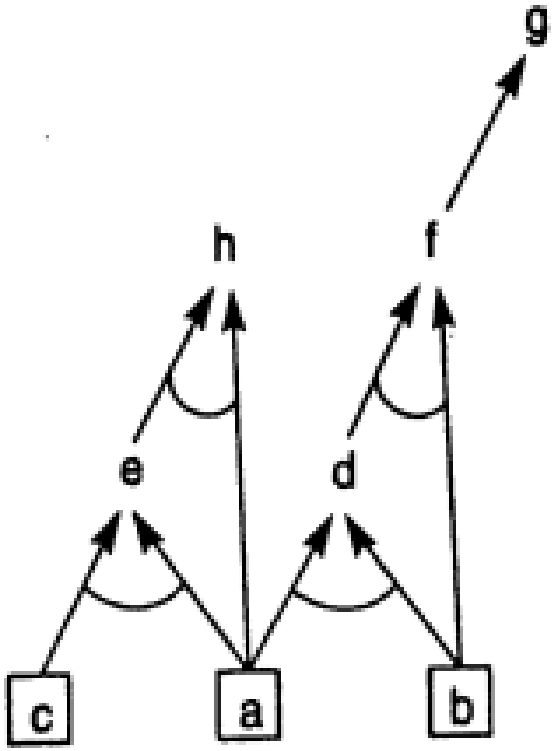
- Giả sử một tình huống với các mệnh đề sau:

a b c

a [redacted] a [redacted]

b [redacted] f [redacted]

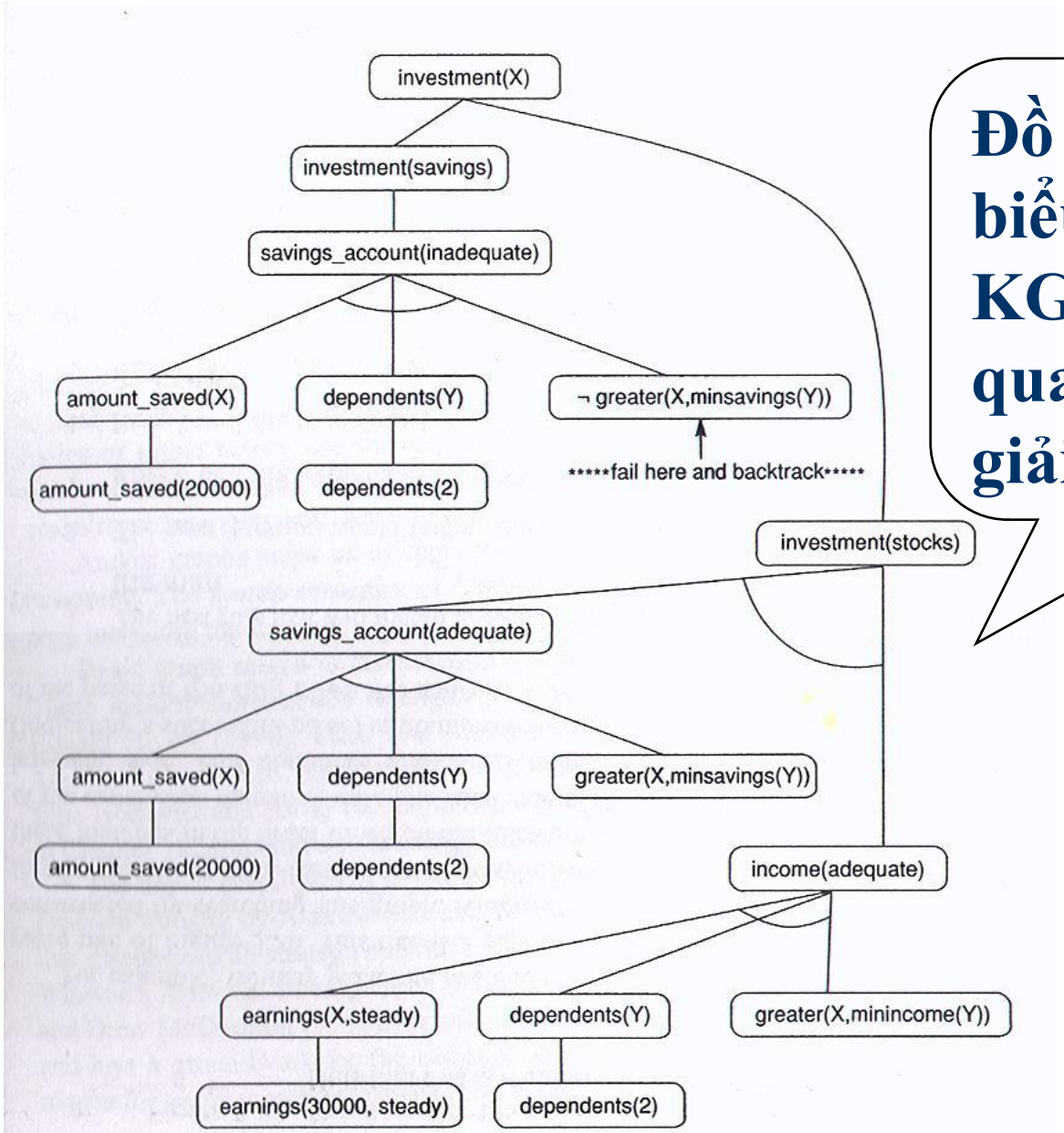
a [redacted]



Hãy trả lời các câu hỏi sau:

1. h có đúng không?
2. h có cón đúng nêu b sai?

Ví dụ: Hệ Tư Vấn Tài Chính



Đồ Thị And/Or biểu diễn phần KGTT đã duyệt qua để đi đến lời giải

1. Fred is a collie.
   collie(fred).
2. Sam is Fred's master.
   master(fred,sam).
3. The day is Saturday.
   day(saturday).
4. It is cold on Saturday.
   ¬ (warm(saturday)).
5. Fred is trained.
   trained(fred).
6. Spaniels are good dogs and so are trained collies.
   ∀ X[spaniel(X) ∨ (collie(X) ∧ trained(X)) → gooddog(X)]
7. If a dog is a good dog and has a master then he will be with his master.
   ∀ (X,Y,Z) [gooddog(X) ∧ master(X,Y) ∧ location(Y,Z) → location(X,Z)]
8. If it is Saturday and warm, then Sam is at the park.
   (day(saturday) ∧ warm(saturday)) → location(sam,park).
9. If it is Saturday and not warm, then Sam is at the museum.
   (day(saturday) ∧ ¬ (warm(saturday))) → location(sam,museum).

<u>VÍ DỤ ĐỒ THỊ AND/OR</u>:

Cho một bài toán được mô tả bằng các câu vị từ:

⇨Hãy vẽ đồ thị AND/OR biểu diễn phần KGTK để trả lời câu hỏi: "Fred đang ở đâu?" (Áp dụng suy diễn lùi)

Bài Tập Chương 3

Chương 4 – Tìm kiếm heuristic

- **Heuristics**: là các phỏng đoán, ước chừng dựa trên kinh nghiệm, trực giác.
- Các hệ giải quyết AI **sử dụng heuristic trong hai tình huống cơ bản**:
  - Bài toán được định nghĩa chính xác nhưng *chi phí tìm lời giải bằng TK vét cạn là không thể chấp nhận*.

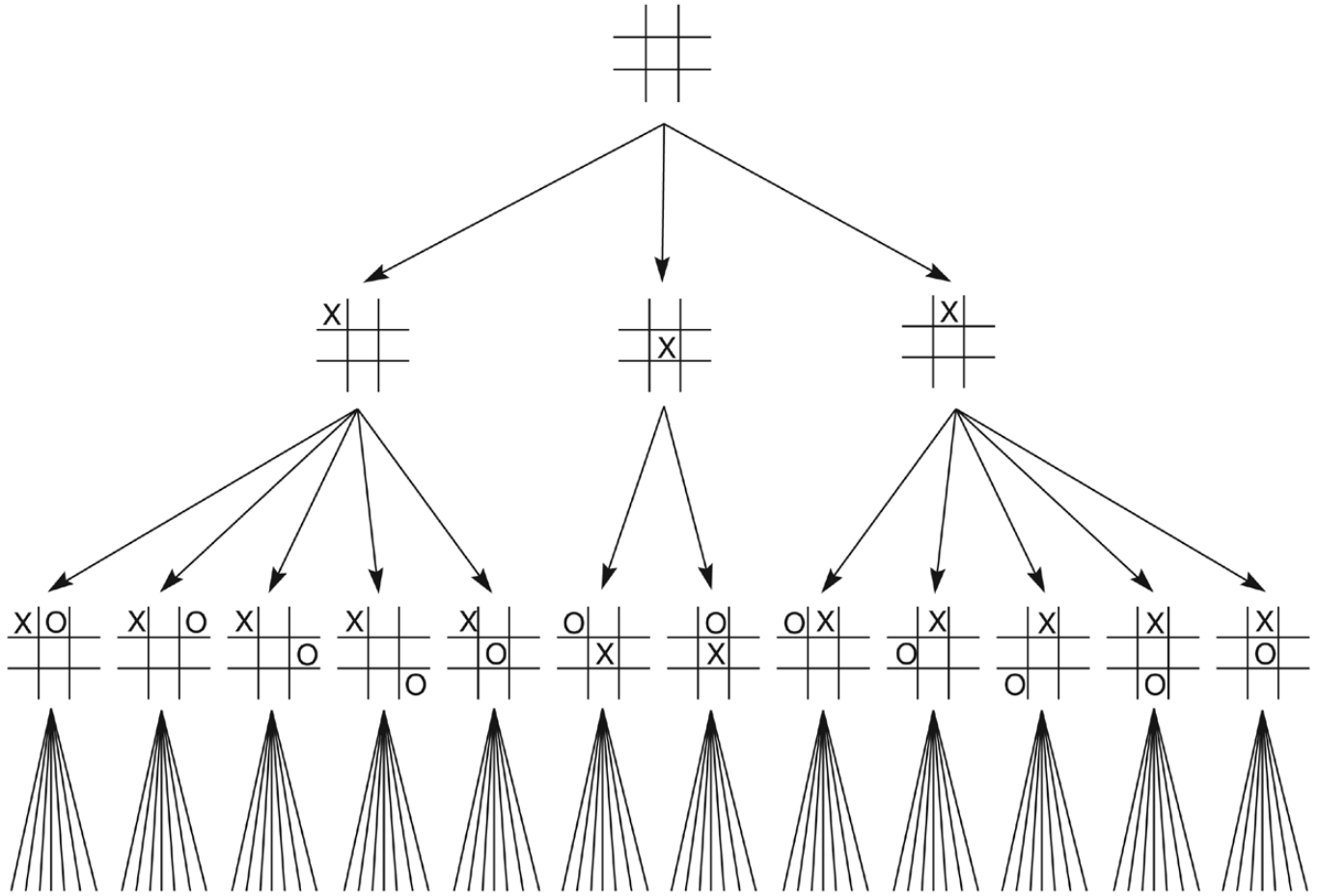
    VD: Sự bùng nổ KGTT trong trò chơi cờ vua.
  - *Vấn đề với nhiều sự mơ hồ* trong lời phát biểu bài toán hay dữ liệu cũng như tri thức sẵn có.

    VD: Chẩn đoán trong y học.

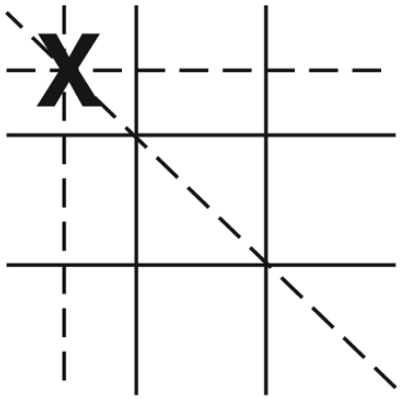
Giải Thuật Heuristic

- Một giải thuật heuristic có thể được xem gồm 2 phần:
  - Phép đo heuristic: thể hiện qua *hàm đánh giá heuristic (evaluation function)*, dùng để đánh giá các đặc điểm của một trạng thái trong KGTT.
  - Giải thuật tìm kiếm heuristic:
    - *Giải thuật leo núi (hill-climbing)*
    - *TK tốt nhất (best-first search)*

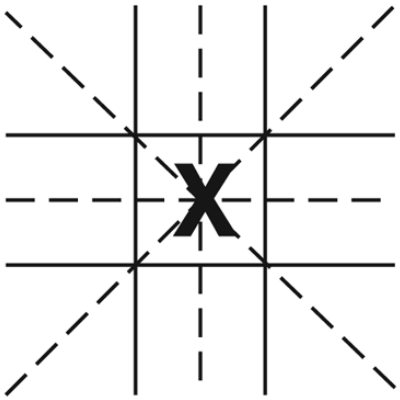
KGTT của tic-tac-toe được thu nhỏ nhờ tính đối xứng của các trạng thái.



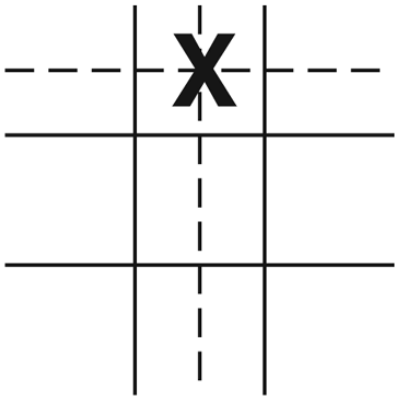
Phép đo heuristic (2)



Three wins through
a corner square



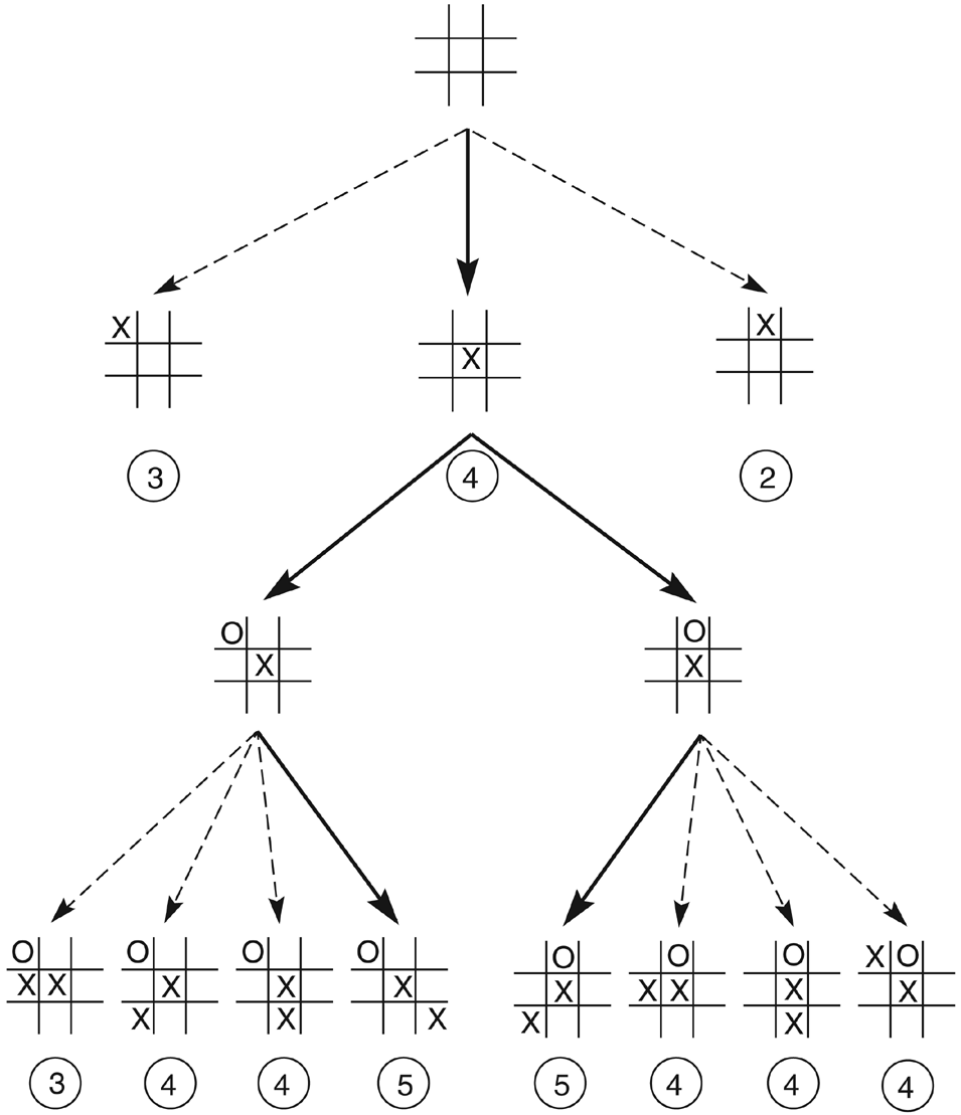
Four wins through
the center square



Two wins through
a side square

Heuristic "*Số đường thắng nhiều nhất*" áp dụng cho các nút con đầu tien trong tic-tac-toe.

KGTT càng thu nhỏ khi áp dụng heuristic

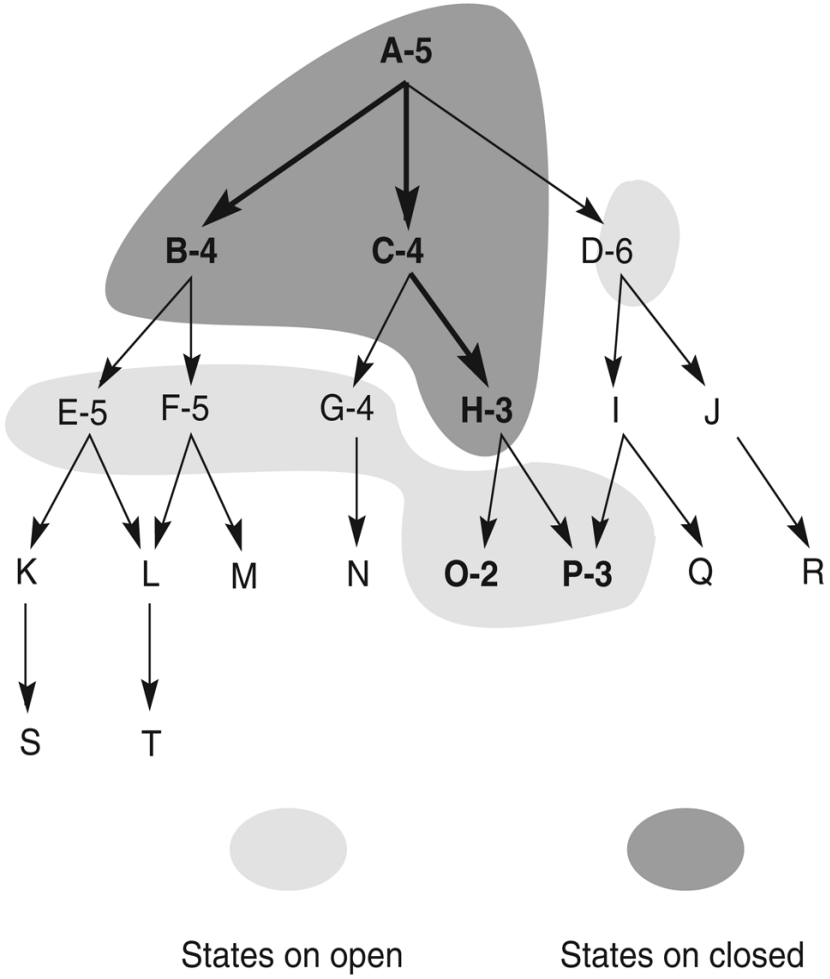


Giải thuật Leo Núi

- Giải thuật:
  - Mở rộng trạng thái hiện tại và đánh giá các trạng thái con của nó bằng hàm đánh giá heuristic.
  - Con "tốt nhất" sẽ được chọn để đi tiếp.
- Giới hạn:
  - Giải thuật có khuynh hướng bị sa lầy ở những cực đại cục bộ:
    - Lời giải tìm được không tối ưu
    - Không tìm được lời giải mặc dù có tồn tại lời giải
  - Giải thuật có thể gặp vòng lặp vô hạn do không lưu giữ thông tin về các trạng thái đã duyệt.

Giải thuật TK Tốt Nhất

1. open = [A5]; closed = []
2. Đánh giá A5; open = [B4,C4,D6]; closed = [A5]
3. Đánh giá B4;
   open = [C4,E5,F5,D6];
   closed = [B4,A5]
4. Đánh giá C4;
   open = [H3,G4,E5,F5,D6];
   closed = [C4,B4,A5]
5. Đánh giá H3;
   open = [O2,P3,G4,E5,F5,D6];
   closed = [H3,C4,B4,A5]
6. Đánh giá O2;
   open = [P3,G4,E5,F5,D6];
   closed = [O2,H3,C4,B4,A5]
7. Đánh giá P3; tìm được lời giải!



Cài Đặt Hàm Đánh Giá (Evaluation Function)

Xét trò chơi 8-puzzle. Cho mỗi trạng thái n một giá trị f(n):

$$f(n) = g(n) + h(n)$$

g(n) = khoảng cách thực sự từ n đến trạng thái bắt đầu

h(n) = hàm heuristic đánh giá khoảng cách từ trạng thái n đến mục tiêu.

1	2	3
8		4
7	6	5

goal

h(n): số lượng các vị trí còn sai

start

g(n) = 0

2	8	3
1	6	4
7		5

g(n) = 1

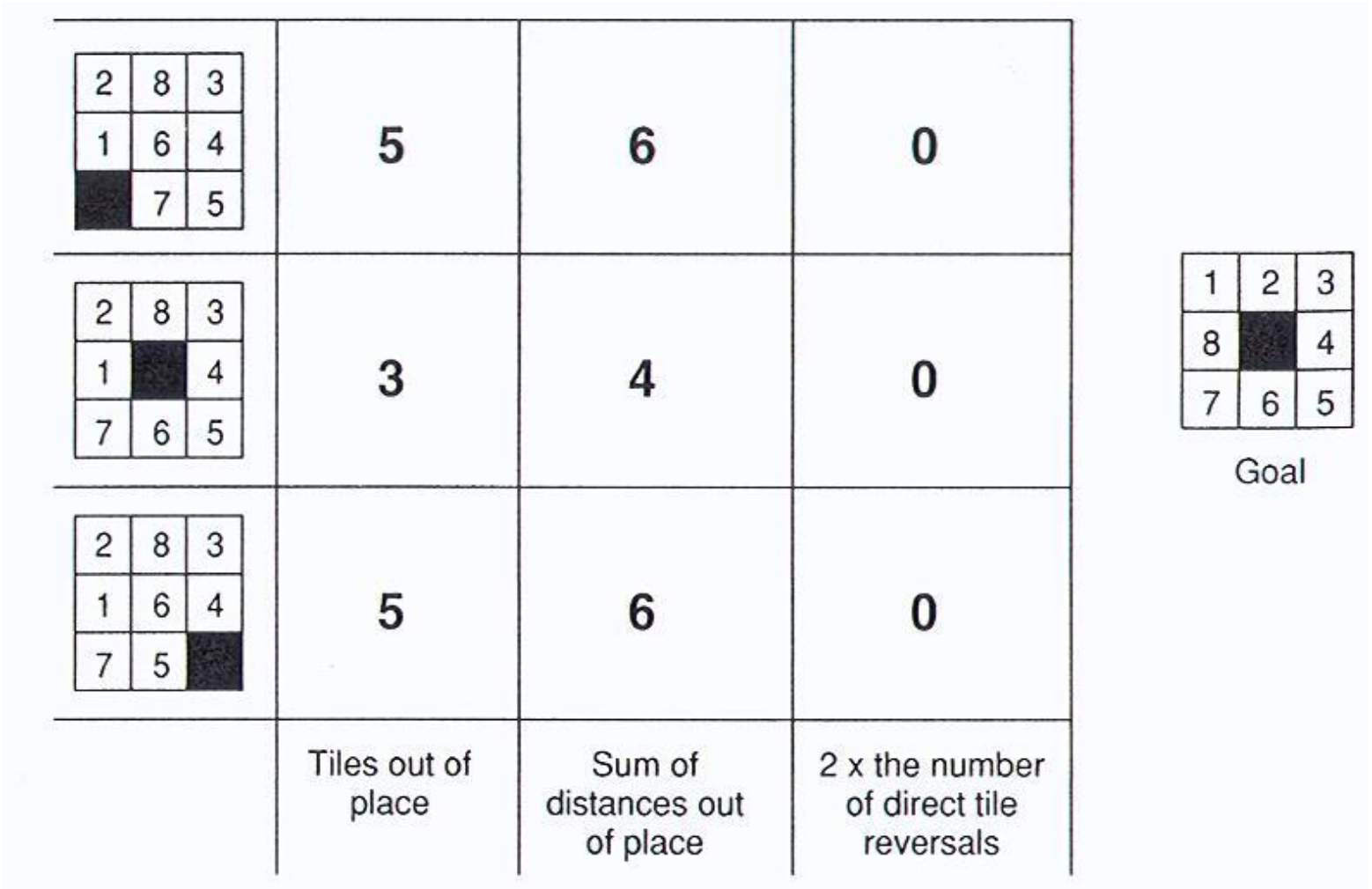
2	8	3
1	6	4
	7	5

2	8	3
1		4
7	6	5

2	8	3
1	6	4
7	5	

f(n) = **6** **4** **6**

Khó khăn trong thiết kế hàm heuristic



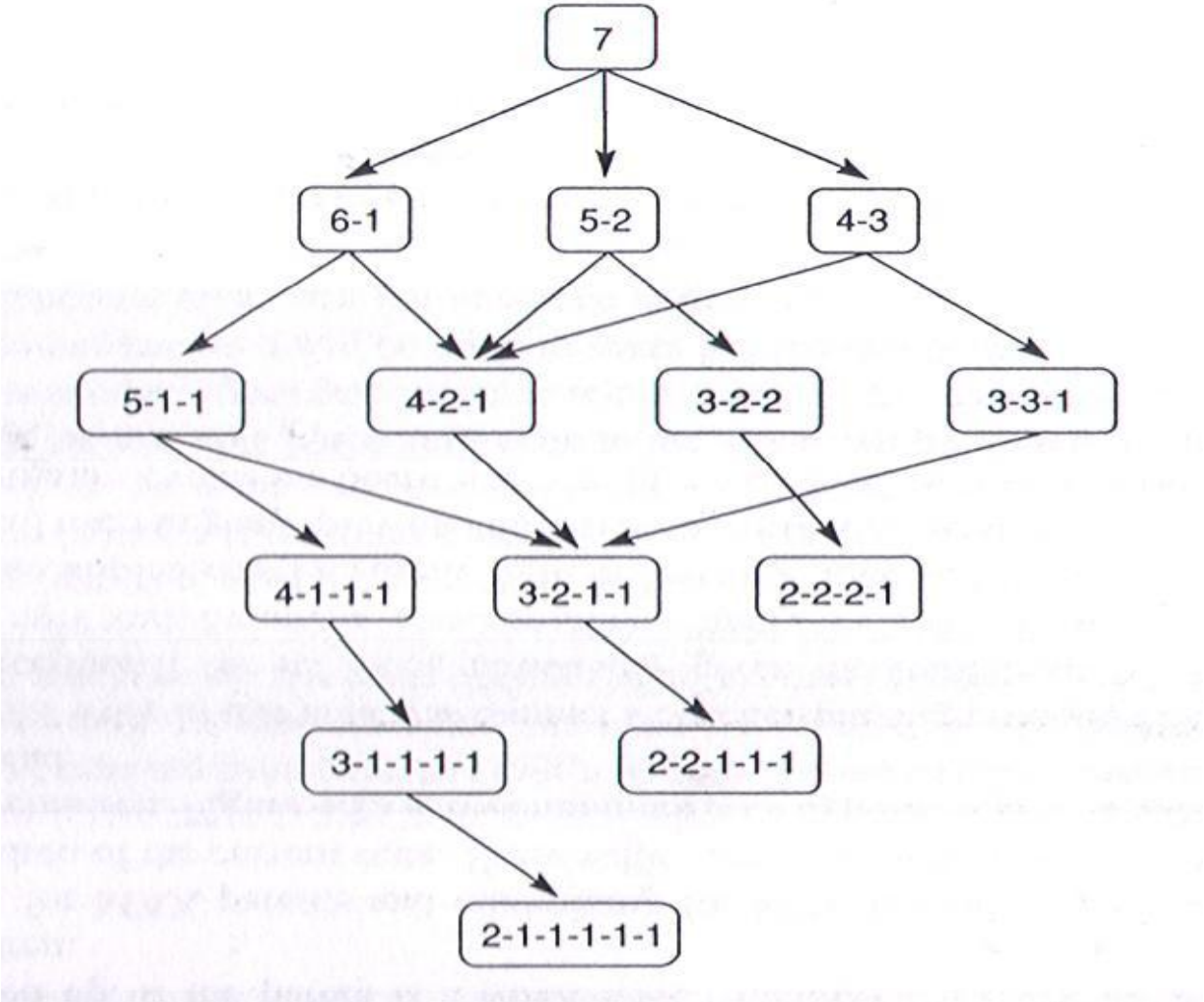
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals
2 8 3 / 1 6 4 / ■ 7 5	5	6	0
2 8 3 / 1 ■ 4 / 7 6 5	3	4	0
2 8 3 / 1 6 4 / 7 5 ■	5	6	0

Ba heuristic áp dụng vào 3 trạng thái của trò chơi ô đố 8 số

Heuristic trong trò chơi đối kháng

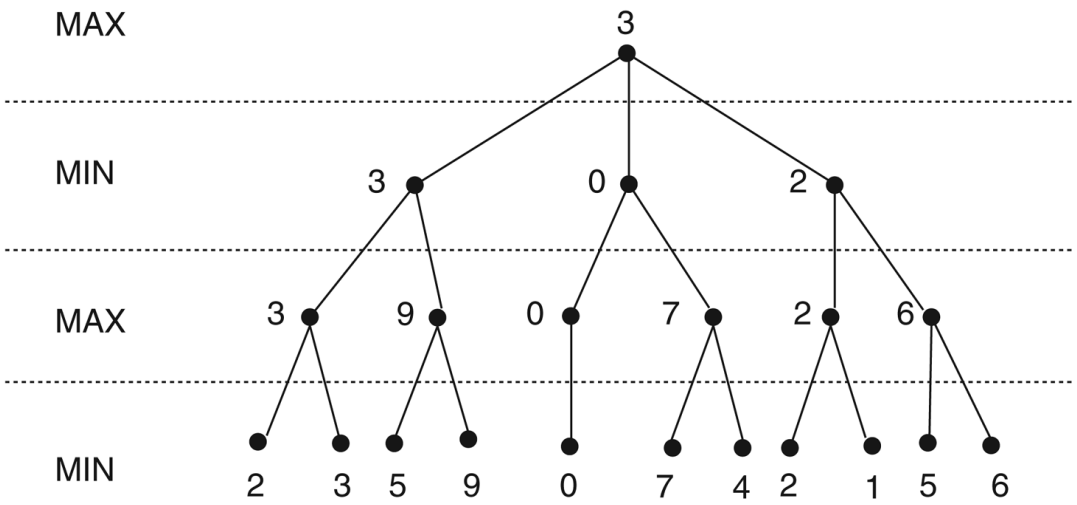
- **Giải thuật minimax**:
  - Hai đấu thủ trong trò chơi được gọi là **MIN** và **MAX**.
  - Mỗi nút lá có giá trị:
    - **1** nếu là MAX thắng,
    - **0** nếu là MIN thắng.
  - Minimax sẽ truyền các giá trị này lên cao dần trên đồ thị, qua các nút cha mẹ kế tiếp theo các luật sau:
    - Nếu trạng thái cha mẹ là **MAX**, gán cho nó giá trị **lớn nhất** có trong các trạng thái con.
    - Nếu trạng thái bố, mẹ là **MIN**, gán cho nó giá trị **nhỏ nhất** có trong các trạng thái con.

Hãy áp dụng GT Minimax vào Trò Chơi NIM



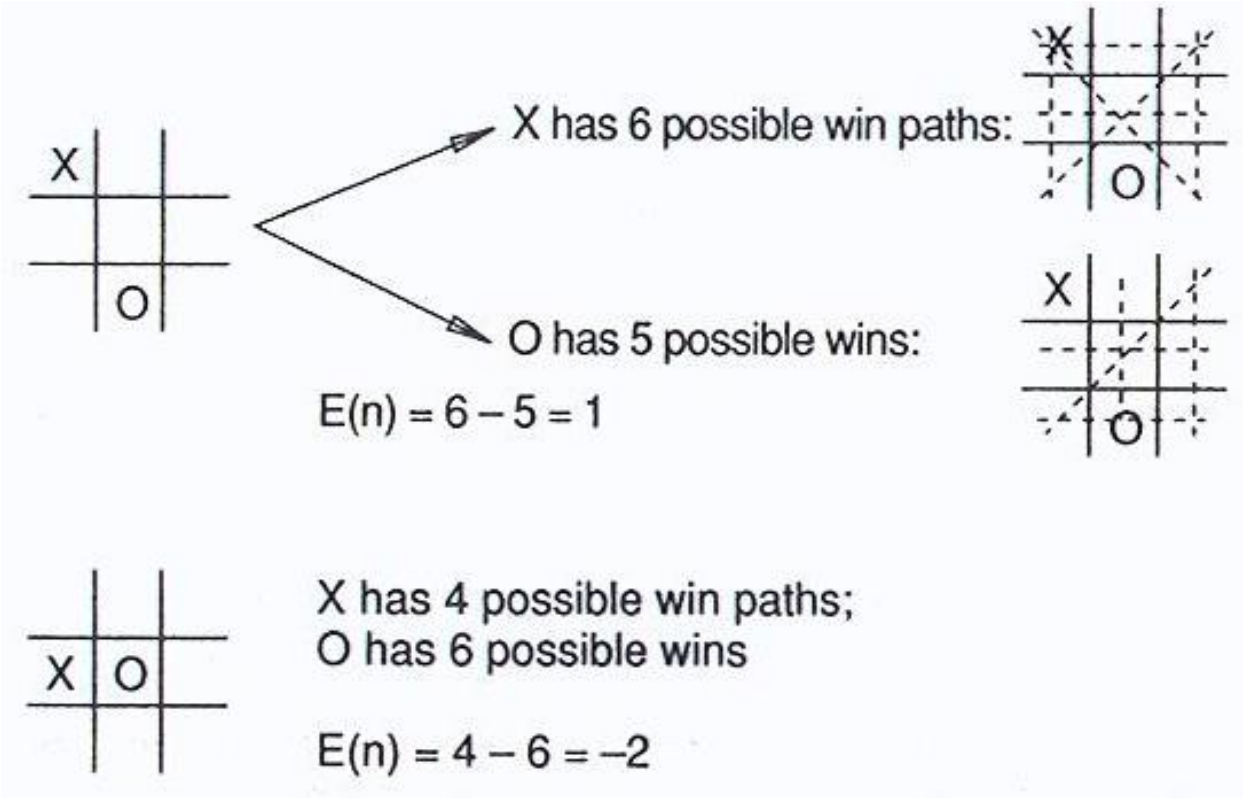
Minimax với độ sâu lớp cố định

- Minimax đối với một KGTT giả định.



- Các nút lá được gán các giá trị ***heuristic***
- Còn giá trị tại các nút trong là các giá trị nhận được dựa trên giải thuật Minimax

Heuristic trong trò chơi tic-tac-toe



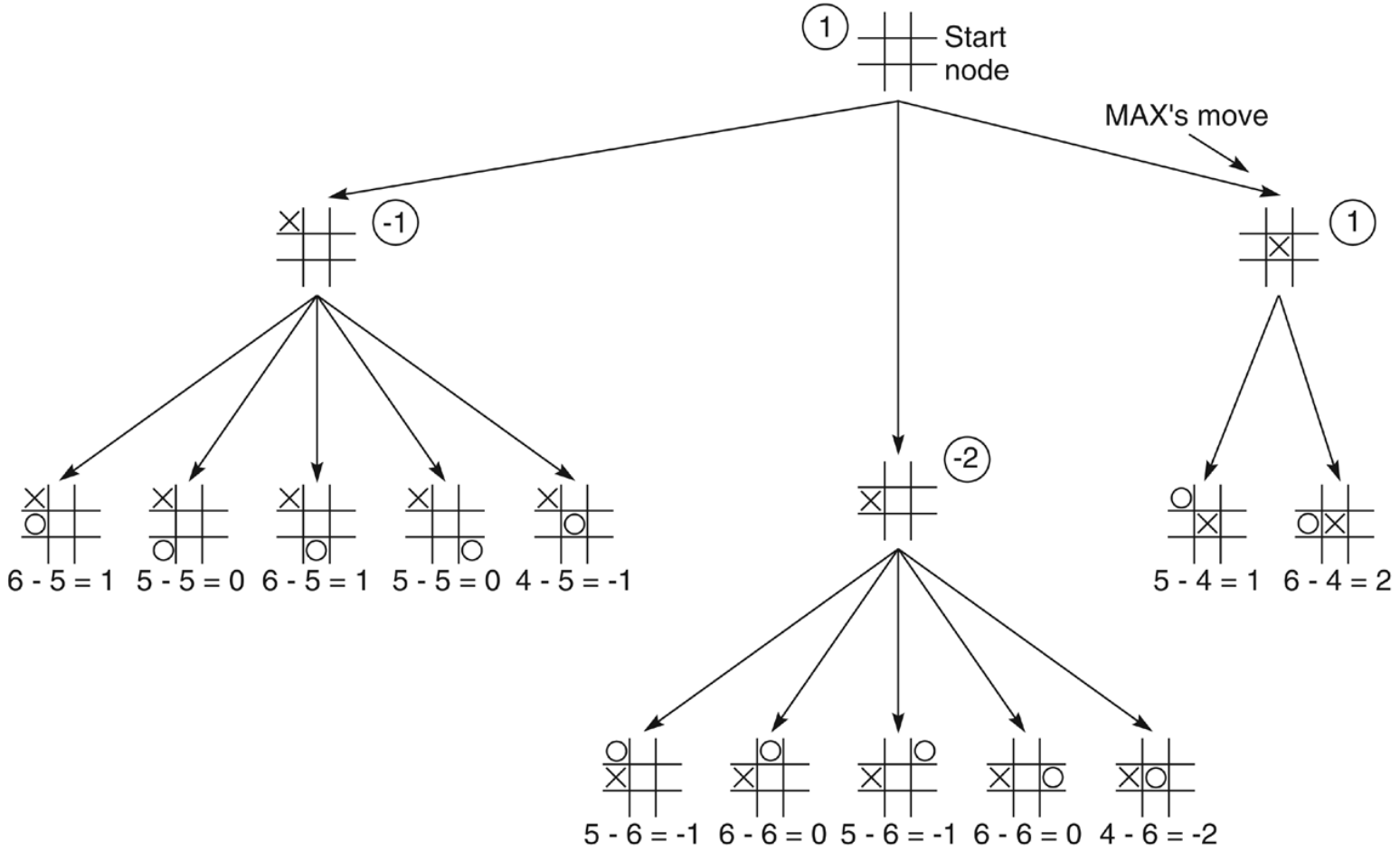
Hàm Heuristic: **E(n) = M(n) – O(n)**

Trong đó: M(n) là *tổng số đường thắng có thể của tôi*

O(n) là *tổng số đường thắng có thể của đối thủ*

E(n) là *trị số đánh giá tổng cộng cho trạng thái n*

Minimax 2 lớp được áp dụng vào nước đi mở đầu trong tic-tac-toe

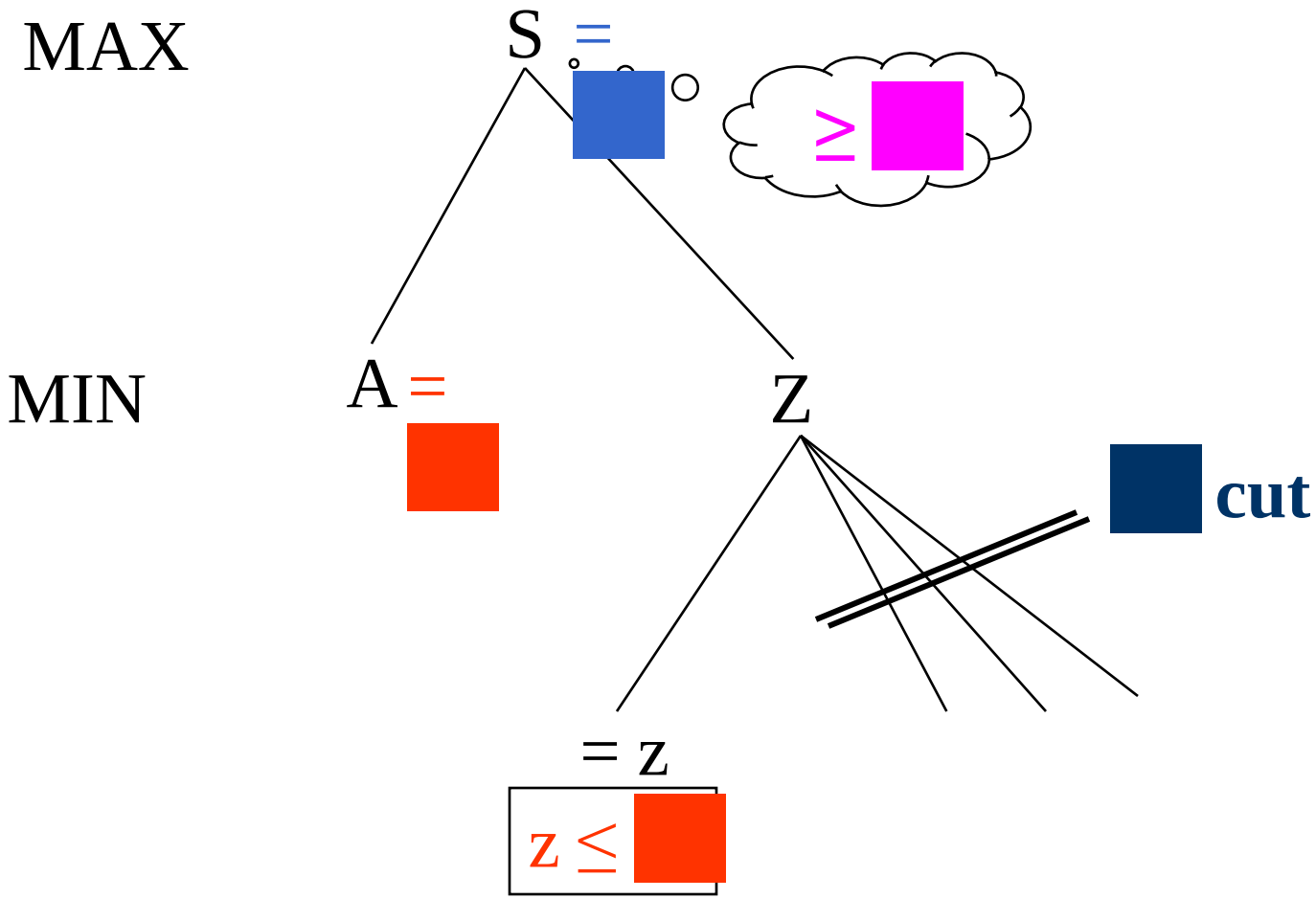


Trích từ Nilsson (1971).

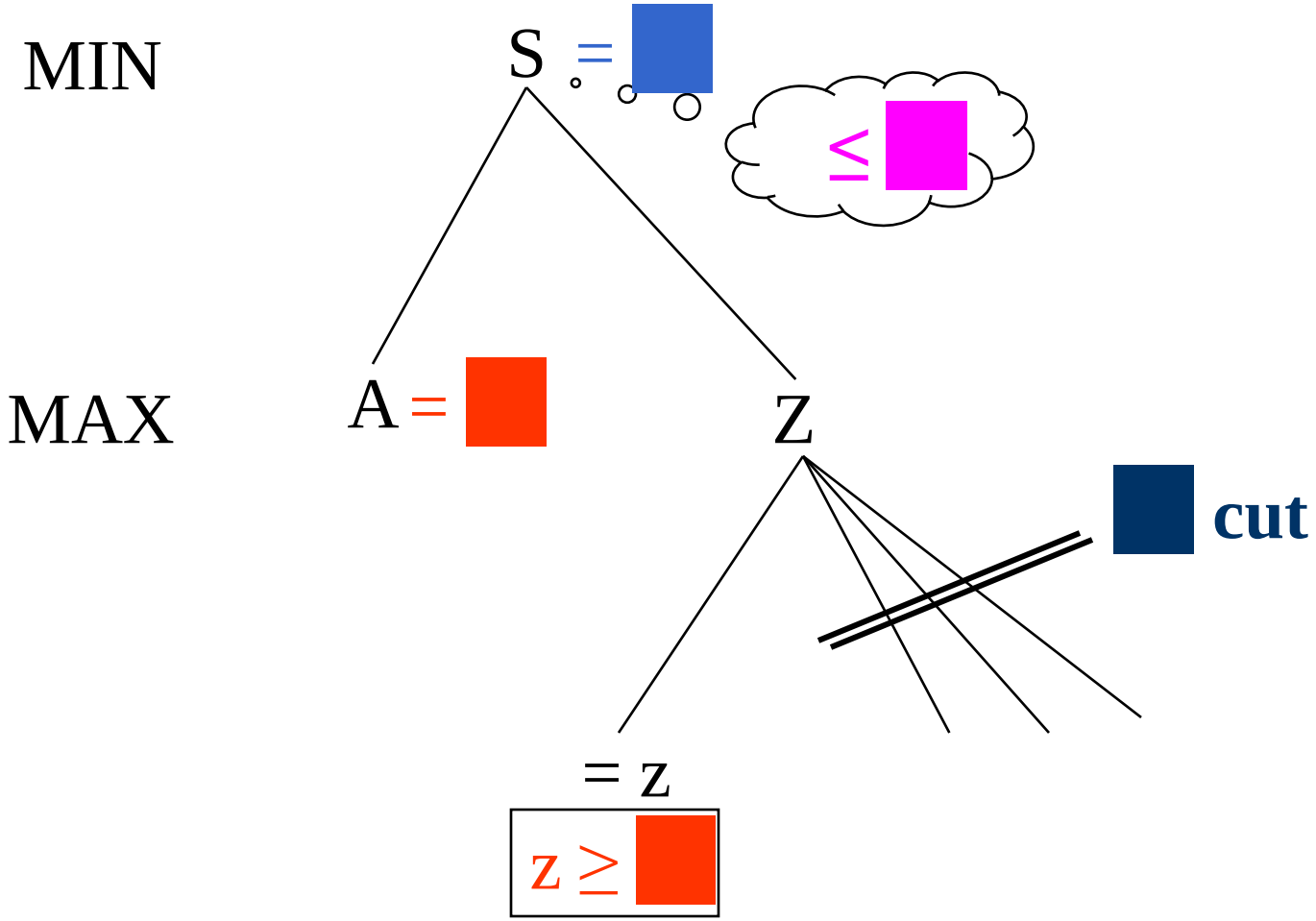
Giải thuật cắt tỉa [redacted]

- Tìm kiếm theo kiểu depth-first.
- Nút MAX có 1 giá trị [redacted] luôn tăng)
- Nút MIN có 1 giá trị [redacted] luôn giảm)
- **TK có thể kết thúc dưới bất kỳ**:
  - Nút MIN nào có [redacted] của bất kỳ nút cha MAX nào.
  - Nút MAX nào có [redacted] của bất kỳ nút cha MIN nào.
- Giải thuật cắt tỉa [redacted] thể hiện *mối quan hệ giữa các nút ở lớp n và n+2*, mà tại đó toàn bộ cây có gốc tại lớp n+1 có thể cắt bỏ.

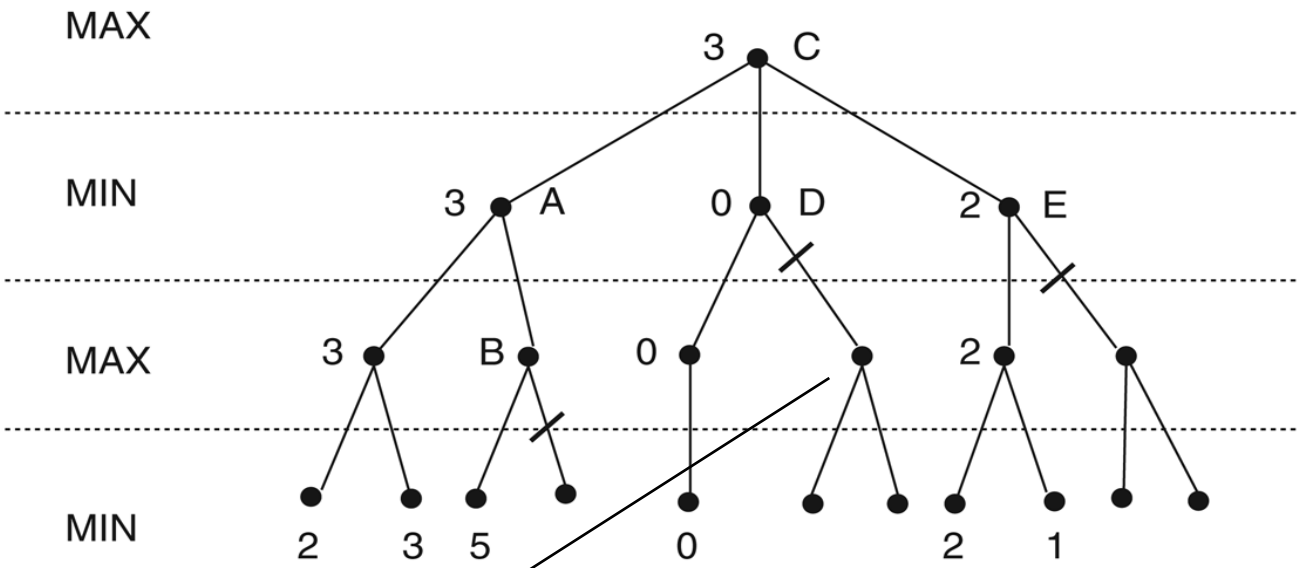
Cắt tỉa



Cắt tỉa



GT Cắt Tỉa [redacted]p dụng cho KGTT giả định



Các nút không có giá trị là các nút không được duyệt qua

A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since $5 > 3$
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since $0 < 3$
E is α pruned, since $2 < 3$
C is 3

Bài Tập Chương 4

TRÍ TUỆ NHÂN TẠO

Artificial Intelligent

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Nội dung môn học – Giới thiệu

- Chương 1: **Giới thiệu**
  - Ngành Trí tuệ nhân tạo là gì?
  - Mục tiêu nghiên cứu của ngành Trí tuệ nhân tạo
  - Lịch sử hình thành và hiện trạng
  - Turing Test

- Chương 2: **Logic vị từ**
  - Mệnh đề & logic vị từ
  - Logic vị từ dưới góc nhìn của AI

Nội dung môn học – Các kỹ thuật tìm kiếm

- Chương 3:**Tìm kiếm trên không gian trạng thái**
  (State Space Search)
  - AI : Biểu diễn và tìm kiếm
  - Các giải thuật tìm kiếm trên không gian trạng thái
  - Depth first search (DFS) - Breath first search (BFS)

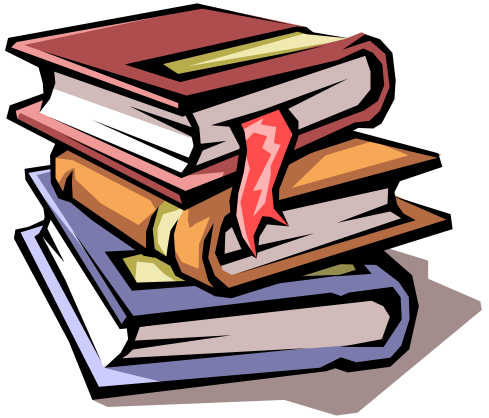
- Chương 4:**Tìm kiếm theo Heuristic**
  - Heuristic là gì?
  - Tìm kiếm theo heuristic
  - Các giải thuật Best first search (BFS), Giải thuật A*
  - Chiến lược Minimax, Alpha Beta

Nội dung môn học – Kỹ thuật phát triển ứng dụng

- Chương 5:**Hệ luật sinh**
  - Tìm kiếm đệ qui
  - Hệ luật sinh: Định nghĩa và ứng dụng
  - Tìm kiếm trên hệ luật sinh
- Chương 6:**Hệ chuyên gia**
  - Giới thiệu về hệ chuyên gia
  - Mô hình hệ chuyên gia: dự trên luật, dựa trên frame
  - Phát triển một hệ chuyên gia
- Chương 7:**Biểu diễn tri thức**
  - Biểu diễn tri thức trong AI: vai trò và ứng dụng
  - Các kỹ thuật biểu diễn tri thức: semantic network, lưu đồ phụ thuộc khái niệm, frame, script

Thực hành &Tài liệu tham khảo

- **Thực hành** Prolog và CLISP
  - Prolog : Các giải thuật tìm kiếm
  - CLISP : Biểu diễn tri thức
  - Bài tập lớn



- **Tài liệu tham khảo**
  - Bài giảng "Trí tuệ nhân tạo" – ThS Nguyễn Cao Trí – KS Lê Thành Sách
  - Artificial Inteligent – George F. Luget & Cilliam A. Stubblefied
  - Giáo trình "Trí tuệ nhân tạo" – KS Nguyễn Đức Cường
  - Trí tuệ nhận tạo – Nguyễn Quang Tuấn – Hà nội

Chương 1: GIỚI THIỆU

- Ngành Trí tuệ nhân tạo là gì?
- Mục tiêu nghiên cứu của ngành Trí tuệ nhân tạo
- Lịch sử hình thành và hiện trạng
- Turing Test

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Đối tượng nghiên cứu của AI

- Đối tượng nghiên cứu của ngành AI

  AI là ngành nghiên cứu về các hành xử thông minh (intelligent behaviour) bao gồm: thu thập, lưu trữ tri thức, suy luận, hoạt động và kỹ năng.

  Đối tượng nghiên cứu là các "hành xử thông minh" chứ không phải là "sự thông minh".

'Không có' Sự Thông Minh

Chỉ có

Biểu hiện thông minh qua hành xử

Sự Thông Minh

- Thông minh hay Hành xử thông minh là gì?
  - **Hành xử thông minh**: là các hoạt động của một đối tượng như là kết quả của một quá trình thu thập, xử lý và điều khiển theo những tri thức đã có hay mới phát sinh (thường cho kết quả tốt theo mong đợi *so với các hành xử thông thường*) là biểu hiện cụ thể, cảm nhận được của "Sự **thông minh**"
  - Khái niệm về tính thông minh của một đối tượng thường biểu hiện qua các hoạt động:
    - Sự hiểu biết và nhận thức được tri thức
    - Sự lý luận tạo ra tri thức mới dựa trên tri thức đã có
    - Hành động theo kết quả của các lý luận
    - Kỹ năng (Skill)

TRI THỨC

???

Tri thức (Knowledge)

- Tri thức là những thông tin chứa đựng 2 thành phần
  - Các khái niệm:
    - Các khái niệm cơ bản: là các khái niệm mang tính quy ước
    - Các khái niệm phát triển: Được hình thành từ các khác niệm cơ bản thành các khái niệm phức hợp phức tạp hơn.
  - Các phương pháp nhận thức:
    - Các qui luật, các thủ tục
    - Phương pháp suy diễn, lý luận,..
- Tri thức là điều kiện tiên quyết của các hành xử thông minh hay "Sự thông minh"
- Tri thức có được qua sự thu thập tri thức và sản sinh tri thức
- Quá trình thu thập và sản sinh tri thức là hai quá trình song song và nối tiếp với nhau – không bao giờ chấm dứt trong một thực thể "Thông Minh"

Tri thức – Thu thập và sản sinh

- Thu thập tri thức:
  - Tri thức được thu thập từ thông tin, là kết quả của một quá trình thu nhận dữ liệu, xử lý và lưu trữ. Thông thường quá trình thu thập tri thức gồm các bước sau:
    - Xác định lĩnh vực/phạm vi tri thức cần quan tâm
    - Thu thập dữ liệu liên quan dưới dạng các trường hợp cụ thể.
    - Hệ thống hóa, rút ra những thông tin tổng quát, đại diện cho các trường hợp đã biết – Tổng quát hóa.
    - Xem xét và giữ lại những thông tin liên quan đến vấn đề cần quan tâm , ta có **các tri thức về vấn đề đó.**
- Sản sinh tri thức:
  - Tri thức sau khi được thu thập sẽ được đưa vào mạng tri thức đã có.
  - Trên cơ sở đó thực hiện các liên kết, suy diễn, kiểm chứng để sản sinh ra các tri thức mới.

Tri thức – Tri thức siêu cấp

- "Trí thức siêu cấp" (meta knowledge) hay "Tri thức về Tri thức"

  Là các tri thức dùng để:

  - Đánh giá tri thức khác
  - Đánh giá kết quả của quá trình suy diễn
  - Kiểm chứng các tri thức mới

- Phương tiện truyền tri thức: ngôn ngữ tự nhiên

Hành xử thông minh – Kết luận

- Hành xử thông minh không đơn thuần là các hành động như là kết quả của quá trình thu thập tri thức và suy luận trên tri thức.
- Hành xử thông minh còn bao hàm
  - Sự tương tác với môi trường để nhận các phản hồi
  - Sự tiếp nhận các phản hồi để điều chỉnh hành động - Skill
  - Sự tiếp nhận các phản hồi để hiệu chỉnh và cập nhật tri thức
- Tính chất thông minh của một đối tượng là sự tổng hợp của cả 3 yếu tố: thu thập tri thức, suy luận và hành xử của đối tượng trên tri thức thu thập được. Chúng hòa quyện vào nhau thành một thể thống nhất " Sự Thông Minh"
- Không thể đánh giá riêng lẽ bất kỳ một khía cạnh nào để nói về tính thông minh.

➔ **THÔNG MINH CẦN TRI THỨC**

Mục tiêu nghiên cứu của ngành AI

Trí tuệ nhân tạo nhằm tạo ra "Máy người"?

Mục tiêu

- Xây dựng lý thuyết về thông minh để giải thích các hoạt động thông minh
- Tìm hiểu cơ chế sự thông minh của con người
  - Cơ chế lưu trữ tri thức
  - Cơ chế khai thác tri thức
- Xây dựng cơ chế hiện thực sự thông minh
- Áp dụng các hiểu biết này vào các máy móc phục vụ con người.

Mục tiêu của AI (tt)

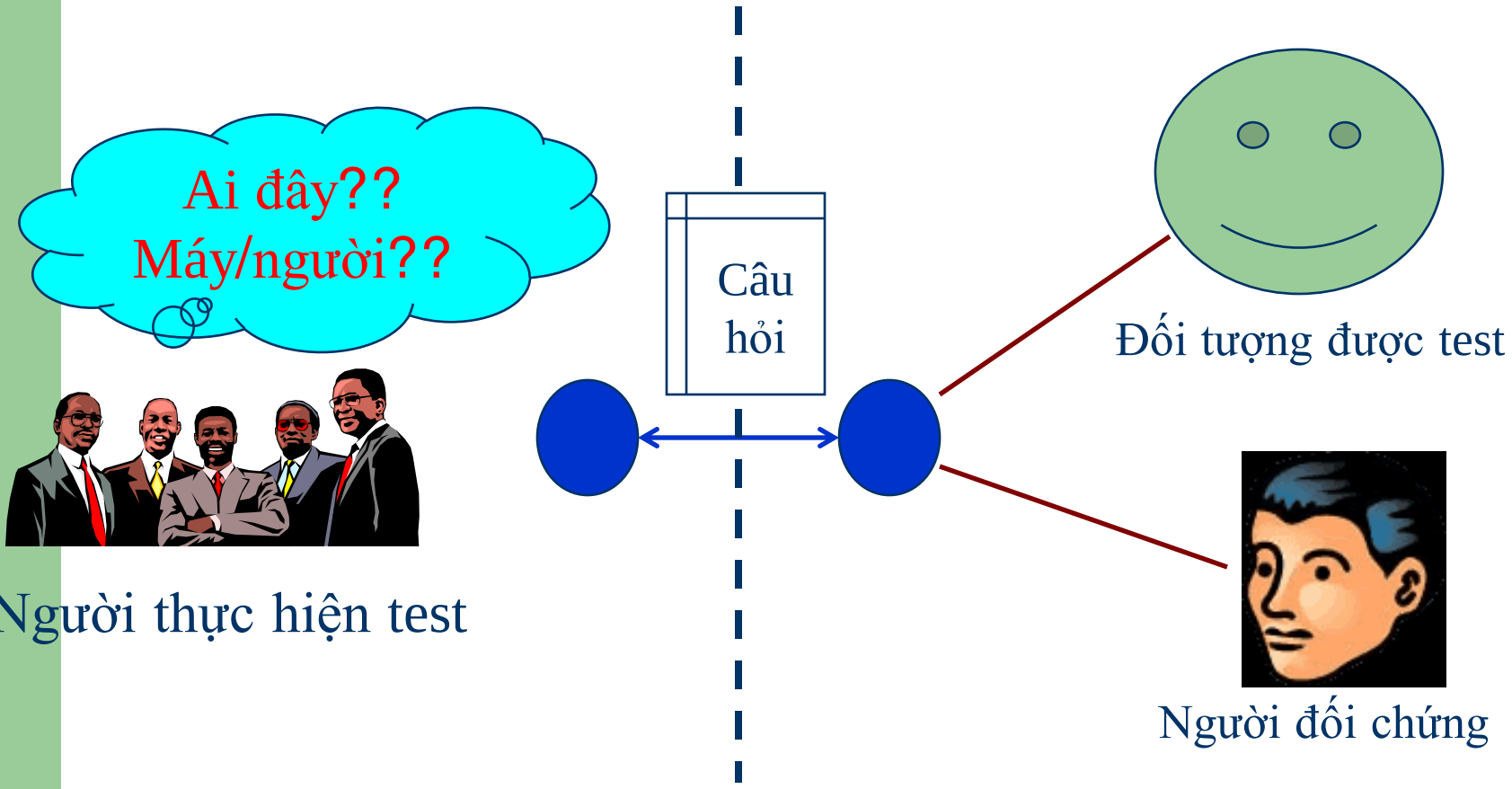
- Cụ thể:
  - Kỹ thuật: xây dựng các máy móc có tính thông minh nhằm đáp ứng tốt hơn nhu cầu của con người.
  - Khoa học: xây dựng và phát triển các khái niệm, thuật ngữ, phương pháp để hiểu được các hành xử thông minh của sinh vật.
  - Đối tượng thường được chú trọng phát triển là máy tính

Sự cần thiết của ngành AI ?????

Làm sao biết máy có thông minh?

Turing Test: Thử tính thông minh

- Bài toán xác định tính thông minh của một đối tượng
- Turing test:



Turing Test: Ưu - Khuyết

- Ưu điểm
  - Đem lại quan điểm khách quan về sự thông minh: Thông minh hay không thể hiện qua các trả lời của các câu hỏi
  - Loại trừ các thành kiến: không thích công nhận tính thông minh của máy móc. Sự thông minh chỉ được đánh giá qua các câu hỏi, không bị chi phối bởi các yếu tố khác.
  - Tránh tình trạng hiểu lầm
- Khuyết điểm:
  - Phép thử tập trung vào các công việc biểu diễn hoàn toàn bằng ký hiệu do đó làm mất một đặc tính rất quan trọng của máy tính là tính toán chính xác và hiệu quả
  - Không thử nghiệm được các khả năng tri giác và khéo léo
  - Giới hạn khả năng thông minh của máy tính theo khuôn mẫu con người. Nhưng con người chưa hẳn là thông minh hoàn hảo.
  - Không có một chỉ số rõ r… …ời tester.

Thông Minh? ➔ Còn tùy ☺

Lịch sử phát triển của AI : Giai đoạn cổ điển

- Giai đoạn cổ điển (1950 – 1965)

  Đây là giai đoạn của 2 lĩnh vực chính:Game Playing (Trò chơi) và Theorem Proving (Chứng minh định ký)

Game Playing: dựa trên kỹ thuật State Space Search với trạng thái (State) là các tình huống của trò chơi. Đáp án cần tìm là trạng thái thắng hay con đường dẫn tới trạng thái thắng. áp dụng với các trò chơi loại đối kháng. Ví dụ: Trò chơi đánh cờ vua.

Có 2 kỹ thuật tìm kiếm cơ bản:

- Kỹ thuật **generate and test :** chỉ tìm được 1 đáp án/ chưa chắc tối ưu.
- Kỹ thuật **Exhaustive search** (vét cạn): Tìm tất cả các nghiệm, chọn lựa phương án tốt nhất.

(Bùng nổ tổ hợp mn với m>=10)

Lịch sử phát triển của AI : Giai đoạn cổ điển (tt)

Theorem Proving: dựa trên tập tiên đề cho trước, chương trình sẽ thực hiện chuỗi các suy diễn để đạt tới biểu thức cần chứng minh.

Nếu có nghĩa là đã chứng minh được. Ngược lại là không chứng minh được.

Ví dụ: Chứng minh các định lý tự động, giải toán,...

Vẫn dựa trên kỹ thuật state space search nhưng khó khăn hơn do mức độ và quan hệ của các phép suy luận: song song, đồng thời, bắc cầu,..

Có các kết quả khá tốt và vẫn còn phát triển đến ngày nay

(Bùng nổ tổ hợp mn , m>=10)

Lịch sử phát triển của AI- Giai đoạn viễn vông

- Giai đoạn viễn vông (1965 – 1975)
  - Đây là giai đoạn phát triển với tham vọng làm cho máy hiểu được con người qua ngôn ngữ tự nhiên.
  - Các công trình nghiên cứu tập trung vào việc biểu diễn tri thức và phương thức giao tiếp giữa người & máy bằng ngôn ngữ tự nhiên.
  - Kết quả không mấy khả quan nhưng cũng tìm ra được các phương thức biểu diễn tri thức vẫn còn được dùng đến ngày nay tuy chưa thật tốt như:
    - Semantic Network (mạng ngữ nghĩa)
    - Conceptial graph (đồ thị khái niệm)
    - Frame (khung)
    - Script (kịch bản)

Vấp phải trở ngại về năng lực của máy tính

Lịch sử phát triển của AI- Giai đoạn hiện đại

- Giai đoạn hiện đại (từ 1975)
  - Xác định lại mục tiêu mang tính thực tiễn hơn của AI là:
    - Tìm ra **lời giải tốt nhất** trong khoảng thời gian chấp nhận được.
    - Không cầu toàn tìm ra **lời giải tối ưu**
  - Tinh thần HEURISTIC ra đời và được áp dụng mạnh mẽ để khắc phục bùng nổ tổ hợp.
  - Khẳng định vai trò của tri thức đồng thời xác định 2 trở ngại lớn là biểu diễn tri thức và bùng nổ tổ hợp.
  - Nêu cao vai trò của Heuristic nhưng cũng khẳng định tính khó khăn trong đánh giá heuristic.

Better than nothing ⇨ Phát triển ứng dụng mạnh mẽ: Hệ chuyên gia, Hệ chuẩn đoán,..

Các lĩnh vực ứng dụng

- Game Playing: Tìm kiếm / Heuristic
- Automatic reasoning & Theorem proving: Tìm kiếm / Heuristic
- Expert System: là hướng phát triển mạnh mẽ nhất và có giá trị ứng dụng cao nhất.
- Planning & Robotic: các hệ thống dự báo, tự động hóa
- Machine learning: Trang bị khả năng học tập để giải quyết vấn đề kho tri thức:
  - Supervised : Kiểm soát được tri thức học được. Không tìm ra cái mới.
  - UnSupervised:Tự học, không kiểm soát. Có thể tạo ra tri thức mới nhưng cũng nguy hiểm vì có thể học những điều không mong muốn.

Các lĩnh vực ứng dụng (tt)

- Natural Language Understanding & Semantic modelling: Không được phát triển mạnh do mức độ phức tạp của bài toán cả về tri thức & khả năng suy luận.
- Modeling Human perfromance: Nghiên cứu cơ chế tổ chức trí tuệ của con người để áp dụng cho máy.
- Language and Environment for AI:Phát triển công cụ và môi trường để xây dựng các ứng dụng AI.
- Neurol network / Parallel Distributed processing: giải quyết vấn đề năng lực tính toán và tốc độ tính toán bằng kỹ thuật song song và mô phỏng mạng thần kinh của con người.

Mô hình phát triển ứng dụng AI

- Mô hình ứng dụng Ai hiện tại:

AI = Presentation & Search

Chương 2: PHÉP TOÁN VỊ TỪ

- Phép toán vị từ dưới góc nhìn của AI
- Mệnh đề
- Vị từ

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

AI & Phép toán vị từ

- Tại sao Ai phải nghiên cứu phép toán vị từ?
  - AI ➔ Phát triển các chương trình có khả năng suy luận
  - Suy luận giúp chương trình AI biết được tính đúng/sai của một vấn đề nào đó.
- Phép toán vị từ ➔ cung cấp một khả năng triển khai các quá trình suy diễn trên máy tính
- Phát triển chương trình AI cần phép toán vị từ.
- Phép toán vị từ được hiện thực bằng ngôn ngữ lập trình trên máy tính ***PROLOG***

AI & Phép toán vị từ: Minh họa 1

Mệnh đề thực tế

- "Nếu trời mưa thì bầu trời có mây"
- Trời đang mưa

Vậy ➔ Bầu trời có mây

Mệnh đề logic

- P="Trời mưa"
- Q= "Bầu trời có mây"

Ta có hai phát biểu sau đúng:

- P➔ Q
- P

Vậy theo luật suy diễn ➔ Q là đúng.

Nghĩa là: "Bầu **trời có mây"**

AI & Phép toán vị từ: Minh họa 2

Mệnh đề thực tế

- "Nếu NAM có nhiều tiền thì NAM đi mua sắm"
- "Nam KHÔNG đi mua sắm"

Vậy ➔ Nam KHÔNG có nhiều tiền

Mệnh đề logic

- P="Nam có nhiều tiền"
- Q= "Nam đi mua sắm"

Ta có hai phát biểu sau đúng:

- P➔ Q
- ¬ Q

Vậy theo luật suy diễn ➔ ¬ P là đúng.

Nghĩa là: "Nam **KHÔNG có nhiều tiền"**

Phép toán mệnh đề : Định nghĩa

Mệnh đề:

- Mệnh đề là một phát biểu khai báo
- Mệnh đề chỉ nhận một trong hai giá trị: ĐÚNG (True) hoặc SAI (False)

Ví dụ:

- Ngày 01tháng giêng là ngày tết cổ truyền
- Môn bạn đang học là AI
- Hôm nay là quốc khánh

- Hôm nay trời lạnh
- Tại sao phải học AI ?

Mệnh đề : Các phép toán

- **Biểu thức mệnh đề**: là sự kết hợp của các mệnh đề bởi các phép toán mệnh đề
- **Các phép toán**:

$\neg$	Phủ định	một ngôi
$\wedge$	Hội	hai ngôi
$\vee$	Tuyển	hai ngôi
$\Rightarrow$	Suy ra	hai ngôi
=	Tương đương	hai ngôi

- Cách đánh giá giá trị của phép toán:

Bảng chân trị

Mệnh đề : Các phép toán – ví dụ

P="Nam học giỏi" ; Q="Nam thông minh" ; R="Nam đẹp trai"

Mệnh đề thực tế	Biểu thức mệnhđề
"Nam học giỏi, thông minh, đẹp trai"	$P \wedge Q \wedge R$
"Nam học giỏi hoặc thông minh"	$P \vee Q$
"Nam hoặc học giỏi, hoặc đẹp trai"	$(P \wedge \neg R) \vee (\neg P \wedge R)$
"Nam thông mình thì học giỏi"	$Q \Rightarrow P$

Mệnh đề : Các biểu thức mệnh đề đúng

- Ký hiệu biểu thức đúng: **wff**
- Thành phần cơ bản là P hay ¬P, với P là một mệnh đề
- Các biểu thức đúng định nghĩa theo dạng luật sinh sau:

Wff= "Thành phần cơ bản"|

¬wff | wff^wff |

wff v wff |

wff ⇒ wff |

wff = wff |

(wff)

Mệnh đề : Ngữ nghĩa

- ***Ngữ nghĩa*** của một biểu thức mệnh đề là ***giá trị*** của biểu thức mệnh đề đó.
- Giá trị của biểu thức mệnh đề là có khả năng tính toán được. Trong đó:
  - Mỗi mệnh đề được gán một giá trị true hay false
  - Mỗi toán tử được đánh giá theo bảng chân trị và thứ tự ưu tiên của toán tử.
- Giá trị của biểu thức mệnh đề tính bằng cách:
  - Dùng bảng chân trị
  - Đánh giá ngược từ node lá khi biểu thức mệnh đề được biểu diễn ở dạng cây

Mệnh đề : Các tương đương

- Các tương đương được sử dụng thường xuyên trong quá trình biến đổi một biểu thức từ dạng này sang dạng khác.
- Khả năng biến đổi tương đương trên máy tính có thể được làm tự động
- Các tương đương:

Trong các tương đương sau A,B,C là các mệnh đề bất kỳ.

- **Dạng phủ định kép**

  $\neg\neg A = A$

- **Dạng tuyển**

  $A \vee TRUE = TRUE$

  $A \vee FALSE = A$

  $A \vee A = A$

  $A \vee \neg A = TRUE$

- **Dạng hôäi**

  $A \wedge TRUE = A$

  $A \wedge FALSE = FALSE$

  $A \wedge A = A$

  $A \wedge \neg A = FALSE$

Mệnh đề : Các tương đương (tt)

- **Dạng suy ra**

  $A \Rightarrow TRUE = \mathbf{TRUE}$

  $\mathbf{A} \Rightarrow FALSE = \mathbf{\neg A}$

  $TRUE \Rightarrow A = \mathbf{A}$

  $FALSE \Rightarrow A = \mathbf{TRUE}$

  $\mathbf{A} \Rightarrow \mathbf{A} = \mathbf{TRUE}$

- **Dạng hấp thu**

  $A \wedge (A \vee B) = \mathbf{A}$

  $A \vee (A \wedge B) = A$

  $A \wedge (\neg A \vee B) = A \wedge B$

  $A \vee (\neg A \wedge B) = A \vee B$

- **Dạng De Morgan**

  $\neg (A \wedge B) = \mathbf{\neg A \vee \neg B}$

  $\neg (A \vee B) = \mathbf{\neg A \wedge \neg B}$

- **Dạng khác**

  $A \Rightarrow B = \mathbf{\neg A \vee B}$

  $\neg (A \Rightarrow B) = \mathbf{A \wedge \neg B}$

  $A \Rightarrow B =$

  $\mathbf{A \wedge \neg B} \Rightarrow FALSE$

- Phép $\wedge$ và $\vee$ có khả năng kết hợp.
- Phép $\wedge$ và $\vee$ có khả năng hoán vị.
- Phép $\wedge$ có khả năng phân phối trên $\vee$

  $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
- Phép $\vee$ có khả năng phân phối trên $\wedge$

  $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Mệnh đề : Các dạng chuẩn CNF & DNF

- **Dạng chuẩn** *là kết xuất chuẩn của các giải thuật làm việc với phép toán mệnh đề.*
- **Tuyển cơ bản**: *là thành phần cơ bản hay sự kết hợp của các thành phần cơ bản bằng phép tuyển(v)*
- **Hội cơ bản**: *là thành phần cơ bản hay sự kết hợp của các thành phần cơ bản bằng phép hội (^).*
- **Dạng chuẩn hội – CNF**: *là thành phần* ***tuyển*** *cơ bản hay các* ***tuyển*** *cơ bản kết hợp bởi phép* ***hội***.
- **Dạng chuẩn tuyển – DNF**: *là thành phần* ***hội*** *cơ bản hay các* ***hội*** *cơ bản kết hợp bởi phép* ***tuyển***.

Mệnh đề : Luật suy diễn & chứng minh

Luật suy diễn được áp dụng để phát triển các ứng dụng có khả năng suy luận. Suy luận là hoạt động thường xuyên của con người để hiểu các lý lẽ, kiểm chứng, phán đoán các vấn đề.

- Luật Modus Ponens (MP)

  $A, A \Rightarrow B \quad \therefore \quad B$
- Luật Modus Tollens (MT)

  $A \Rightarrow B, \neg B \quad \therefore \quad \neg A$
- Luật Hội

  $A, B \quad \therefore \quad A \wedge B$
- Luật đơn giản

  $A \wedge B \quad \therefore \quad A$
- Luật Cộng

  $A \quad \therefore \quad A \vee B$
- Luật tam đoạn luận tuyển

  $A \vee B, \neg A \quad \therefore \quad B$
- Luật tam đoạn luận giả thiết

  $A \Rightarrow B, B \Rightarrow C \quad \therefore \quad A \Rightarrow C$

Mệnh đề : Luật suy diễn & chứng minh – Ví dụ 1

- Ta có các biểu thức sau: AvB, AvC,và ¬A là TRUE
- Chứng minh B^C có trị TRUE

1	AvB	P (tieân ñeà)
2	AvC	P (tieân ñeà)
3	¬A	P (tieân ñeà)
4	B	1,3, tam ñoaïn luaän tuyeån
5	C	2,3, tam ñoaïn luaän tuyeån
6	B^C	4,5, Luaät hoäi

- Đã chứng minh xong

Mệnh đề : Luật suy diễn & chứng minh – Ví dụ 2

- Ta có các biểu thức sau là đúng:

  AvB, A⇒ C, B⇒ D, ¬D **Chứng minh C**

- *Ta giả thiết ¬C dẫn đến false*

1	AvB	P (tieân ñeà)
2	A⇒C	P (tieân ñeà)
3	B⇒ D	P (tieân ñeà)
4	¬D	P (tieân ñeà)
5	¬C	P (giaû thieát phaûn chöùng)
6	¬B	3,4,Modus Tollens
7	A	1,6, Tam ñoaïn luaän tuyeån
8	¬A	2,5,Modus Tollens
9	A ^¬A	7,8, Luaät hoäi
10	False	Maâu thuaãn vôùi Luaät hoäi

Mệnh đề : Luật phân giải mệnh đề

- **Clause**: *là tuyển của không hay nhiều thành phần cơ bản.*
- **Dạng clause**:*là **hội** của một hay nhiều* Clause
- Luật phân giải mệnh đề:

$$P \vee D1,\ \neg P \vee D2 \qquad \therefore (D1-P) \vee (D2-\neg P)$$

  - D1,D2 là tuyển của không hay một thành phần cơ bản.
  - P là mệnh đề
  - D1-P : là một clause thu được bằng cách xóa bỏ các P trong D1
  - D2- ¬P : là một clause thu được bằng cách xóa bỏ các ¬P trong D2

Mệnh đề : Luật phân giải mệnh đề (tt)

- Luật phân giải bảo toàn tính Unsatisfiable
- S là unsatisfiable ⇔ Rn(S)cũng unsatisfiable

R: luật phân giải, n số lần áp dụng R trên S, n>0

- Ứng dụng của luật phân giải : dùng để chứng minh: *Có S là tập các clause, dùng S chứng minh biểu thức mệnh đề W*
- Phương pháp:
  i. *Thành lập phủ định của W*
  ii. *Đưa* ¬W về dạng clause
  iii. Thêm clause trong bước ii vào S thành lập S1
  iv. Dùng luật phân giải trên S1 để dẫn ra clause rỗng.

Mệnh đề : Luật phân giải mệnh đề - Ví dụ

- *Cho đoạn sau:*

“ Nam đẹp trai, giàu có. Do vậy, Nam hoặc là phung phí hoặc là nhân từ và giúp người. Thực tế, Nam không phung phí hoặc cũng không kêu căng.”

“Do vậy, có thể nói Nam là người nhân từ”

- *Kiểm chứng kết quả suy luận trên, bằng luật phân giải.*

Mệnh đề : Luật phân giải mệnh đề - Ví dụ

- **(i) Chuyển sang mệnh đề:**
  - P1 = "Nam đẹp trai."
  - P2 = "Nam giàu có."
  - P3 = "Nam phung phí."
  - P4 = "Nam kêu căng."
  - P5 = "Nam nhân từ."
  - P6 = "Nam giúp người."

Các biểu thức thành lập được từ đoạn trên:

- Wff1 = P1 ^ P2
- Wff2 = (P1 ^ P2) => (P3 ^ ¬(P5 ^ P6)) v (¬P3 ^ (P5 ^ P6))
- Wff3 = ¬P3 ^ ¬P4

Wff4 = P5 Biểu thức cần chứng minh.

Mệnh đề : Luật phân giải mệnh đề - Ví dụ

- **(ii) Đưa về dạng clause:**
- Wff1, sinh ra hai clause: **C1 = P1** **C2 = P2**
- Wff2 = ¬(P1 ^ P2) v ((P3 ^ ¬(P5 ^ P6)) v (¬P3 ^ (P5 ^ P6)))

.....

= (¬P1 v ¬P2 v P3 v ¬P3 v ¬P6) ^ (¬P1 v ¬P2 v P5 v ¬P3 v ¬P6)^(¬P1 v ¬P2 v P3 v P3 v P5) ^ (¬P1 v ¬P2 v P5 v P3 v P5) ^ (¬P1 v ¬P2 v P3 v P5 v ¬P6)^ (¬P1 v ¬P2 v P5 v P5 v ¬P6) ^(¬P1 v ¬P2 v P3 v P3 v P6) ^ (¬P1 v ¬P2 v P5 v P3 v P6)

Sinh ra các clause:

C3 = (¬P1 v ¬P2 v ¬P6) | **C4 = (¬P1 v ¬P2 v P5 v ¬P3 v ¬P6)**

C5 = (¬P1 v ¬P2 v P3 v P5) | **C6 = (¬P1 v ¬P2 v P3 v P5)**

C7 = (¬P1 v ¬P2 v P3 v P5 v ¬P6) | **C8 = (¬P1 v ¬P2 v P5 v ¬P6)**

C9 = (¬P1 v ¬P2 v P3 v P6) | **C10 =(¬P1 v ¬P2 v P5 v P3 v P6)**

- Wff3 sinh ra các clause: **C11 = ¬P3** **C12 = ¬P4** **C13 = ¬P5**

(gồm cả bước lấy phủ định kết luận)

Mệnh đề : Luật phân giải mệnh đề - Ví dụ

●iii) áp dụng luật phân giải trên các clause:

TT	Clauses	Luật áp dụng
1	P1	P
2	P2	P
3	¬P1 v ¬P2 v ¬P6	P
4	¬P1 v ¬P2 v P5 v ¬P3 v ¬P6	P
5	¬P1 v ¬P2 v P3 v P5	P
6	¬P1 v ¬P2 v P3 v P5	P
7	¬P1 v ¬P2 v P3 v P5 v ¬P6	P
8	¬P1 v ¬P2 v P5 v ¬P6	P
9	¬P1 v ¬P2 v P3 v P6	P
10	¬P1 v ¬P2 v P5 v P3 v P6	P
11.	¬P3	P
12.	¬P4	P
13	¬P5	P
14	¬P2 v ¬P6	1,2, R
15	¬P6	2, 14, R
16	¬P1 v ¬P2 v P5 v P3	10,15,R
17	¬P2 v P5 v P3	1,16,R
18	P5 v P3	2,17, R
19	P3	13, 18, R
20	□	11, 19, R

ĐÃ CHỨNG MINH

Logic Vị từ: Tại sao?

- Phép toán mệnh đề ➔ suy diễn tự động nhưng **chưa đủ** khi cần phải truy cập vào thành phần nhỏ trong câu, dùng biến số trong câu.

Ví dụ:

"Mọi sinh viên trường ĐHBK đều có bằng tú tài. Lan không có bằng tú tài. Do vậy, Lan không là sinh viên trường ĐHBK"

"Lan" là một đối tượng cụ thể của "SV trường ĐHBK" – không thể đặc tả được "quan hệ" này trong mệnh đề được mà chỉ có thể là:

"LAN là sinh viên trường ĐHBK thì Lan có bằng tú tài. Lan không có bằng tú tài. Do vậy, Lan không là sinh viên trường ĐHBK"

- Mệnh đề phải giải quyết bằng cách liệt kê tất cả các trường hợp

→ Không khả thi

- Do đó, chúng ta cần một Logic khác hơn là phép toán mệnh đề:

PHÉP TOÁN VỊ TỪ

Vị từ: Định nghĩa

- ***Vị từ là một phát biểu nói lên quan hệ giữa một đối tượng với các thuộc tính của nó hay quan hệ giữa các đối tượng với nhau.***

Vị từ được ***biểu diễn*** bởi một tên được gọi là ***tên vị từ***, theo sau nó là một danh sách các thông số.

Ví dụ:

+ Phát biểu: ***"Nam là sinh viên trường ĐHBK"***

+ Biểu diễn: ***sv_bk("Nam")***

Ý nghĩa: đối tượng tên là "Nam" có thuộc tính là "sinh viên trường ĐHBK".

Vị từ: Biểu diễn vị từ – Cú pháp

- Chúng ta có bao nhiêu cách biểu diễn đúng cú pháp cho phát biểu nói trên?

➔ ***Không biết bao nhiêu nhưng chắc chắn nhiều hơn 1*** ☺

Ví dụ chúng ta có thể thay đổi các tên vị từ thành các tên khác nhau như : sinhvien_bk, sinhvien_bachkhoa, … Tất cả chúng đều đúng cú pháp.

Một số quy ước/ chú ý khi biểu diễn:

- Không mô tả những vị từ thừa, có thể suy ra từ một tập các vị từ khác. Hình thức thừa cũng tương tự dư (thừa) dữ liệu khi thiết kế CSDL.
- Tên vị từ phải có tính gợi nhớ. Cụ thể, trong ví dụ trên chúng ta có thể biểu diễn bởi ***q("Nam")***, nhưng rõ ràng cách này không mấy thân thiện và dễ nhớ.

Bạn có biết q("Nam") có nghĩa gì ???

Vị từ: Biểu diễn vị từ – Cú pháp (tt)

Dạng vị từ: tên_vị_từ($term_1$, $term_2$, …, termn)

- Tên vị từ: [a..z](a..z| A..Z| 0..9|_)*

  Bắt đầu bởi một ký tự chữ thường.

  Ví dụ: ban_than, banThan,bAN_THAN,…
- Term có thể là: Hằng,Biến, Biểu thức hàm.
- Hằng: có thể hằng chuỗi hay hằng số.

  Hằng chuỗi: **[“](a..z| A..Z| 0..9|_)*[“] hay [a..z](a..z| A..Z| 0..9|_)***

  Ví *dụ:* *“Nam”, “nam”, “chuoi”,* nam, chuoi, qua,…

  Hằng số: **(0..9)*** Ví dụ: 10, 32,..
- Biến: [A..Z](a..z| A..Z| 0..9|_)* Ví dụ: Nguoi, NGUOI,..
- Biểu thức hàm có dạng: **tên_hàm($term_1$, $term_2$, …, termk)**

  Trong đó **Tên hàm = [a..z](a..z| A..Z| 0..9|_)***

Vị từ : Biểu thức vị từ

- **Biểu thức Vị từ**: là sự kết hợp của các vị từ bởi các phép toán vị từ.
- **Các phép toán**:

¬	Phủ định		- một ngôi.
∀X	Với mọi		- một ngôi
∃X	**Tồn** tại	- một ngôi	
∧	Hội	- hai ngôi.	
v	Tuyển	- hai ngôi.	
=>	Suy ra		- hai ngôi.
=	Tương đương		- hai ngôi.

Vị từ: Các biểu thức vị từ đúng

- **Biểu thức vị từ đúng được ký hiệu wff.**
- **Biểu thức cơ bản: Có thể là một vị từ , một đại diện trị TRUE (trị là T - đúng), một đại diện trị FALSE (trị là F - sai).**
- **Một biểu thức đúng cú pháp được định nghĩa như sau:**

Wff = "Biểu thức cơ bản" |¬ wff |wff ^ wff |wff v wff |wff=>wff |wff

= wff |(wff) |∀X wff |∃X wff

Với

- **X : Là một biến.**
- **∀ : Lượng từ với mọi.**
- **∃ : Lượng từ tồn tại.**

Vị từ: Lượng từ

- **Giả sử chúng ta có:**

Nam là học sinh khá. Lan là học sinh trung bình. Mai học sinh khá

- Xét tập **D = [Nam, Lan, Mai]**
- Gọi **p(X)** cho biết: "X **là học sinh khá"** ta có các vị từ

p("Nam") : trị là T. p("Lan") : trị là F. p("Mai") : trị là T.

- **Lượng từ tồn tại:**

Xét mệnh đề **p("Nam") v p("Lan") v p("Mai")** có thể biểu diễn bằng vị từ **∃X ∈ D: p(X)**

"Tồn **tại X thuộc tập D, mà X là học sinh khá"**

- **Lượng từ với mọi:**

Xét mệnh đề **p("Nam") ^ p("Lan") ^ p("Mai")** có thể biểu diễn bằng vị từ **∀X ∈ D: p(X)**

"Mọi **X thuộc tập D đều là học sinh khá"**

Vị từ: Biểu diễn thế giới thực

- Chuyển các câu sau sang biểu thức vị từ:

  "Mọi sinh viên trường ĐHBK đều có bằng tú tài.

  Lan không có bằng tú tài.

  Do vậy, Lan không là sinh viên trường ĐHBK"

- Với *sv_bk(X) cho biết: "X là sinh viên trường DHBK"*

  tu_tai(X) cho biết: "X có bằng tú tài"

- Các câu trên được chuyển qua vị từ là:

  ∀X(sv_bk(X) => tu_tai(X)).

  ¬tu_tai("Lan").

  Do vậy, ¬sv_bk("Lan").

Vị từ: Biểu diễn thế giới thực (tt)

- **"Chỉ vài sinh viên máy tính lập trình tốt."**

  với sv_mt(X) : "X là sinh viên máy tính"

  laptrinh_tot(X) : "X lập trình tốt"

  Câu trên chuyển sang vị từ là: **∃X(sv_mt(X) ^ laptrinh_tot(X))**

- **"Không một sinh viên máy tính nào không cần cù."**

  với: sv_mt(X) : "X là sinh viên máy tính

  can_cu(X) : "X cần cù"

  Câu trên chuyển sang là: **∀X (sv_mt(X) => can_cu(X))**

- **"Không phải tất cả các sinh viên máy tính đều thông minh"**

  với thong_minh(X) : "X thông minh"

  Câu trên chuyển sang là: **∃X(sv_mt(X) ^ ¬thong_minh(X))**

Vị từ: Ngữ nghĩa

- **Vấn đề:** Nếu chúng ta có biểu thức sau:

  $\forall X \exists Y\ p(X,Y)$

  Chúng ta hiểu như thế nào ????!

  -> Cần sự diễn dịch.

+ Cách hiểu 1:

X, Y : là con người.

p(X,Y) cho biết : "X là cha của Y"

Do vậy:

$\forall X \exists Y\ p(X,Y)$ có thể hiểu là:

"Mọi người X, tồn tại người Y để X là cha của Y"

-> wff = $\forall X \exists Y\ p(X,Y)$ có trị là F (sai)

Vị từ: Ngữ nghĩa (tt)

+ Cách hiểu 2:

X, Y : là con người.

p(X,Y) cho biết : "Y là cha của X"

Do vậy:

$\forall X \exists Y$ p(X,Y) có thể hiểu là:

"Mọi người X, tồn tại người Y là cha của X"

-> wff = $\forall X \exists Y$ p(X,Y) có trị là T (đúng)

+ Cách hiểu 3:

X, Y : là số nguyên.

p(X,Y) cho biết : "Y bằng bình phương của X"

-> wff = $\forall X \exists Y$ p(X,Y) có trị là T (đúng)

Vị từ: Ngữ nghĩa (tt)

- **Diễn dịch:** gồm

- Tập D, không rỗng, miền diễn dịch.
- Các phép gán:
  - Vị từ : Quan hệ trên D
  - Hàm : Hàm (ánh xạ) trên D
  - Biến tự do : Một trị trên D, cùng một trị cho các xuất hiện
  - Hằng : Một trị trên D, cùng một trị cho các xuất hiện

Vị từ: Ngữ nghĩa (tt)

- **Ngữ nghĩa:**

  Có diễn dịch I trên miền D của wff.

  - Wff không có lượng từ:

    Ngữ nghĩa = trị sự thật (T|F) của wff khi áp dụng diễn dịch

  - wff có lượng từ:

    ∃XW là T, nếu: W(X/d) là T cho một d thuộc D

    ngược lại: ∃XW là F

    ∀XW là T, nếu: W(X/d) là T cho mọi d thuộc D

    ngược lại: ∀XW là F

Vị từ: Khái niệm

Có I : diễn dịch, E là wff

- **Model:**

  I là cho E có trị T ---> I là Model của E

  Ngược lại: ---> I là **CounterModel** của E

- **Valid:**

  E là valid nếu mọi diễn dịch I đều là Model.

  Ngược lại là : **Invalid**

- **Unsatisfiable:**

  E là unsatisfiable : mọi I đều là CounterModel

  Ngược lại :**Satisfiable**

Vị từ: Tương đương

- Từ tương đương của mệnh đề:

  Nếu chúng ta thay thế các mệnh đề bởi các biểu thức vị từ, các mệnh đề cùng tên thì được thay cùng một biểu thức vị từ, thì được một tương đương của vị từ.

  Ví dụ:

  Mệnh đề:

  $(P \Rightarrow Q) = (\neg P \vee Q)$

  Vị từ:

  P bởi: $\forall X \exists Y p(X,Y)$, Q bởi: $q(X)$

  tương đương:

  $(\forall X \exists Y p(X,Y) \Rightarrow q(X)) = (\neg(\forall X \exists Y p(X,Y)) \vee q(X))$

Vị từ: Tương đương

- Lượng từ:

  $\neg(\forall X\ W) = \exists X(\neg W)$

  $\neg(\exists X\ W) = \forall X(\neg W)$

  Với W là một wff

- Tương đương có ràng buộc:

  Sau đây: Y: biến, W(X): wff *có chứa* biến X, C là wff không chứa X

  Ràng buộc: Y không xuất hiện trong W(X)

  Tương đương:

  - $\forall X\ W(X) = \forall Y\ W(Y)$
  - $\exists X\ W(X) = \exists\ Y\ W(Y)$

Vị từ: Tương đương

Tương đương:

Dạng tuyển:

- $C \vee \forall X A(X) = \forall X(C \vee A(X))$
- $C \vee \exists X A(X) = \exists X(C \vee A(X))$

Dạng hội:

- $C \wedge \forall X A(X) = \forall X(C \wedge A(X))$
- $C \wedge \exists X A(X) = \exists X(C \wedge A(X))$

Dạng suy ra:

- $C \Rightarrow \forall X A(X) = \forall X(C \Rightarrow A(X))$
- $C \Rightarrow \exists X A(X) = \exists X(C \Rightarrow A(X))$
- $\forall X A(X) \Rightarrow C = \forall X(A(X) \Rightarrow C)$
- $\exists X A(X) \Rightarrow C = \exists X(A(X) \Rightarrow C)$

Vị từ: Dạng chuẩn Prenex

- Dạng Chuẩn Prenex:

  $Q_1X_1Q_2X_2\ldots QnXnM$

  Qi : $\forall, \exists$.

  M : wff không có lượng từ.

  Ví dụ:

  - sv_bk(x)
  - $\forall X(sv\_bk(X) \wedge hoc\_te(X))$
  - $\forall X \exists Y cha(X,Y)$

- Giải thuật đưa wff về chuẩn Prenex:
  - Đổi tên biến --> wff không còn lượng từ cùng tên biến, biến lượng từ không trùng tên biến tự do.
  - Đưa lượng từ sang trái dùng tương đương.

Vị từ: Dạng chuẩn Prenex

- Dạng chuẩn Tuyển Prenex:

  $Q1X_1Q_2X_2\ldots QnXn(C_1 \text{ v } \ldots \text{ v } Ck)$

  Ci : Thành phần hội cơ bản.

- Dạng chuẩn Hội Prenex:

  $Q1X_1Q_2X_2\ldots QnXn(D_1 \text{ v } \ldots \text{ v } Dk)$

  Di : Thành phần tuyển cơ bản.

- Giải thuật:
  - Đổi tên biến.
  - Loại bỏ => bởi : A => B = ¬A v B
  - Chuyển ¬ sang phải dùng De Morgan và phủ định kép.
  - Chuyển lượng từ sang trái dùng tương đương.
  - Phân phối v trên ^ (CNF), hay ^ trên v (DNF)

Chương 3: TÌM KIẾM TRÊN KHÔNG GIAN TRẠNG THÁI
(State Space Search)

- AI : Biểu diễn và tìm kiếm
- Không gian tìm kiếm
- Graph Search
- Các giải thuật tìm kiếm trên không gian trạng thái
- Depth first search (DFS) - Breath first search (BFS)

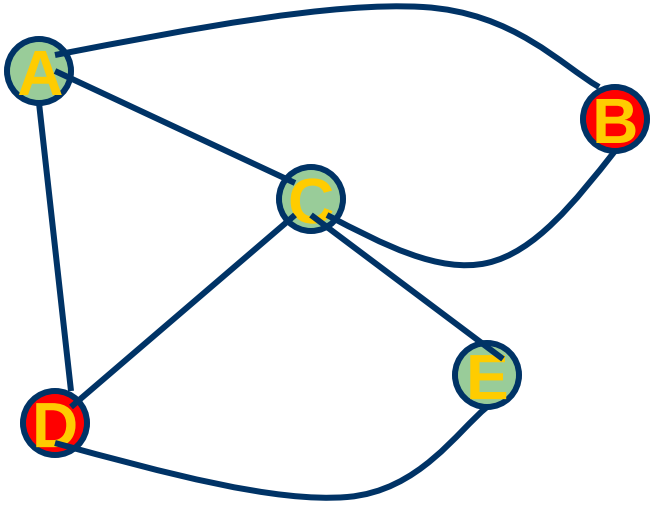
ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn
KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

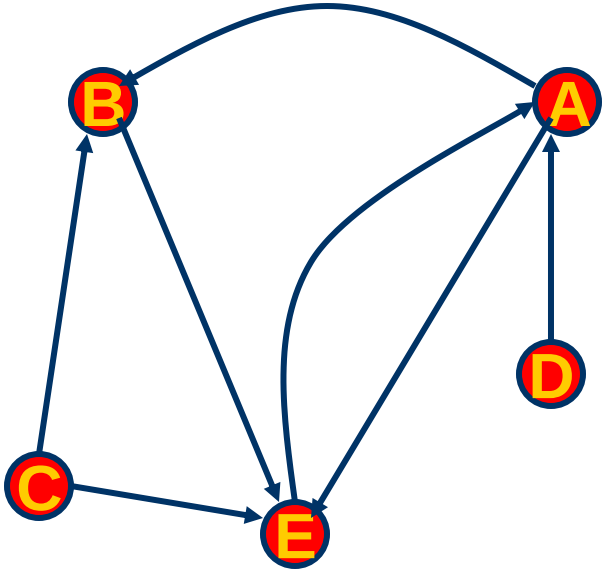
Tại sao phải tìm kiếm?

- Tìm kiếm cái gì?
- Biểu diễn và tìm kiếm là kỹ thuật phổ biến giải các bài toán trong lĩnh vực AI
- Các vấn đề khó khăn trong tìm kiếm với các bài toán AI
  - Đặc tả vấn đề phức tạp
  - Không gian tìm kiếm lớn
  - Đặc tính đối trượng tìm kiếm thay đổi
  - Đáp ứng thời gian thực
  - Meta knowledge và kết quả "tối ưu"
- Khó khăn về kỹ thuật

Lý thuyết đồ thị - Review

- Đồ thị: là một cấu trúc bao gồm:
  - Tập các nút N1, N2,… Nn,.. Không hạn chế
  - Tập các cung nối các cặp nút, có thể có nhiều cung trên một cặp nút





Nút: {A,B,C,D,E}

Cung: {(a,d), (a,b), (a,c), (b,c), (c,d), (c

Đặc tính đồ thị

- **Đồ thị có hướng**: là đồ thị với các cung có định hướng, nghĩa là cặp nút có quan hệ thứ tự trước sau **theo từng cung.** Cung (Ni,Nj) có hướng từ Ni đến Nj, Khi đó Ni là nút cha và Nj là nút con.
- **Nút lá**: là nút không có nút con.
- **Path**: là chuỗi có thứ tự các nút mà 2 nút kế tiếp nhau tồn tại một cung.
- **Đồ thị có gốc**: Trên đồ thị tồn tại nút X sao cho tất cả các path đều đi qua nút đó. X là gốc - Root
- **Vòng** : là một path đi qua nút nhiều hơn một lần
- **Cây**: là graph mà không có path vòng
- **Hai nút nối nhau** :nếu có một path đi qua 2 nút đó

Không gian trạng thái

- **Định nghĩa**:Không gian trạng thái là một hệ thống gồm **4** thành phần [N,A,S,GD]. Trong đó:
  - N là tập nút của Graph. Mỗi nút là một trạng thái của quá trình giải quyết vấn đề
  - A: Tập các cung nối giữa các nút N. Mỗi cung là một bước trong giải quyết vấn đề. Cung có thể có hướng
  - S: Tập các trạng thái bắt đầu. S khác rỗng.
  - GD: Tập các trạng thái đích. GD Không rỗng.
- **Solution path**: Là một path đi từ một nút bắt đầu Si đến một nút kết thúc GDj .
- Mục tiêu của các giải thuật tìm kiếm là tìm ra một solution path và/hay solution path tốt nhất.

Biểu diễn không gian trạng thái-Ví dụ

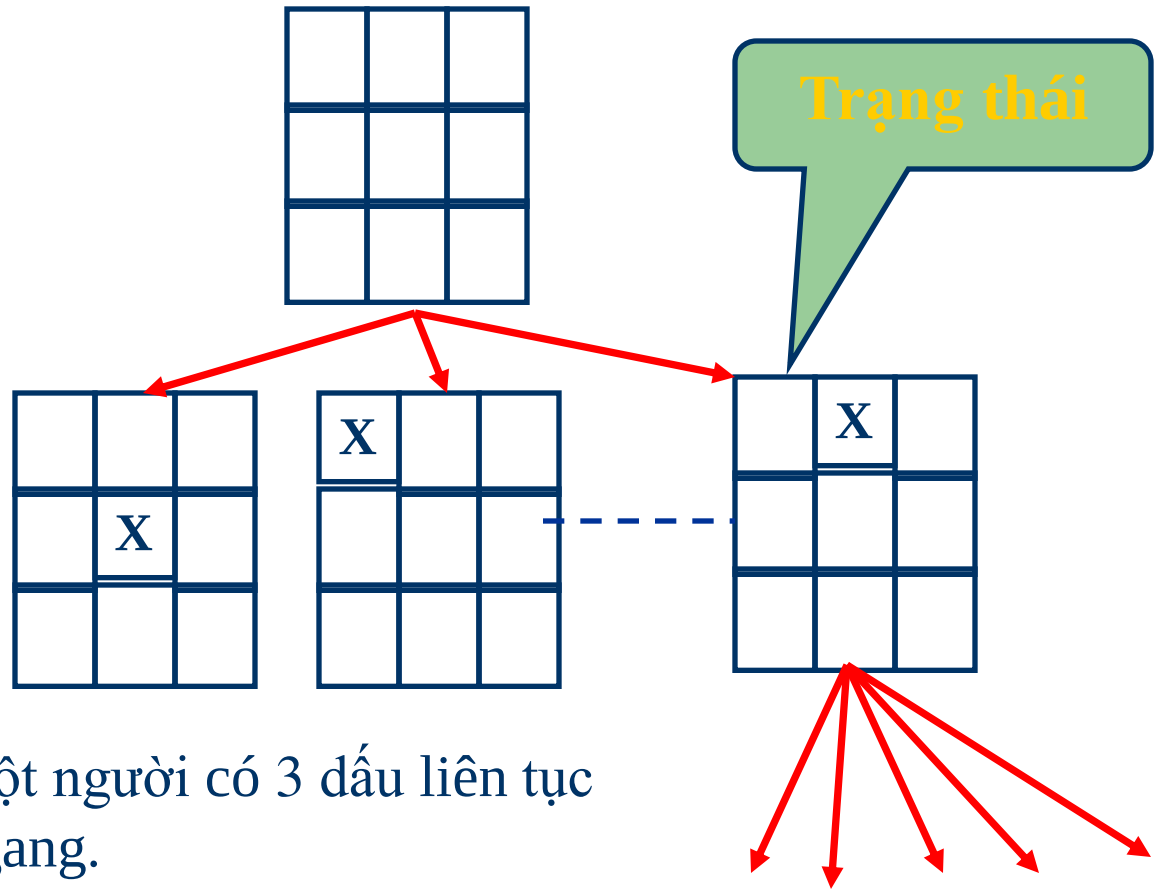
- Trò chơi Tic Tac Toa

▪Trạng thái là một tình huống của bàn cờ

▪ Số trạng thái bùng nổ nhanh.

▪Biểu diễn trạng thái

▪Biểu diễn không gian



Trạng thái kết thúc: có một người có 3 dấu liên tục theo đường chéo, thẳng, ngang.

Số trạng thái kết thúc=???

State Space & Database search

State Space

- Không gian tìm kiếm thường là một graph
- Mục tiêu tìm kiếm là một path
- Phải lưu trữ toàn bộ không gian trong quá trình tìm kiếm
- Không gian tìm kiếm biến động liên tục trong quá trình tìm kiếm
- Đặc tính của trạng thái/nút là phức tạp & biến động

Database

- Không gian tìm kiếm là một list hay tree
- Tìm kiếm một record/nút
- Phần tử đã duyệt qua là không còn dùng tới
- Không gian tìm kiếm là cố định trong quá trình tìm kiếm
- Thuộc tính của một record/nút là cố định

Chiến lược điều khiển trong SSS

- Mục tiêu của bài toán tìm kiến trên không gian trạng thái: **PATH** vs STATE
- Xuất phát từ đâu và kết thúc như thế nào?
- Chiến lược Data-Driven-Search: Quá trình search sẽ đi từ trạng thái hiện thời áp dụng các luật để đi đến trạng thái kế tiếp và cứ thế cho đến khi đạt được một goal.
- Chiến lược Goal-Driven-Search: Quá trình search sẽ đi từ trạng thái hiện tại (goal tạm thời) tìm xem luật nào có thể sinh ra trạng thái này. Các điều kiện để áp dụng được các luật đó trở thành subgoal. Quá trình lặp lại cho đến khi lui về đến các sự kiện ban đầu.

Data-Driven Search hay Goal-Driven Search??

Data-Driven vs Goal-Driven

- Cả hai chiến lược cùng làm việc trên không gian trạng thái nhưng thứ tự và số các sự kiện duyệt qua khác nhau. Do cơ chế sinh ra các state khác nhau.
- Quyết định chọn lựa chiến lược tùy thuộc vào:
  - Độ phức tạp của các luật
  - Độ phân chia của không gian trạng thái
  - Sự hiện hữu của dữ liệu
    - Goal đã có hay chưa, nhiều hay ít
    - Goal được đặc tả như thế nào: state cụ thể hay mô tả mang tính đặc tính
- Cơ sở thông tin để chọn lựa chiến lược hợp lý là một META KNOWLEDGE

Data-Driven vs Goal-Driven – Ví dụ

- Ba và Nam là bà con. Ba hơn nam 250 tuổi. Tìm mối quan hệ giữa Ba và Nam.
- Trong bài toán này:
  - Không gian trạng thái là cây phả hệ
  - Mục tiêu tìm kiếm là path nối Ba với Nam
- Giả sử mỗi thế hệ cách nhau 25 năm, như vậy Ba cách nam 10 thế hệ
- Data-Driven-Search: Tìm từ Ba đến Nam.

  nếu trung bình mỗi thế hệ có X con thì số trạng thái cần xét là X^{10}
- Goal-Driven search: Tìm từ Nam đến Ba

  mỗi người chỉ có 1 cha và 1 mẹ. Số trạng thai cần xét là 2^{10}.

Như vậy Goal-Driven sẽ tốt hơn Data driven nếu số con > 2

Graph Search

- Giải thuật graph search phải có khả năng tìm kiếm ra tất cả các path có thể có để tìm được nghiệm : **PATH từ trạng thái khởi đầu đến goal.**
- Graph search thực hiện bằng cách "lần" theo các nhánh của graph. Từ một trạng thái, sinh ra các trạng thái con, chọn một trạng thái con, xem đó là trạng thái xét kế tiếp. Lặp lại cho đến khi tìm thấy một trạng thái đích.
- "Lần" theo các trạng thái → Đi vào ngõ cụt ?
- Khi gặp nhánh không đi tiếp được, giải thuật phải có khả năng quay lui lại trạng thái trước đó để đi sang nhánh khác: BACK TRACKING. Do đó giải thuật còn có tên là **BACKTRACK** search.

Giải thuật chi tiết

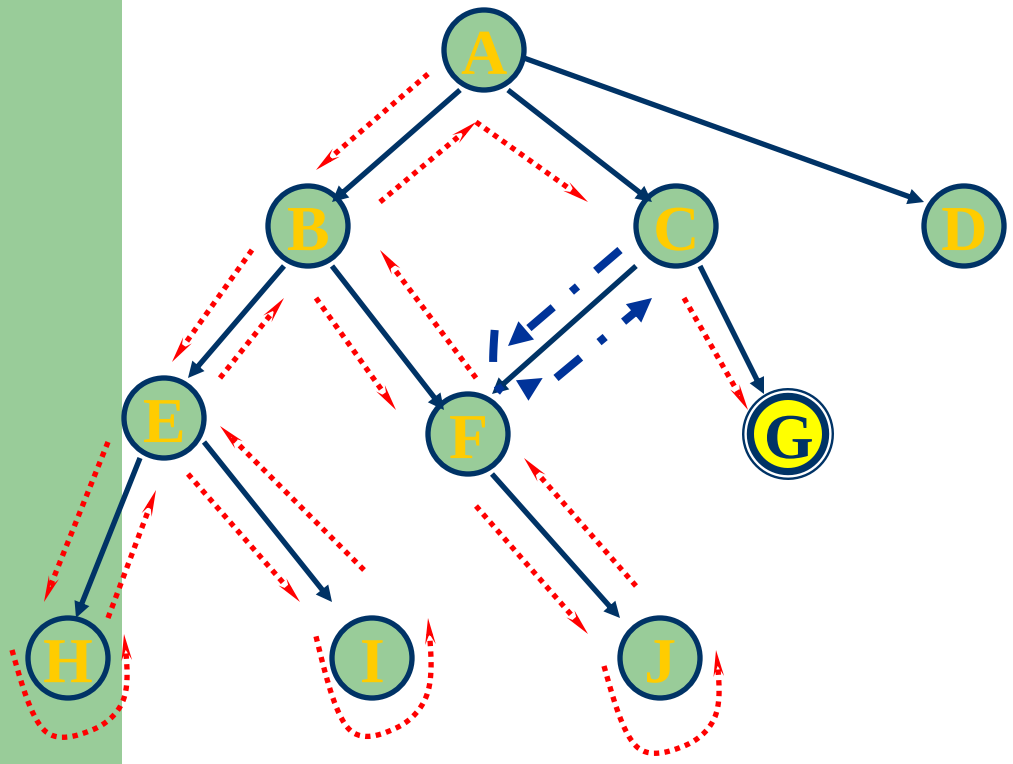
```
Procedure backtrack;
Begin
S:=[start]; NLS:=[start]; De:=[ ];
CS:=start;
While (NSL<>[ ]) do
 Begin
  if CS = Goal then return(SL);
  if CS has no children (Except node in DE,
     Sl and NSL) then
   begin
     while ((SL<>[ ]) and CS=First element
     of SL)) do
    begin add CS to DE
      remove first element from SL;
      remove first element from NSL;
       Cs:= first element of NSL;
     end;
  add CS to SL;
 End;
Else begin
  add children of CS (Except node in
     DE,SL and NSL) to NSL
  CS:= first element of NSL;
  add CS to SL;
      end;
 end;
End; {end while}
Return FAIL;
End;
```

Giải thuật chi tiết (tt)

- Trong đó:
  - SL (State list) : chứa danh sách các trạng thái trên path hiện đang xét. Nếu tìm ra goal thì SL chính là nghiệm.
  - NSL (New State List): chứa danh sách các trạng thái đang đợi xét.
  - DE (Dead End): chứa các trạng thái mà con cháu của chúng không chứa đích.
  - CS (Current State): chứa trạng thái đang xét.
- Hướng phát triển của quá trình search tùy theo cơ cấu tổ chức của NSL: FIFO, FILO hay Evaluated.
- Giải thuật có thể bị loop vô tận. **Lý do????**

Giải thuật chi tiết (tt) – Ví dụ

- Xét graph sau:



S=[A] bắt đầu

GD=[G] là goal. Kết thúc

Cung graph

Đường đi

Quy trình tìm kiếm?

F: đi qua mấy lần ??

Giải thuật chi tiết (tt) – Ví dụ

- Với G là goal ta có kết quả tìm kiếm theo bảng sau:

Laàn laëp	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[B A]	[B C D A]	[]
2	E	[E B A]	[E F B C D A]	[]
3	H	[H E B A]	[H I E F B C D A]	[]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	**[G C A]**	[G C D A]	[B F J E I H]

Breath First Search

```
Procedure Breath_frist_search;
Begin
 open :=[start]; close:=[];
While (open <>[]) do
 begin
  remove X which is the leftmost of Open;
  If (X=goal) the return (Success)
  else begin
     generate children of X; Put X to close;
     eleminate children of X which is in Open or Close;
     Put remain children on RIGHT end of open;
       End;
  End;
Return (FALL);
End;
```

Là graph search với các nút "anh em" của nút hiện thời được xem xét trước các nút "con cháu"

Breath First Search – Ví dụ

- Với đồ thị đã có trong ví dụ graph search.Với Breath first search ta có quá trình như sau:

Laàn laëp	X	Open	Close
0		[A]	[]
1	A	[B C D]	[A]
2	B	[C D E F]	[A B]
3	C	[D E F G]	[A B C]
4	D	[E F G]	[A B C D]
5	E	[F G H I]	[A B C D E]
6	F	[G H I J]	[A B C D E F]
7	**G**	[H I J]	[A B C D E F]

Depth First Search

```
Procedure depth_frist_search;
Begin
 open :=[start]; close:=[];
While (open <>[]) do
 begin
  remove X which is the leftmost of Open;
  If (X=goal) the return (Success)
  else begin
     generate children of X; Put X to close;
     eleminate children of X which is in Open or Close;
     Put remain children on LEFT end of open;
       End;
  End;
Return (FALL);
End;
```

Là graph search với các nút "con cháu" của nút hiện thời được xem xét trước các nút "anh em".

Depth First Search – Ví dụ

- Với đồ thị đã có trong ví dụ graph search.Với Depth First Search ta có quá trình như sau:

Laàn laëp	X	Open	Close
0		[A]	[]
1	A	[B C D]	[A]
2	B	[E F C D]	[A B]
3	E	[H I F C D]	[A B E]
4	H	[I F C D]	[A B E H]
5	I	[F C D]	[A B E H I]
6	F	[J C D]	[A B E H I F]
7	J	[C D]	[A B E H I F J]
8	C	[G D]	[A B E H I F J C]
9	**G**		

Breath First vs Depth First

- Breath First: open được tổ chức dạng FIFO
- Depth First: open được tổ chức dạng LIFO
- Hiệu quả
  - Breath First luôn tìm ra nghiệm có số cung nhỏ nhất
  - Depth First "thường" cho kết quả nhanh hơn.
- Kết quả
  - Breath First search chắc chắn tìm ra kết quả nếu có.
  - Depth First có thể bị lặp vô tận. Tại sao??????
- Bùng nổ tổ hợp là khó khăn lớn nhất cho các giải thuật này.

Giải Pháp cho bùng nổ tổ hợp??

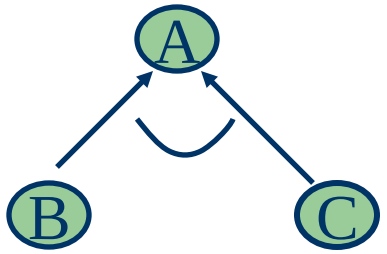
Depth first search có giới hạn

- Depth first search có khả năng lặp vô tận do các trạng thái con sinh ra liên tục. Độ sâu tăng vô tận.
- Khắc phục bằng cách giới hạn độ sâu của giải thuật.

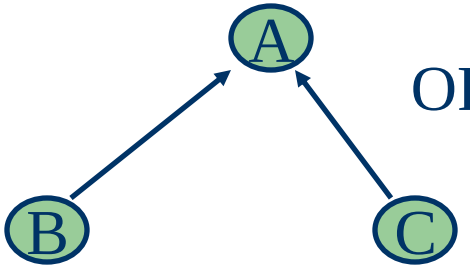
- Sâu bao nhiêu thì vừa?
- Chiến lược giới hạn:
  - Cố định một độ sâu MAX, như các danh thủ chơi cờ tính trước được số nước nhất định
  - Theo cấu hình resource của máy tính
  - Meta knowledge trong việc định giới hạn độ sâu.
- Giới hạn độ sâu => co hẹp không gian trạng thái => có thể mất nghiệm.

AND/OR Graph

- AND/OR graph là một đồ thị với các nút có thể là OR hay AND của các nút con.



AND node



OR node

- Hypergraph: Một cung xác định bởi một cặp 2 phần tử:
  - Phần tử đầu là một node thuộc N.
  - Phần tử sau là một **tập con** của N.
  - Nếu phần tử sau có k node thì ta nói Hypergraph có K-Connector
- AND/OR Graph đòi hỏi lưu trữ nhiều dữ liệu hơn
  - Các node OR kiểm tra như Backtrack Search
  - Các node AND phải kiểm tra đồng thời
  - Phải lưu trữ tất cả các vết đã đi qua để kiểm tra AND/OR

Chương 4: HEURISTIC SEARCH

- Heuristic là gì?
- Tìm kiếm theo heuristic
- Các giải thuật Best first search (BFS), Giải thuật A*
- Chiến lược Minimax, Alpha Beta

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Heuristic

- **Heuristic là gì?**
  - Heuristic là những tri thức được rút tỉa từ những kinh nghiệm, "trực giác" của con người.
  - Heuristic có thể là những tri thức "đúng" hay "sai".
  - Heuristic là những meta knowledge và "thường đúng".
- **Heuristic dùng để làm gì?**

Trong những bài toán tìm kiếm trên không gian trạng thái, có 2 trường hợp cần đến heuristic:

1. Vấn đề có thể không có nghiệm chính xác do các mệnh đề không phát biểu chặt chẽ hay thiếu dữ liệu để khẳng định kết quả.
2. Vấn đề có nghiệm chính xác nhưng phí tổn tính toán để tìm ra nghiệm là quá lớn (hệ quả của bùng nổ tổ hợp)

Heuristic giúp tìm kiếm đạt kết quả với chi phí thấp hơn

Heuristic (tt)

- Heuristic dùng như thế nào trong SSS?
  - Tìm kiếm trên không gian trạng thái theo chiều nào? Sâu hay rộng?
  - Tìm theo Heuristic : Heuristic định hướng quá trình tìm kiếm theo hướng mà "nó" cho rằng khả năng đạt tới nghiệm là cao nhất. Không "sâu" cũng không "rộng"
- Kết quả của tìm kiếm với Heuristic
  - Việc tìm kiếm theo định hướng của heuristic có kết quả tốt hay xấu tùy theo heuristic "đúng" hay "sai".
  - Heuristic có khả năng bỏ xót nghiệm
  - Heuristic càng tốt càng dẫn đến kết quả nhanh và tốt.

Làm sao tìm được Heuristic tốt???

Best First Search

Procedure Best_First_Search;
Begin
open:=[start]; close:=[];
While (open<>[]) **do**
begin Lấy phần tử đầu tiên X khỏi Open.
if X là goal **then** return path từ start đến X
else begin
sinh ra các nút con của X;
for mỗi nút con Y của X **do**
case Y **of**
Y không có trong open hay close:
begin gán giá trị heuristic cho Y;
đưa Y vào open; **end;**

Y đã có trong Open:
if đến được Y bằng một path ngắn hơn **then** gán path ngắn hơn này cho Y trên Open.

Y đã có trên close:
if đến được Y bằng một path ngắn hơn **then begin** xóa Y khỏi danh sách Close;
thêm Y vào danh sách Open; **end;**

end; /*end case*/
Đưa X vào close;
Xếp thứ tự các trạng thái trên Open theo giá trị Heuristic (**tăng dần**)
end; / while/
return failure;
End;

Best First Search (tt)

- Best First search vs Depth First & Breath First
  - Best First search tương tự như Depth First & Breath First nhưng phần tử được xét tiếp theo là phần tử có giá trị heuristic tốt nhất.
  - Cần có một hàm đánh giá các trạng thái để xác định giá trị heuristic cho các trạng thái.
  - Không gian trạng thái vẫn không thay đổi về "toàn cục" tuy nhiên thường Heuristic search có không gian trạng thái làm việc nhỏ hơn Depth First và Breath First. **Tại sao??**
    - Do sự định hướng các trạng thái kế tiếp theo hướng có khả năng tìm ra nghiệm nhanh hơn nên số trạng thái xét dư thừa sẽ hạn chế ➔ sinh ít trạng thái con hơn
    - Điều này cũng là nguyên nhân làøm cho Best First Search có thể dẫn đến kết quả là "nghiêäm phụ" thay vì "nghiệm tối ưu".

Hàm lượng giá Heuristic

- Hàm lượng giá Heuristic là hàm ước lượng phí tổn để đi từ trạng thái hiện tại đến trạng thái goal.
- Cơ sở để xác định hàm lượng giá là dựa vào tri thức/kinh nghiệm thu thập được.
- Hàm lượng giá cho kết quả đúng (gần thực thế) hay sai (xa giá trị thực) sẽ dẫn đến kết quả tìm được tốt hay xấu.
- Không có chuẩn mực cho việc đánh giá một hàm lượng giá Heuristic. Lý do:
  - Không có cấu trúc chung cho hàm lượng giá
  - Tính đúng/sai thay đổi liên tục theo từng vấn đề cụ thể
  - Tính đúng/sai thay đổi theo từng tình huống cụ thể trong một vấn đề
- **Có thể dùng nhiều hàm lượng giá khác nhau theo tình huống ➔ cần hàm lượng giá về các hàm lượng giá.**

Hàm lượng giá Heuristic – Ví dụ

- Xét bài toán 8 pussle với goal là:

1	2	3
8		4
7	6	5

Heuristic 1: Tổng số miếng sai vị trí

Heuristic 2: Tổng khoảng cách sai vị trí của từng miếng.

Heuristic 3: Số cặp hoán đổi vị trí nhân cho 2

2 8 3 1 6 4 7 5	5	6	0
2 8 3 1 4 7 6 5	3	4	0
2 8 3 1 6 4 7 5	5	6	0

Việc chọn lựa hàm Heuristic là khó khăn và có ý nghĩa quyết định đối với tốc độ của giải thuật

Hàm lượng giá Heuristic – Cấu trúc

- Xét lại hoạt động của giải thuật Best First Search:
  - Khi có 2 nút cùng có giá trị kỳ vọng đạt đến mục tiêu bằng nhau thì nút có path từ nút bắt đầu đến nút đó ngắn hơn sẽ được chọn trước như vậy nút này có giá trị Heuristic tốt hơn.
  - Hay nói cách khác hàm lượng giá Heuristic cho nút gần start hơn là tốt hơn nếu kỳ vọng đến goal là bằng nhau.
  - Vậy chọn nút nào nếu kỳ vọng của 2 nút khác nhau? Nút kỳ vọng tốt hơn nhưng xa start hay nút kỳ vọng xấu hơn nhưng gần root
- Hàm lượng giá bao gồm cả 2 và có cấu trúc:

F(n) := G(n) + H(n)

G(n): phí tổn thực từ root đến n
H(n): phí tổn ước lượng heuristic từ n đến goal.

Ví dụ – Best first search

- Xét ví dụ là bài toán 8 puzzle với:

2	8	3
1	6	4
7		5

Bắt đầu

1	2	3
8		4
7	6	5

Mục tiêu

- Hàm lượng giá: F(n) = G(n) + H(n)

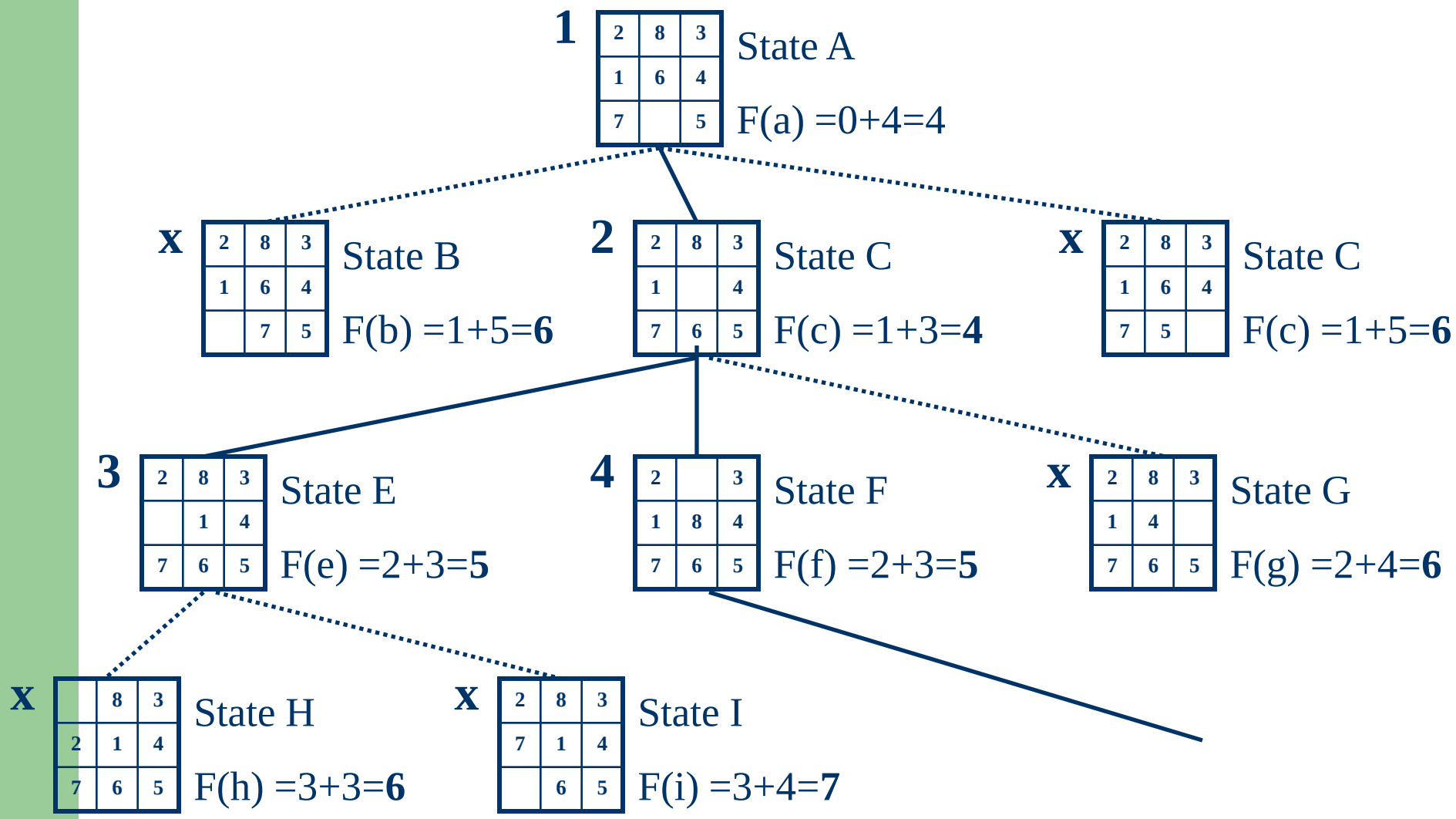
Với G(n): số lần chuyển vị trí tile đã thực hiện

H(n): Số tile nằm sai vị trí

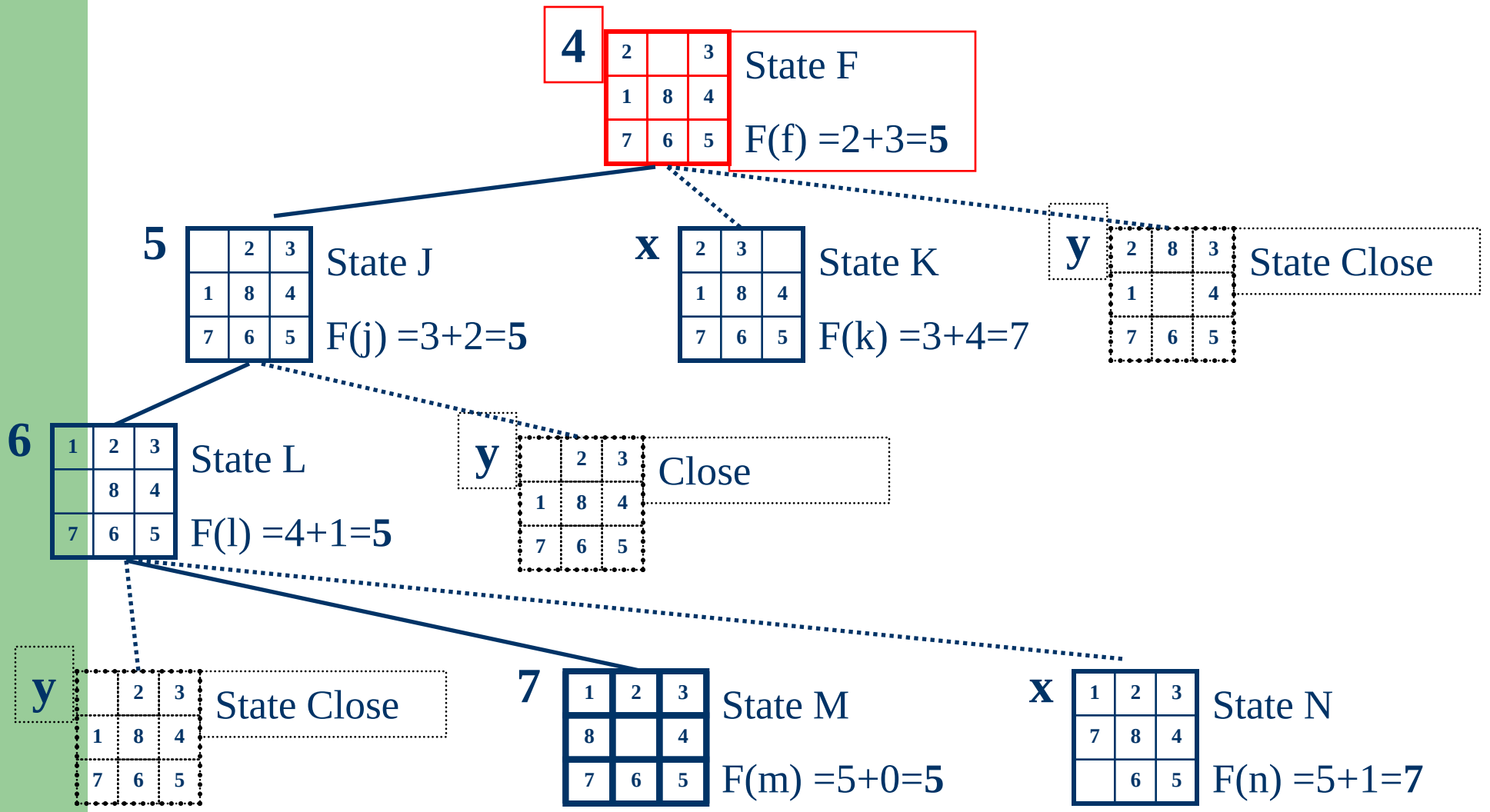
Nút X có giá trị heuristic tốt hơn nút Y nếu **F(x) < F(y)**.

Ta có hoạt động của giải thuật Best First search trên như hình sau:

Ví dụ – Best first search (tt)



Ví dụ – Best first search (tt)



Hoạt động theo giải thuật Best First Search

Laàn	X	Open	Close
0		[a4]	[]
1	A4	[c4,b6,d6]	[a4]
2	C4	[e5,f5,g6,b6,d6]	[a4,c4]
3	E5	[f5,h6,g6,b6,d6,i7]	[a4,c4,e5]
4	F5	[j5,h6,g6,b6,d6,k7,i7]	[a4,c4,e5,f5]
5	J5	[l5,h6,g6,b6,d6,k7,i7]	[a4,c4,e5,f5,j5]
6	l5	[m5,h6,g6,b6,d6,k7,i7,n7]	[a4,c4,e5,f5,j5,l5]
7	m5		

Đánh giá giải thuật Heuristic

- **Admissibility** – Tính chấp nhận

  Một giải thuật Best first search với hàm đánh giá

  F(n) = G(n) + H(n) với

  N : Trạng thái bất kỳ

  G(n) : Phí tổn đi từ nút bắt đầu đến nút n

  H(n) : Phí tổn ước lượng heuristic đi từ nút **n** đến **goal**

  Được gọi là giải thuật **A**

Một giải thuật tìm kiếm được xem là admissible nếu đối với một đồ thị bất kỳ nó luôn dừng ở path nghiệm tốt nhất (nếu có).

Giải thuật A*: Là giải thuật A với hàm heuristic H(n)luôn luôn ≤ giá trị thực đi từ n đến goal.

Giải thuật A* là admissible

Đánh giá giải thuật Heuristic

- **Monotonicity** – Đơn điệu

Một hàm heuristic H(n) được gọi là monotone (đơn điệu) nếu:

1. ∀ni, nj : nj là nút con cháu của ni ta có H(ni)-H(nj) ≤ phí tổn thật đi từ ni đến nj
2. Đánh giá heuristic của đích là 0 : H(goal) = 0.

Giải thuật A có hàm H(n) monotone là giải thuật A* và Admissible

- **Informedness**

Xét 2 hàm heuristic $H_1(n)$ và $H_2(n)$ nếu ta có $H_1(n) \leq H_2(n)$ với mọi trạng thái n thì $H_2(n)$ được cho là informed hơn $H_1(n)$.

Chiến lược minimax

- Giải thuật tìm kiếm Heuristic với các hàm heuristic chỉ thích hợp cho các bài toán không có tính đối kháng. Như các trò chơi chỉ có một người chơi: Puzzle, tìm lối ra mê cung, bài toán n quân hậu,…
- Các trò chơi có tính đối kháng cao, thường là các trò chơi 2 người chơi như: tic tac toa, caro, cờ quốc tế,… giải thuật trên không có tác dụng vì: Đối phương không bao giờ đi theo con đường cho ta có thể đi đến goal
- Cần phải có một giải thuật khác phù hợp hơn.

Chiến lược MINIMAX

- Chiến lược Minimax (được thể hiện bằng giải thuật minimax) dựa trên 2 giả thiết sau:
  - Cả 2 đối thủ có cùng kiến thức như nhau về không gian trạng thái của trò chơi
  - Cả 2 đối thủ có cùng mức cố gắng thắng như nhau

Giải thuật minimax

- Chiến lược Minimax

Hai đối thủ trong trò chơi có tên là **MAX** và **MIN**

- Max: biểu diễn cho mục đích của đối thủ này là làm lớn tối đa lợi thế của mình
- Min: biểu diễn cho mục đích của đối thủ này là làm nhỏ tối đa lợi thế của đối phương.

Trên cây tìm kiếm sẽ phân lớp thành các lớp Max và Min.

Với một node **n** bất kỳ,

- Nếu nó thuộc lớp Max thì gán cho nó giá trị Max của các node con
- Nếu nó thuộc lớp Min thì gán cho nó giá trị nhỏ nhất của các node con.

Giải thuật minimax – ví dụ

- Bài toán que diêm

  Một tập que diêm ban đầu đặt giữa 2 người chơi. Lần lượt đi xen kẽ. Người đến lượt đi phải chia nhóm que diêm theo nguyên tắc:

  – Chọn nhóm bất kỳ có số que >2

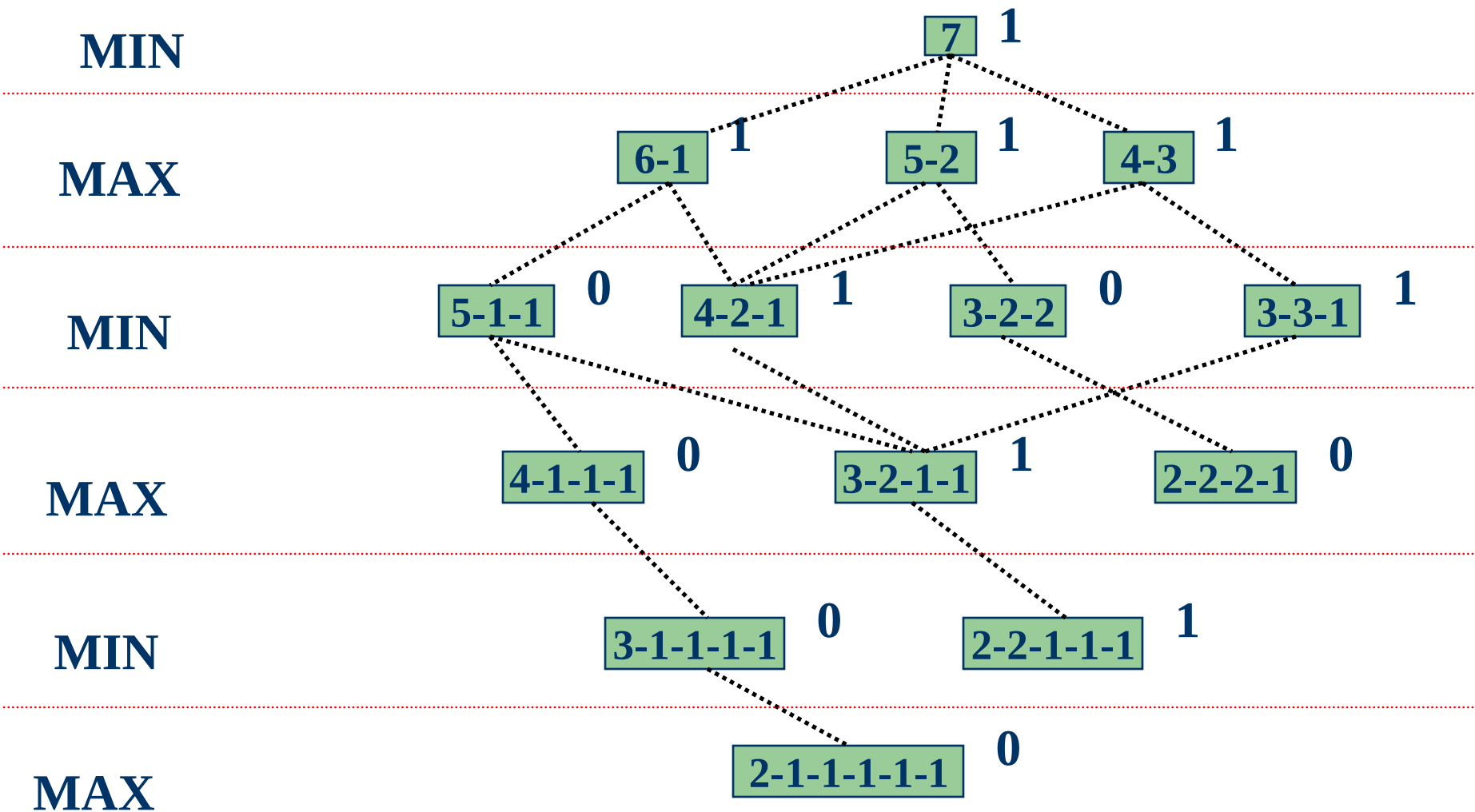
  – Chia thành 2 nhóm có số que khác nhau

Goal: người nào đến lượt mà không chia được là thua.

MINIMAX

- Không gian trạng thái của trò chơi được phát triển toàn bộ, các node lá được gán giá trị 1 nếu là MAX thắng và 0 nếu là MIN thắng.
- Với một node bất kỳ nếu thuộc lớp MAX gán cho nó giá trị lớn nhất của các node con. Nếu thuộc lớp MIN gán cho nó giá trị nhỏ nhất của các node con.

Minimax – bài toán que diêm



Minimax với độ sâu giới hạn

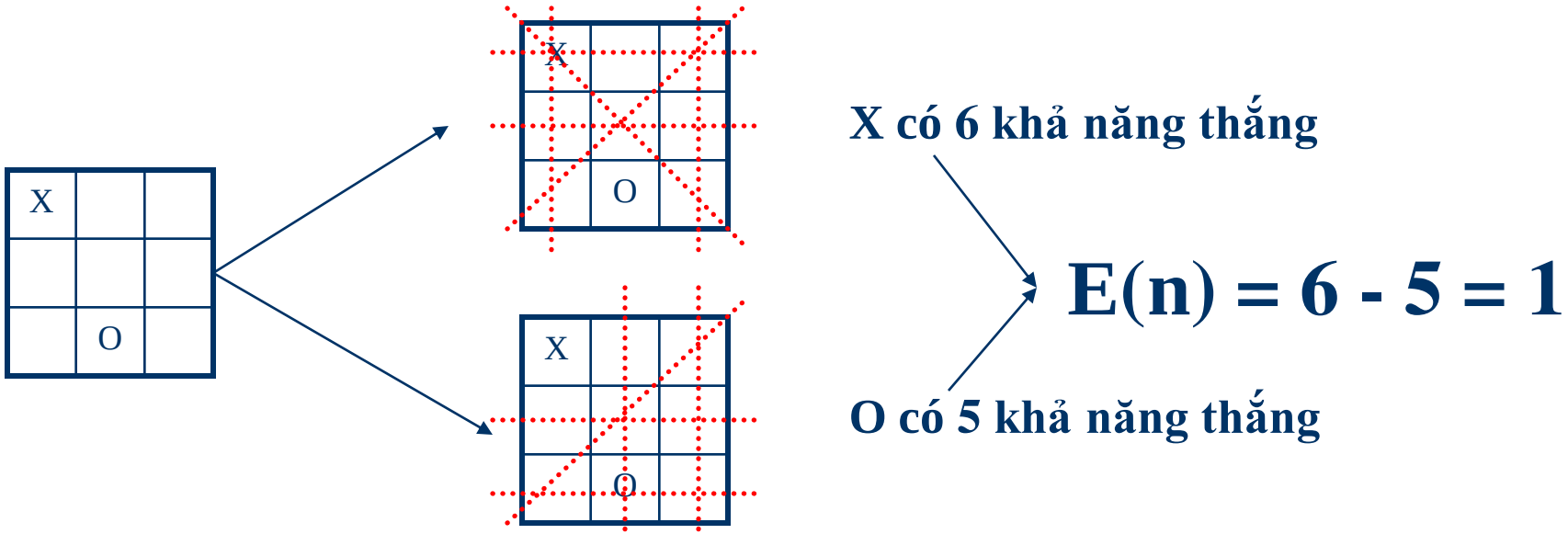
- Minimax như đã xét buộc phải có toàn bộ không gian trạng thái đã được triển khai để có thể gán trị cho các nút lá và tính ngược lại ➔ Không khả thi với các bài toán lớn vì không gian trạng thái là quá lớn.

➔ Giới hạn không gian trạng thái lại theo một độ sâu nào đó và giới hạn các node con theo một qui tắc nào đó.

- Đây là chiến lược thông thường của các người chơi cờ: khả năng tính trước bao nhiêu nước.
- Khi đó ta chỉ triển khai các nút con đến độ sâu giới hạn.
- Đánh giá cho các nút này như là nút lá bằng một hàm lượng giá Heuristic.
- áp dụng chiến lược minimax cho việc đánh giá các nút cấp trên.
- Kỹ thuật này gọi là nhìn trước K bước với K la øđộ sâu giới hạn.

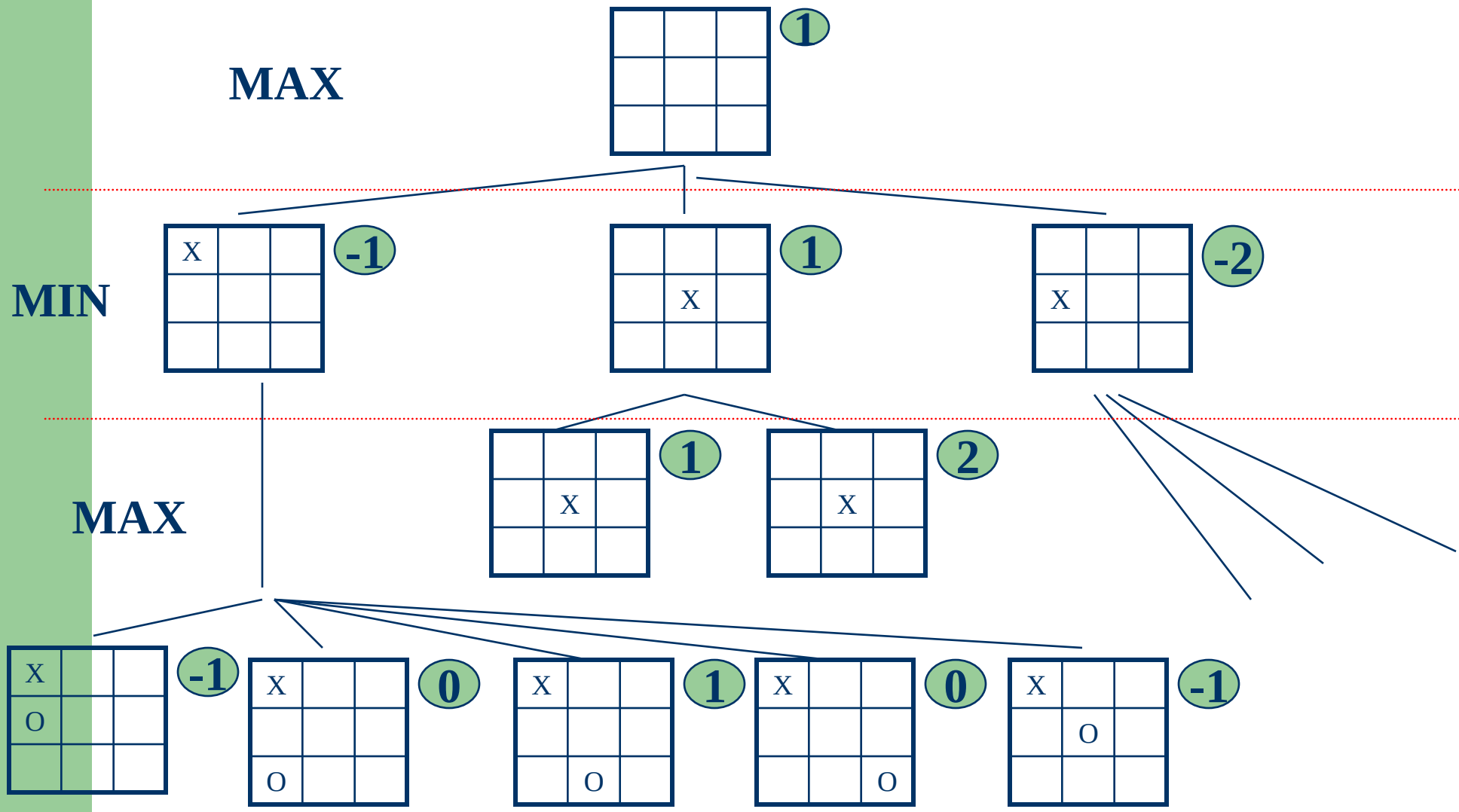
Ví dụ: Bài toán Tic Tac Toa

- Hàm lượng giá heuristic E(n) = X(n) – O(n) với
  - X(n) số khả năng thắng của quân X.
  - O(n) số khả năng thắng của quân O



Với hàm Heuristic trên X sẽ cố làm cho E(n) lớn nhất (MAX) và O làm cho E(n) nhỏ nhất (MIN). Triển khai bài toán với 2 bước nhìn trước.

Ví dụ: Bài toán Tic Tac Toa



Độ phức tạp của giải thuật SSS

- Độ phức tạp tính theo hệ số rẽ nhánh.
- Xét bài toán có hệ số rẽ nhánh trung bình là B, độ sâu trung bình của solution path là D . Với T là số trạng thái đã được xét qua để tìm ra lới giải thì ta có

$$T= (B +B^2 + B^{2 +}.....^{+}BD)/(B-1)$$

T sẽ rất lớn đối với các vấn đề thực tế.

Phải dùng Heuristic để giới hạn độ phức tạp của giải thuật bằng cách giảm số trạng thái phải đi qua.

Tuy nhiên hàm Heuristic tốt thì lại đòi hỏi yêu cầu tính toán nhiều ➔ Phí tổn cho tính toán tăng cao.

Chương 5: HỆ LUẬT SINH

Production system

- Tìm kiếm đệ qui
- Hệ luật sinh: Định nghĩa và ứng dụng
- Tìm kiếm trên hệ luật sinh

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Đặc tính dữ liệu và điều khiển của SSS

- Các đặc tính của giải thuật SSS:
  - Lời giải là một PATH từ điểm START đến điểm GOAL
  - Tìm kiếm là sự kiểm tra có hệ thống các đường dẫn đến GOAL
  - Backtracking cho phép giải thuật "phục hồi" khi đi vào một nhánh không có đáp án.
  - Các danh sách sẽ giữ các trạng thái đang xem xét:
    - Danh sách Open: cho phép hệ thống backtrack về các trạng thái chưa được xét.
    - Danh sách Close: cho phép hệ thống kiểm tra sự quay vòng tránh lặp vô tận
  - Dùng STACK cho DFS, QUEUE cho BFS và dùng PRIORITY QUEUE cho BFS.

Tìm kiếm đệ qui – Tại sao???

- Tìm kiếm đệ qui là giải thuật tìm kiếm trên SSS với các đặc tính:
  - Ngắn gọn xúc tích hơn
  - Tiếp cận của giải thuật tự nhiên hơn
  - Hợp nhất với phương thức hiện thực của Logic vị từ
- Các giải thuật tìm kiếm đệ qui chính:
  - Tìm kiếm đệ qui – Recursive Search (RS)
  - Tìm kiếm theo mẫu – Pattern Directed Search (PDS)
- Các giải thuật tìm kiếm đệ qui được sử dụng rộng rải trong các shell của các Hệ chuyên gia (Expert System).
- Pattern Directed Search là nền tản của PROLOG

Thế nào là đệ qui?

- Đệ qui là sự định nghĩa một đối tượng bằng cách sử dụng chính đối tượng đó – Toán học
- Đệ qui được dùng để định nghĩa và phân tích các cấu trúc dữ liệu cũng như các thủ tục xử lý trong ngành máy tính.
- Một thủ tục đệ qui bao gồm:
  - Thành phần đệ qui, trong đó thủ tục gọi chính nó để lặp lại chuỗi các thao tác.
  - Thành phần dừng dùng để dừng quá trình đệ qui vô tận. (lặp vô tận)
- Hai thành phần này tồn tại đồng thời trong tất cả các định nghĩa đệ qui cũng như giải thuật đệ qui.
- Đệ qui là một cấu trúc điều khiển dữ liệu tự nhiên cho những cấu trúc không xác định số phần tử cố định: list, tree, và đồ thị.

Thủ tục đệ qui – ví dụ

```
Function Member(item, list);
 begin
  if List rỗng then return (Fail)
  else
   if Item = phần tử đầu của list then return
      (succes)
     else
      begin
       Tail:= List \ phần tử đầu;
        member (item, Tail);
      end
 end;
```

- Đệ qui có đầy đủ tính năng của các cấu trúc điều khiển truyền thống như Loop và rẽ nhánh➔mọi chương trình viết được bằng cấu trúc truyền thống đều có thể viết đệ qui.
- Đệ qui thích hợp biểu diễn các cấu trúc toán học ➔ thuận tiện trong việc kiểm tra tính đúng đắn của giải thuật đệ qui.
- Công thức đệ qui cũng thường được dùng trong việc sinh và kiểm tra chương trình tự động.
- Đệ qui là công cụ tự nhiên và mạnh mẻ cho hiện thực các chiến lược giải quyết vấn đề của AI.

Tại sao?

Giải thuật DFS đệ qui - DFS

```
Function Depth_First_Search;
Begin
 if Open rỗng then return (fail);
Current_state := phần tử đầu tiên của open;
If (current_state là mục tiêu) then return (Success)
else begin
  open:=phần đuôi của open;
 Closed := Closed + current_state;
 for mỗi phần tử con Y của current_state do
  if not (Y  in close) and not (Y in open) then thêm Y vào đầu của Open;
End;
depth_first_search;
End;
```

Nhược điểm của recursive ?

Pattern-Directed Search (PDS)

- Các giải thuật Search đã tìm hiểu và Recursive Seach không trình bày cách biểu diễn một trạng thái trong không gian trạng thái cũng như cách sinh các trạng thái mới.
- Pattern-Directed Search là một giải thuật search đệ quy dùng Logic Vị từ để hiện thực việc sinh các trạng thái mới.
- Paterm-Directed Search xuất phát từ goal và các modus ponen dạng q(x)->p(x) để chuyển trạng thái. Các modus ponen này gọi là các luật sinh.
- Giải thuật: Xuất phát từ goal P, áp dụng một giải thuật để tìm các rule với P ở vế phải , sau đó xem vế trái Q là subgoal. Đệ quy với Q cho đến khi Qx là một sự kiện trong kho tri thức.

Sự kiện (FACT) trong kho tri thức?????

Giải thuật PDS

Function Pattern_search(current_goal);
Begin
If current_goal có trong closed then retuen fail
 else thêm current_goal vào trong closed;
 while còn trong database các rule hay fact chưa xét
 begin case
  current_goal trùng với fact:
   return(success);
  current_goal là một phép hội:
   begin
    for mỗi thành phần hội Pi do
     pattern_search(Pi);
If tất cả các hội đều success
 then return success else return fail.
 end;
Current_goal là vế phải một rule:
 begin
  áp dụng các thành phần vào vế trái Q.
  if pattern_search(Q) then return success else return fail;
 end;
end; /* case
Return fail;
End;

Giải thuật PDS

- PDS dùng các rule và thành phần hội để sinh các trạng thái con.
- Tách bạch quá trình điều khiển của giải thuật và dữ liệu của giải thuật
  - ➔ Cùng giải thuật chỉ cần thay đổi database : Fact & Rule ta sẽ áp dụng cho các bài toán khác nhau.
  - ➔Xây dựng các shell và có thể vận hành cho các hệ thống khác nhau bằng cách thay đổi Database.
- Để đơn giản hoá giải thuật chưa giải quyết ở mức độ có các biến trong các rule. Ví dụ P(x)^Q(x) chỉ thõa khi P và Q cùng thỏa với cùng giá trị X.
- Các phép ¬, v,.. cũng chưa giải quyết trong giải thuật này.

Ví dụ: Bài toán mã đi tuần

Đặc tả bài toán: tìm đường đi cho con mã qua tất cả các ô trên bàn cờ. Ví dụ với bàn cờ 3x3.

move(1,8)	move(1,6)	move(2,9)	move(2,7)
move(3,4)	move(3,8)	move(4,9)	move(4,3)
move(6,1)	move(6,7)	move(7,2)	move(7,6)
move(8,3)	move(8,1)	move(9,2)	move(9,4)

1	2	3
4	5	6
7	8	9

∀X path(X,X);

∀X,Y [path(X,Y) ← ∃Z[move(X,Z]^path(Z,Y)]]

Giải thuật PDS đầy đủ

```
Function Pattern_search(current_goal);
Begin
If current_goal có trong Closed then return fail else
     thêm current_goal vào Closed;
while còn các rule hay fact chưa xét
 begin
 case
 current_goal trùng với fact:     return(success);
 current_goal là negated(¬p):
  begin
     if pattern_search(p) then return fail
      else return{}
  end;
 current_goal là một phép hội:
     begin
      for mỗi thành phần hội Pi do
      if not (pattern_search(Pi)) the return fail else
     thay thế tất cả các biến cho các thành phần hội
     khác.
      If tất cả các hội đều success then return các
       thành phần hội else return fail.
    end;
 current_goal là phép tuyển:
  begin
   repeat cho mọi thành phần tuyển Vi;
   until (pattern_search(Vi) or (hết thành phần hội)
   if pattern_search (Vi) then return các thay thế
     else return fail;
 end;
Current_goal là vế phải một rule:
 begin
     áp dụng các thành phần vào vế trái Q.
     if pattern_search(Q) then return kết hợp của
     Current_goal và các thay thế của Q else return
     fail;
end;   end;  /* case*/
return fail;   End;
```

Hệ Luật Sinh – Production System

- **Khái niệm**: Hệ luật sinh là một mô hình tính toán quan trọng trong các bài toán tìm kiếm cũng như mô phỏng cách giải quyết vấn đề của con người trong lĩnh vực ứng dụng AI.
- **Định nghĩa**: Hệ luật sinh là một mô hình tính toán cung cấp cơ chế điều khiển Pattern_directed trong quá trình giải quyết vấn đề (Proplem solving process).

- **Cấu trúc hệ luật sinh bao gồm 3 thành phần**:
  1. Production rules (Tập luật sản sinh)
  2. Working memory (Vùng nhớ làm việc)
  3. Recognize-action control (Bộ điều khiển nhận dạng và hành động)

Định nghĩa (tt) – Production Rule

- **Production rules**: là một tập các luật sản sinh được đặc tả dạng:

  Condition – Action (điều kiện – hành động)

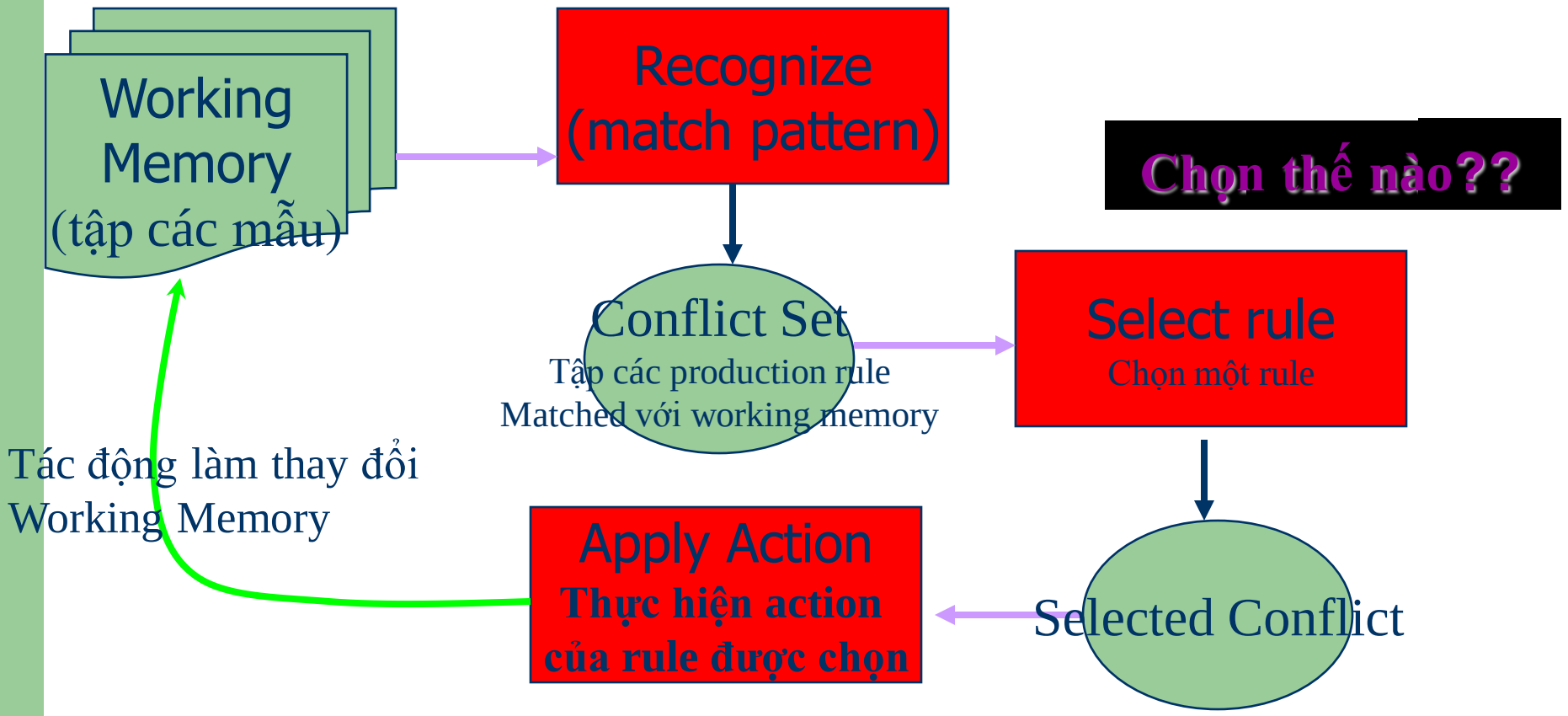
- Một luật là một mắt xích của kho tri thức giải quyết vấn đề. Kho tri thức là một database của các production rules.
- Thành phần Condition: là một mẫu (pattern) dùng xác định điều kiện áp dụng của rule cho một vấn đề tương ứng.
- Thành phần action: mô tả bước giải quyết vấn đề tương ứng sẽ được thực hiện. Đây là phần sẽ tác động lên hiện trạng của không gian tìm kiếm.

Định nghĩa (tt) – Working memory

- **Working memory** chứa những đặc tả trạng thái hiện tại của quá trình suy luận. Chúng được lưu trữ như là tập các mẫu.
- Những đặc tả này là các mẫu để so trùng với các condition của các production rules.
- Khi một production rule được so trùng phần condition thì phần action của nó có thể được áp dụng, và phần action này được xây dựng đặc thù để tác động trực tiếp lên working memory.
- Working memory được khởi tạo bằng trạng thái bắt đầu của vấn đề cần giải quyết.
- Working memory diễn tả hiện trạng của vấn đề cần suy luận

Định nghĩa (tt) – Recognize-action

- Đây là cấu trúc điều khiển dùng trong production system. Quy trình hoạt động của Recognize -Action



Các vấn đề khác

- Chọn lựa Conflict để thực hiện

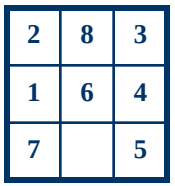
  Có thể chọn bằng cách đơn giản hay áp dụng các chiến lược lựa chọn heurictic → Ứng dụng meta knowledge.

Nếu gọi chiến lược heuristic chọn conflict là meta knowledge thì knowledge "thường" ở đâu trong hệ thống?

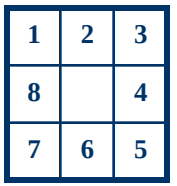
- Các hệ luật sinh đơn thuần không cung cấp cơ chế để quay lui khi việc áp dụng các action làm cho working memory thay đổi dẫn đến lúc không còn production rule nào có thể áp dụng được→ Hệ thống sẽ dừng.
- Cần cung cấp cơ chế backtracking (Thường dùng UNDO). Tuy nhiên cần chú ý để tránh lặp vòng.

Ví dụ: bài toán 8 Puzzle

- **Working Memory**



Current state



Goal state

- **Production Rules**

Goal state	→ Halt
Blank is not on top edge	→ Move the blank up
Blank is not on the right edge	→ Move the blank right
Blank is not on the bottom edge	→ Move the blank down
Blank is not on the left edge	→ Move the left down

- **Recognize-Action**

1- Try each production rule in order

2- Do not allow loops

3- Stop when goal is found

Điều khiển tìm kiếm trong hệ luật sinh

- Chiến lược Data-Driven / Goal-Driven:
  - Data-Driven: bắt đầu với problem trong working memory, matching các condition trong production rule → conflict → áp dụng các action → thay đổi working memory. Lặp lại cho đến khi đạt được goal state.
  - Goal-Driven: Bắt đầu với mô tả goal trong working memory, matching với các kết quả của Action → sinh tập các condition→ đưa các condition vào trong working memory. Lặp lại cho đến khi working memory chứa FACT.
- Điều khiển qua cấu trúc rule: Dùng các biến đổi tương đương của các biểu thức trong rule để điều khiển quá trình tìm kiếm.
- Điều khiển bằng sự phân tích các Conflict: Các conflict có thể được chọn lựa thông qua các Heuristic. áp dụng các meta knowledge trong việc chọn conflict. Ví dụ:

1) Refraction: **Khi một rule đã được áp dụng, nó sẽ không được dùng nữa cho đến khi thánh phần trùng lặp với rule cũ trong working memory được thay đỗi.**
2) Recency: **Chọn rule có phần condition match với phần mới thêm vào working memory. Theo đuổi một hướng triển khai.**
3) Specificity: **Rule nào càng được đặc tả chi tiết càng được ưu tiên cao.**

Các ưu điểm của Hệ luật sinh

- Hệ luật sinh là khung làm việc tổng quát để thực thi các giải thuật tìm kiếm. Với đặc tính đơn giản, dễ sửa đổi, và linh động, hệ luật sinh được dùng như một công cụ quan trọng để xây dựng các hệ chuyên gia và các ứng dụng AI khác
- Các ưu điểm của Hệ luật sinh:
  - Tách bạch giữa Tri thức & Điều khiển:
    - Điều khiển: nằm trong chu trình Recognize-Action
    - Tri thức: được chứa đựng trong bản thân các luật sinh.
    - ➔ Cung cấp khả năng cập nhật tri thức mà không cần điều chỉnh chương trình. Thay đổi mã chương trình mà không ảnh hưởng đến tập luật sinh.
  - Dễ dàng áp dụng trong tìm kiếm trên không gian trạng thái. Các state của working memory là các node. Các production rule là các chuyển đổi giữa các trạng thái (cơ chế sinh các trạng thái mới)
  - Tính độc lập của các luật sinh.
  - Khả năng áp dụng heuristic cho việc điều khiển quá trình hoạt động.
  - Theo dõi và giải thích quá trình hoạt động
  - Độc lập với ngôn ngữ & có thể dùng như kỹ thuật mô phỏng giải pháp của người.

Chương 6: HỆ CHUYÊN GIA (ES)

- Giới thiệu về hệ chuyên gia
- Mô hình hệ chuyên gia: dự trên luật, dựa trên frame
- Phát triển một hệ chuyên gia

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Nội dung.

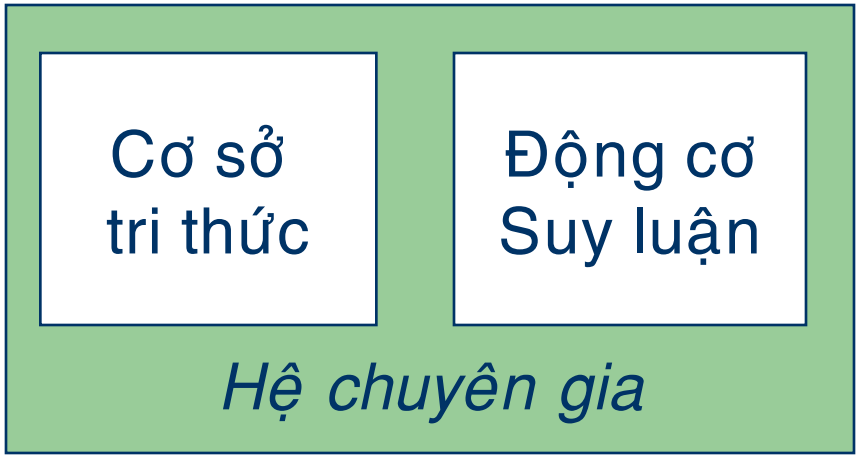
- Giới thiệu về hệ chuyên gia.
  - Định nghĩa, khả năng ứng dụng.
  - Cấu trúc, các đặc trưng cơ bản của ES.
  - Biểu diễn tri thức.
  - Các kỹ thuật suy luận.
- Khảo sát một vài hệ chuyên gia đã có.
  - XCON: ES trợ giúp cấu hình hệ thống máy tính của DEC
  - MYCIN: ES chuẩn đoán bệnh nhiễm trùng máu.
- Hệ chuyên gia dựa trên luật.
  - Kiến trúc, thiết kế.
  - Ưu - nhược điểm.
- Hệ chuyên gia dựa vào Frame.
  - Kiến trúc, thiết kế.
  - Ưu - nhược điểm.

Giới thiệu về hệ chuyên gia.

- Định nghĩa:

  Hệ chuyên gia là một chương trình được thiết kế để theo mô hình có khả năng giải quyết vấn đề của chuyên gia con người.

- Sơ đồ khối cơ bản:



Giới thiệu về hệ chuyên gia.

- Cơ sở tri thức:

  ♣ Dùng để chứa tri thức trong một lĩnh vực nào đó, tri thức này do chuyên gia con người chuyển giao.

  ♣ Nó bao gồm: các khái niệm cơ bản, các sự kiện, các luật và quan hệ giữa chúng.

  Ví dụ:

- Tri thức về *bệnh nhiễm trùng máu* do các bác sĩ chuyên khoa này chuyển giao.
- Tri thức về *chiến lược đầu tư* do các nhà cố vấn đầu tư chuyển giao.
- Tri thức về sự diễn dịch dữ liệu khảo sát địa vật lý do các kỹ sư địa chất chuyển giao.
- …?

Giới thiệu về hệ chuyên gia.

- Động cơ suy luận:

  Là *bộ xử lý* cho tri thức, được mô hình sao cho giống với việc suy luận của chuyên gia con người. Bộ xử lý này làm việc dựa trên thông tin mà người dùng mô tả về vấn đề, kết hợp với CSTT, cho ra kết luận hay đề nghị.

- Tạo sao phải xây dựng ES ?

  Chuyên gia con người là tài nguyên quý giá cho nhiều tổ chức. Họ có thể giải quyết những vấn đề khó, hiệu quả,…. Vậy có giá trị không khi chúng ta xây dựng một chương trình có khả năng như chuyên gia con người ?Một số mặt nào đó còn có thể hơn hẳn. Xem bảng so sánh sau:

Giới thiệu về hệ chuyên gia.

- Bảng so sánh:

Tiêu chí	CG con người	ES.
1. Sẵng dùng	TG. hành chính Mọi lúc.	
2. Vị trí	Cục bộ	Mọi nơi.
3. An toàn	không thể thay thế	Có thể thay thế.
4. Có thể chết	Có	Không.
5. Hiệu suất	Thay đổi	Hằng số.
6. Tốc độ	Thay đổi	Hằng số (thường nhanh hơn)
7. Chi phí	Cao	Có thể cố gắng.

Giới thiệu về hệ chuyên gia.

- Vài lý do để phát triển ES thay cho chuyên gia con người:
  - Tạo cho tính chuyên gia sẵn dùng ở mọi nơi, mọi lúc.
  - Tự động hoá các công việc đòi hỏi chuyên gia.
  - Các chuyên gia đang nghỉ hưu hay chuyển đến nơi khác – Cần thay thế.
  - Thuê chuyên gia với chi phí quá lớn.
  - Tính chuyên gia cần thiết trong các môi trường làm việc không thân thiện, ở đó hỏi một ES sẽ nhanh hơn một chuyên gia con người.
  - Phát triển ES để trợ giúp cho chuyên gia con người.

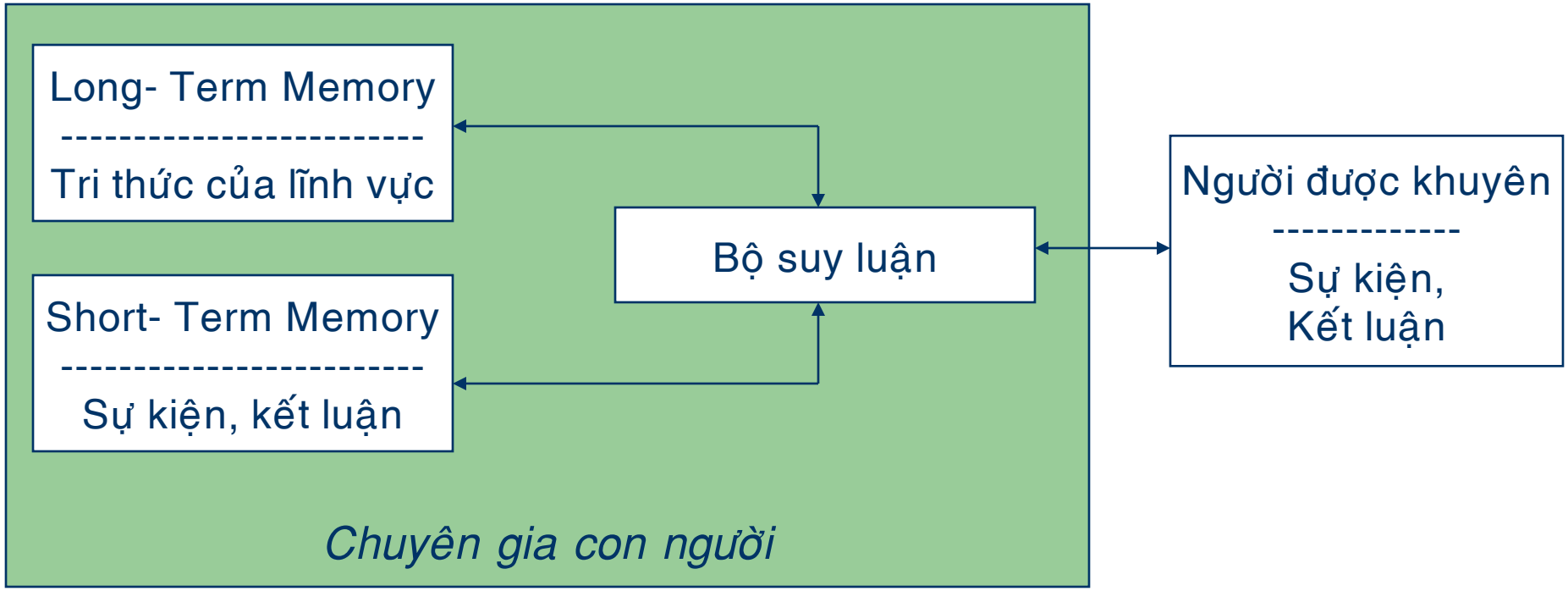
Giới thiệu về hệ chuyên gia.

- Các kiểu vấn đề thường được giải quyết bởi ES:
  - ♣ Điều khiển:
  - ♣ Thiết kế:
  - ♣ Chuẩn đoán:
  - ♣ Dạy học:
  - ♣ Diễn dịch:
  - ♣ Giám sát:
  - ♣ Hoạch định:
  - ♣ Dự đoán:
  - ♣ Lựa chọn:
  - ♣ Mô phỏng:

Cấu trúc của ES.

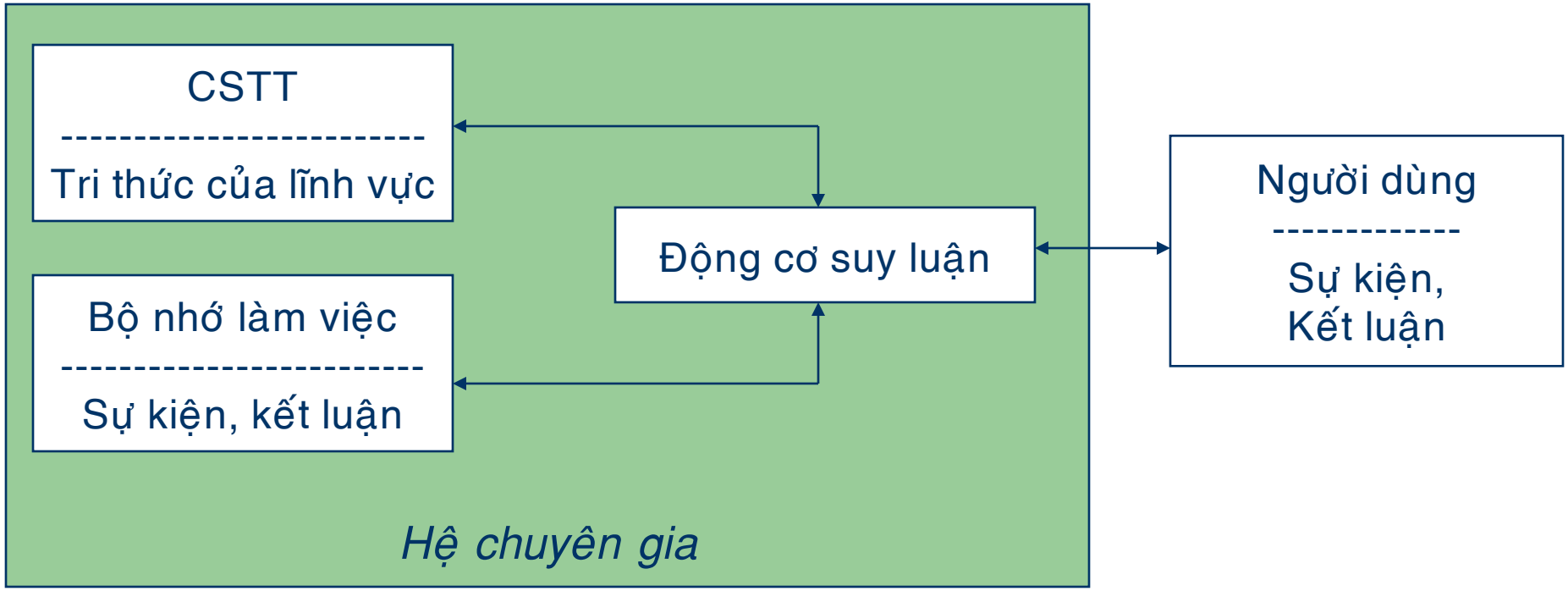
- Cấu trúc của ES:

ES mô phỏng khả năng giải quyết vấn đề của chuyên gia con người. Do vậy, chúng ta cần xem xét cách thức giải quyết của chuyên gia con người, để từ đó mô phỏng.



Cấu trúc của ES.

♣ Cách giải quyết vấn đề ở ES:



Cấu trúc của ES.

- CSTT:

Là một bộ phận của ES nhằm chứa tri thức của lĩnh vực.

♣ ES chứa tri thức của chuyên gia con người trong một bộ phận được gọi là CSTT. Để có tri thức này, người kỹ sư tri thức phải thu thập tri thức từ chuyên gia con người rồi mã hoá vào CSTT – cách thức mã hoá sẽ được đề cập trong phần kỹ thuật biểu diễn tri thức.

♣ Một trong các cách tiêu biểu để biểu diễn là dùng luật, như sau:

RULE 1:

IF "Xe car không thể khởi động được"

THEN "Vấn đề trong hệ thống điện"

RULE 2:

IF "Vấn đề trong hệ thống điện"

AND "Điện thế AC-quy nhỏ hơn 10Volt"

THEN lỗi tại bộ AC-quy"

Cấu trúc của ES.

- Bộ nhớ làm việc:

  Là bộ phận của ES dùng để chứa các sự kiện của vấn đề. Các sự kiện này có thể do người dùng nhập vào lúc đầu hay do ES sinh ra trong quá trình làm việc.

  - Với ES dùng cho nhiều người cùng thì bộ nhớ làm việc thường phân nhóm theo phiên làm việc (session) của người dùng. Đó là trường hợp một ES chung cho nhiều người dùng từ xa.

  - Nhiều ES cũng tận dụng các thông tin được chứa trong các nguồn ngoài như: CSDL, bảng tính, sensor,…ES sẽ tải thông tin này vào bộ nhớ làm việc đầu mỗi session hay khi cần thiết.

Cấu trúc của ES.

- Động cơ suy luận:

Là bộ xử lý trong hệ chuyên gia, là nhiệm vụ *so trùng các sự kiện được chứa trong bộ nhớ làm việc với tri thức được chứa trong CSTT nhằm dẫn ra kết luận cho vấn đề.*

♣ Tiêu biểu, nếu CSTT có chứa luật, ES sẽ tìm ra luật mà các tiên đề của luật so trùng với các sự kiện được chứa trong bộ nhớ làm việc, lúc đó ES sẽ thêm các kết luận của luật đó vào bộ nhớ làm việc, rồi tiếp tục tìm ra sự so trùng khác – giống như nguyên lý hoạt động của hệ luật sinh.

♣ Ví dụ: Giả sử CSTT chỉ với hai luật nêu trên

Bước 1:

ES: Có phải xe car không khởi động được ?

Người dùng: Đúng.

Cấu trúc của ES.

Chú thích: *Người dùng trả lời "Đúng", nên ES thêm vào bộ nhớ làm việc sự kiện để mô tả:*

"Xe car không thể khởi động được"

Động cơ suy diễn của ES làm nhiệm vụ so trùng, nhận thấy RULE 1 có thể so trùng được, nên nó thêm vào bộ nhớ làm việc phần kết luận của RULE 1, đó là:

"Vấn đề trong hệ thống điện"

Bước 2:

ES: Có phải điện Ac-quy dưới 10 Volt?

Người dùng: Đúng.

Chú thích: *Người dùng trả lời "Đúng", nên ES thêm vào bộ nhớ làm việc sự kiện để mô tả:*

"Điện thế Ac-quy nhỏ hơn 10Volt"

Động cơ suy diễn của ES làm nhiệm vụ so trùng, nhận thấy RULE 2 có thể so trùng được, nên nó thêm vào bộ nhớ làm việc phần kết luận của RULE 2, đó là:

lỗi tại bộ Ac-quy" – phiên làm việc cũng kết thúc vì CSTT chỉ gồm hai luật trên.

Cấu trúc của ES.

- Tiện ích giải thích.

Một trong các điểm nổi bật của ES là khả năng giải thích về suy luận của nó. ES còn có một khối cơ bản nữa trong cấu trúc của nó đó là: *khối tiện ích giải thích*. Với khối này ES có thể cung cấp cho người dùng các khả năng giải thích:

- Tại sao ES lại hỏi câu hỏi nào đó. (WHY)
- Bằng cách nào ES có thể suy ra kết luận nào đó. (HOW)

Khối tiện ích giải thích thuận tiện cho cả người phát triển ES và người dùng. Người phát triển có thể nhờ đó khám phá các lỗi trong tri thức của ES. Người thì có thể yên tâm hơn khi nhận một kết luận nào đó, không cần thiết phải quan tâm với cấu trúc tri thức của ES.

♣ Giải thích bằng cách nào (HOW)

Ngoài chức năng cung cấp cho người dùng kết quả suy luận cuối cùng, ES càng có thể cung cấp nó đạt đến kết luận bằng cách nào.

Cấu trúc của ES.

Khả năng giải thích bằng cách nào rất quan trọng với ES, nó làm cho ES khác với các chương trình truyền thống. Các chương trình truyền thống làm việc trên các bài toán được định nghĩa tốt, kết quả của chương trình đã được kiểm nghiệm trước khi chuyển giao. ES làm việc trên các bài toán thiếu nhiều thông tin, thậm chí tri thức còn có thể cập nhật khi ES đã được triển khai – chính vì vậy mà kết luận của nó phải được giải thích bằng cách nào để cho kết quả cuối cùng có giá trị hơn.

Ví dụ: trong phiên làm việc của ES nói trên.

ES: lỗi tại bộ Ac-quy".

Người dùng: HOW

ES: "Bởi vì, khi xe của bạn không khởi động được, tôi cho rằng đã có vấn đề trong hệ thống điện. Mỗi lần, tôi thấy điện thế của ac-quy dưới 10 volt, tôi biết rằng ac-quy đã hư."

- Khả năng giải thích HOW của ES có thể thực hiện được bằng cách cho phép nó theo vết các luật mà nó đã sử dụng vào suy luận.

Cấu trúc của ES.

♣ Giải thích tại sao (WHY)

khả năng của ES cung cấp cách giải thích: "tạo sao nó lại hỏi một câu hỏi nào đó." Với khả năng này, người dùng cũng có thể theo vết được luồng suy luận của ES và yên tâm hơn với kết quả.

Ví dụ: trong phiên làm việc với ES nói trên.

ES: Có phải xe car không khởi động được ?

Người dùng: WHY

ES: "Nếu tôi biết rằng xe car không khởi động được, thì tôi thường cho rằng có vấn đề trong hệ thống điện."

Khi chúng ta hỏi WHY, ES thường đáp trả bằng cách mô tả cái gì mà nó có thể kết luận từ câu trả lời. Hầu hết các ES thường đáp trả bằng cách hiện luật mà nó đang quan tâm.

Cấu trúc của ES.

- Giao diện người dùng:

  Giao diện cũng là một thành phần quan trọng của ES, nó giúp cho ES có thể đặt câu hỏi với người dùng và nhận về câu trả lời chính xác. Yêu cầu cao nhất cho giao diện là có khả năng cung cấp cách hỏi đáp tương tự như giữa người - với - người.

  Khi hiện thực hệ thống, vì những hạn chế của kỹ thuật hiện tại nên người thiết kế phải nghĩ đến những hình thức giao tiếp sao cho tiện lợi, tuy chưa thật giống với "người- người". Cụ thể, có thể dùng giao diện đồ họa , dạng menu chọn, phát âm câu hỏi, … *cũng cần phải tính đến khả năng dùng web như môi trường tương tác.*

Các đặc trưng cơ bản của ES.

- Phân tách tri thức và điều khiển.

  Đã đề cập trong hệ luật sinh. Đây cũng là đặc điểm phân biệt giữa chương trình truyền thống và ES.

  Hãy so sánh khả năng thay đổi tri thức của vấn đề giữa hai loại chương trình nói trên.

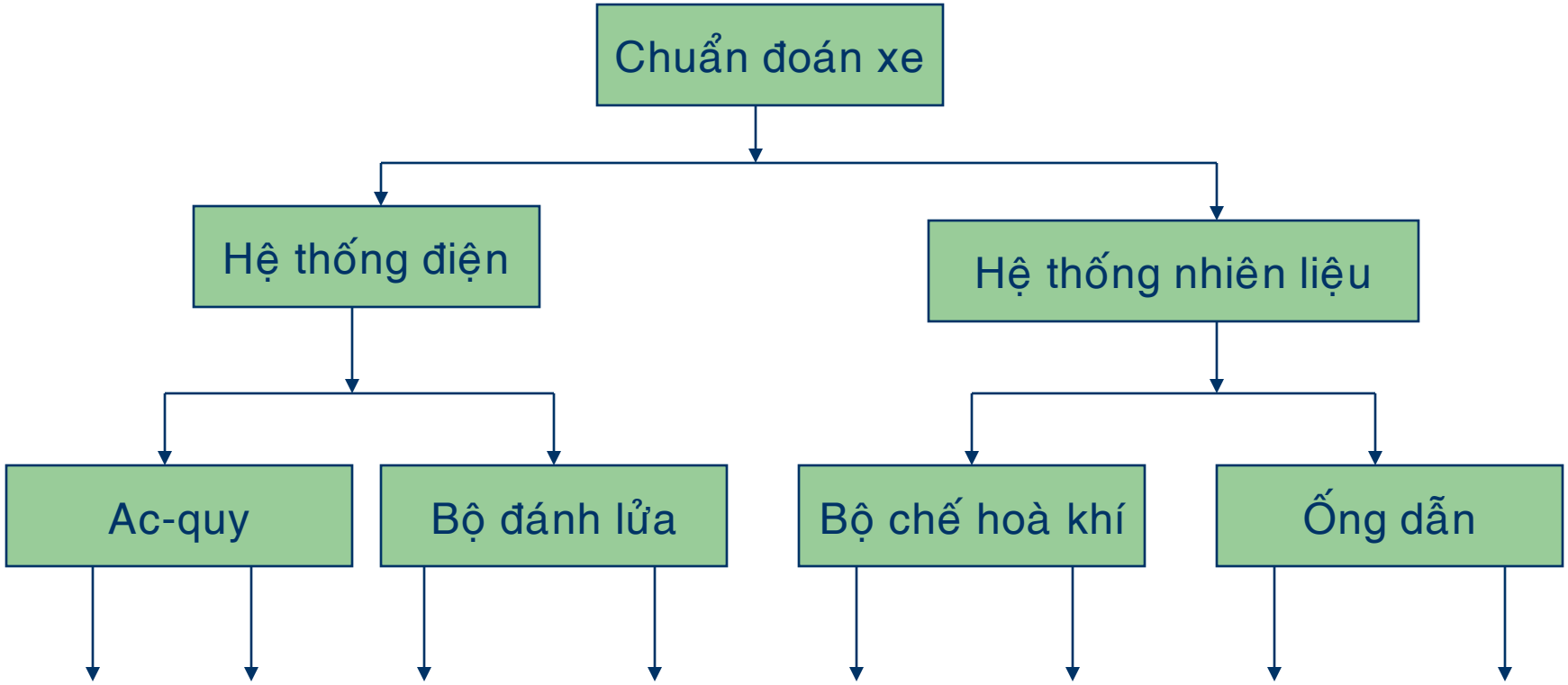
- Sỡ hữu tri thức chuyên gia.

  ES có chứa tri thức của lĩnh vực trong CSTT. Nhờ có tri thức mà nó có giá trị. Đặc biệt là tri thức này có thể được nhân ra thành nhiều bản, có thể cập nhật trong khi hệ thống đã được triển khai.

- Tính chuyên gia trong lĩnh vực hẹp.

  Cũng giống như chuyên gia con người, ES được phát triển nhằm vào một lĩnh vực hẹp. Điều này cũng dễ hiểu, vì lý do: trong lĩnh vực hẹp đó số lượng tri thức cũng nhỏ hơn, và giúp cho người thiết dễ dàng quản lý hơn, dể dàng thử nghiệm chiến lược điều khiển trong động cơ suy diễn. Người thiết thường chia tri thức theo từng mảng như hình sau để quản lý nó.

Các đặc trưng cơ bản của ES.



Các đặc trưng cơ bản của ES.

- Suy luận trên ký hiệu.

  Chúng ta có thể dùng ký hiệu để thể hiện tri thức cho ES. Chính vì vậy mà có thể tận dụng được các giải thuật trên ký hiệu để tri thức thức, như các giải thuật đã đề cập trong chương 2 – phần phép toán vị từ.

- Suy luận có heuristic

  Chuyên gia con người có thể từ kinh nghiệm của mình để dẫn ra cách giải quyết vấn đề hiệu quả hơn, ví dụ:

  Khi chuẩn đoán xe, họ có thể giả thiết cách làm:

  - Luôn luôn kiểm tra luật về hệ thống điện trước các luật khác.

  Hay một bác sĩ chuyên khoa có thể giả thiết:

  - Nếu nghi ngờ bị ung thư, thì kiểm tra dòng họ trước.

  Để có thể hiện thực trong ES, người thiết kế cần phải có cách đánh giá thứ tự ưu tiên của các luật, để từ một ngữ cảnh nào đó có thể chọn một luật có lý nhất để bắt đầu.

Các đặc trưng cơ bản của ES.

- Cho phép suy luận không chính xác.

  ES có một khả năng rất mạnh đó là: nó có thể làm việc với các vấn đề đang thiếu thông tin, hay có nhưng hỗn tạp, không rõ ràng. Cũng giống như trường hợp: một ekip bác sĩ đang phải cứu một bệnh nhân hấp hối, lúc đó họ không còn kịp thời gian để làm tất cả các xét nghiệm cần thiết. Khi thiếu thông tin như vậy họ đành tiến hành những cách có lý nhất theo họ. Chúng ta cũng có thể hiện thực ES có tính chất đó bằng cách đưa vào những luật tương ứng với tình huống thiếu thông tin để động cơ suy diễn vận dụng.

- Bị giới hạn vào vấn đề giải quyết.

  Không phải mọi vấn đề đều có thể giải quyết bởi ES. Cụ thể, nếu lĩnh vực chúng muốn xây dựng ES hiện tại chưa có, chưa cần một chuyên gia con người thì việc xây dựng ES khó mà thành công.

- Giải quyết các vấn đề có độ phức tạp vừa phải.

  Nếu vấn đề quá khó, yêu cầu chuyên gia con người đến vài giờ, cần thiết nghĩ đến khả năng chia thành nhiều bài toán con tương ứng mỗi ES.

Các đặc trưng cơ bản của ES.

- Có khả năng bị lỗi.

Giống như chuyên gia con người ES có khả năng bị lỗi. Chính vì vậy, cần thiết đưa vào khả năng phục hồi lại lỗi cho ES – ES có khả năng lưu vết quá trình suy luận, nếu nó đưa ra một kết luận mà người dùng kiểm nghiệm với thực tế có sai và báo cho ES, lúc đó nó phải có khả năng ghi nhận và theo đuổi một hướng suy luận khác.

đặc điểm này không xuất hiện trong các chương trình truyền thống, nhưng đừng vội kết luận loại chương trình đó tốt hơn. Mỗi loại có những đặc điểm riêng như bảng so sánh sau:

CT truyền thống	ES
Xử lý số	Xử lý ký hiệu.
Giải thuật	Heuristic
Tích hợp thông tin+ điều khiển	Tách bạch thông tin+ điều khiển
Khó thay đổi	dễ thay đổi.
Thông tin chính xác	Thông tin không chắc chắn.
Giao diện lệnh điều khiển	Hội thoại + giải thích.
Kết quả cuối cùng	đề nghị + giải thích
Tối ưu	Có thể chấp nhận.

Công nghệ tri thức.

- Quá trình gồm các giai đoạn như hình bên.
- Một số định nghĩa:
  - ♣ *Công nghệ tri thức:* ***Là quá trình xây dựng ES.***
  - ♣ Thu thập tri thức: Là quá trình thu thập, tổ chức và nghiên cứu tri thức.

Các nhân tố trong một dự án ES

- **Các nhân tố chính:**
  - Chuyên gia lĩnh vực.
  - Kỹ sư tri thức
  - Người dùng sản phẩm

- **Các yêu cầu cho mỗi nhân tố:**

Chuyên gia lĩnh vực:

- Có tri thức chuyên gia
- Có kỹ năng giải quyết vấn đề hiệu quả
- Có thể chuyển giao tri thức
- Không chống đối (thân thiện).

Kỹ sư tri thức:

- Có kỹ năng về công nghệ tri thức
- Có kỹ năng giao tiếp tốt.
- Có thể làm cho vấn đề được giải quyết bởi phần mềm.
- Có kỹ năng lập trình hệ chuyên gia.

Người dùng sản phẩm:

- Có thể trợ giúp thiết kế giao diện cho ES.
- Có thể trợ giúp việc thu thập tri thức.
- Có thể trợ giúp trong quá trình phát triển ES.

Các kỹ thuật suy luận

- Suy luận: là quá trình làm việc với tri thức, sự kiện, chiến lược giải toán để dẫn ra kết luận.
- Bạn suy luận như thế nào?

Các hình thức cơ bản:

- Suy luận diễn dịch.
- Suy luận quy nạp.
- Suy luận tương tự.
- Suy luận khả sai.
- Suy luận common-sense.
- Suy luận đơn điệu
- Suy luận không đơn điệu.

Các kỹ thuật cơ bản:

- Suy luận tiến (forward-chaing)
- Suy luận lùi (backward-chaining)

Ưu – nhược điểm của mỗi kỹ thuật

Suy luận tiến – forward chaining

- Ưu điểm:
  - ♣ Làm việc tốt với bài toán có bản chất: gôm thông tin và sau đó tìm xem có thể suy ra cái gì từ thông tin đó.
  - ♣ Có thể dẫn ra rất nhiều thông tin chỉ từ một ít sự kiện ban đầu.
  - ♣ Thích hợp cho một số vấn đề như: *hoạch định, giám sát, điều khiển, diễn dịch.*

- Nhược điểm:
  - ♣ Không có cách để nhận thấy tính quan trọng của từng sự kiện. Hỏi nhiều câu hỏi thừa, vì đôi lúc chỉ cần một vài sự kiện là cho ra kết luận.
  - ♣ Có thể hỏi những câu hỏi không liên quan gì nhau – chuỗi câu hỏi không ăn nhập nhau.

VD:

- Bạn có thân nhiệt cao ?
- Bạn đến VN đã lâu rồi ?
- …

Ưu – nhược điểm của mỗi kỹ thuật

Suy luận lùi – backward chaining

- Ưu điểm:
  - ♣ Làm việc tốt với bài toán có bản chất: thành lập giả thiết , sau đó tìm xem có thể chứng minh được không.
  - ♣ Hướng đến một goal nào, nên hỏi những câu hỏi có liên quan nhau.
  - ♣ Chỉ khảo sát CSTT trên nhánh vấn đề đang quan tâm.
  - ♣ Tốt cho các vấn đề: *chuẩn đoán, kê toa, gỡ rối.*

- Nhược điểm:
  - ♣ Luôn hướng theo dòng suy luận định trước thậm chí có thể dừng và rẽ sang một goal khác.
  - → Giải quyết: dùng meta-rule để khắc phục.
  - Meta-rule: dùng để hướng không gian tri thức được khảo sát sang một vùng khác.

Khảo sát ES: MYCIN

- **Giới thiệu:**
  - ♣ Là ES về lĩnh vực chuẩn đoán bệnh nhiễm trùng nháu.
  - ♣ Được phát triển vào khoảng năm 1970, tại đại học Standford
  - ♣ Là thước đo cho nhiều ES dựa trên luật sau này.
  - ♣ Được viết trên ngôn ngữ INTERLISP, một nhánh của LISP.
  - ♣ Người ta từ MYCIN, tước bỏ phần CSTT của nó để được một hệ thống EMYCIN (empty MYCIN), dùng để phát triển các ES khác.

- **Các đặc điểm chính:**
  - ♣ Sử dụng kỹ thuật suy luận lùi.
  - ♣ Có khả năng phân tách tri thức và điều khiển.
  - ♣ Có tích hợp Meta-rule.
  - ♣ Có thể dùng khi thiếu thông tin hay thông tin không chính xác.
  - ♣ Có khả năng nhớ được các thông tin trong một phiên làm việc với người bệnh cụ thể.
  - ♣ Dể sử dụng – giao tiếp với bác sĩ qua tiếng anh.
  - ♣ Cung cấp các chức năng giải thích: WHY, HOW.

Khảo sát ES: MYCIN

- Một phiên làm việc cụ thể với MYCIN
  ("Expert System". Author: John Durkin)
  --- đã được Việt hóa.

? Một vài thông tin về bệnh nhân:

1. MYCIN: Họ tên ?
   USER: Jack Durkin.
2. MYCIN: Tuổi ?
   USER: 24
3. MYCIN: Giới tính ?
   USER: Nam
4. MYCIN: Chủng tộc?
   USER: CAUCASIAN

? MYCIN cố gắng xác định bản chất lây nhiễm.

5. MYCIN: Có bất kỳ phép cấy vi khuẩn cho Jack Durkin liên qua đến bệnh hiện tại?
   USER: Có
6. MYCIN: Mẫu vật lấy từ đâu?
   USER: CSF
7. MYCIN: Ngày, giờ thực hiện cấy?
   USER: 7- July-7:00

….

MYCIN: Gần đây Jack Durkin có các triệu chứng như: choáng ván, hôn mê không?

USER: Có.

….

Khảo sát ES: XCON

- Giới thiệu:

  ♣ Là ES về lĩnh vực hướng dẫn cấu hình các hệ thống máy tính của DEC trước khi xuất xưởng.

  ♣ Được phát triển vào khoảng năm 1980, tại đại học CMU

  ♣ Được công bố giảm thời gian cấu hình cho mỗi hệ thống xuống còn 2 phút (so với 25 phút bằng tay.). Tiết kiệm vào khoảng 25 triệu $ cho mỗi năm.

  (Theo "Expert System" – John Durkin)

Hệ chuyên gia dựa trên luật

- **Định nghĩa:**

  Là một chương trình máy tính, xử lý các thông tin cụ thể của bài toán được chứa trong bộ nhớ làm việc và tập các luật được chứa trong CSTT, sử dụng động cơ suy luận để suy ra thông tin mới.

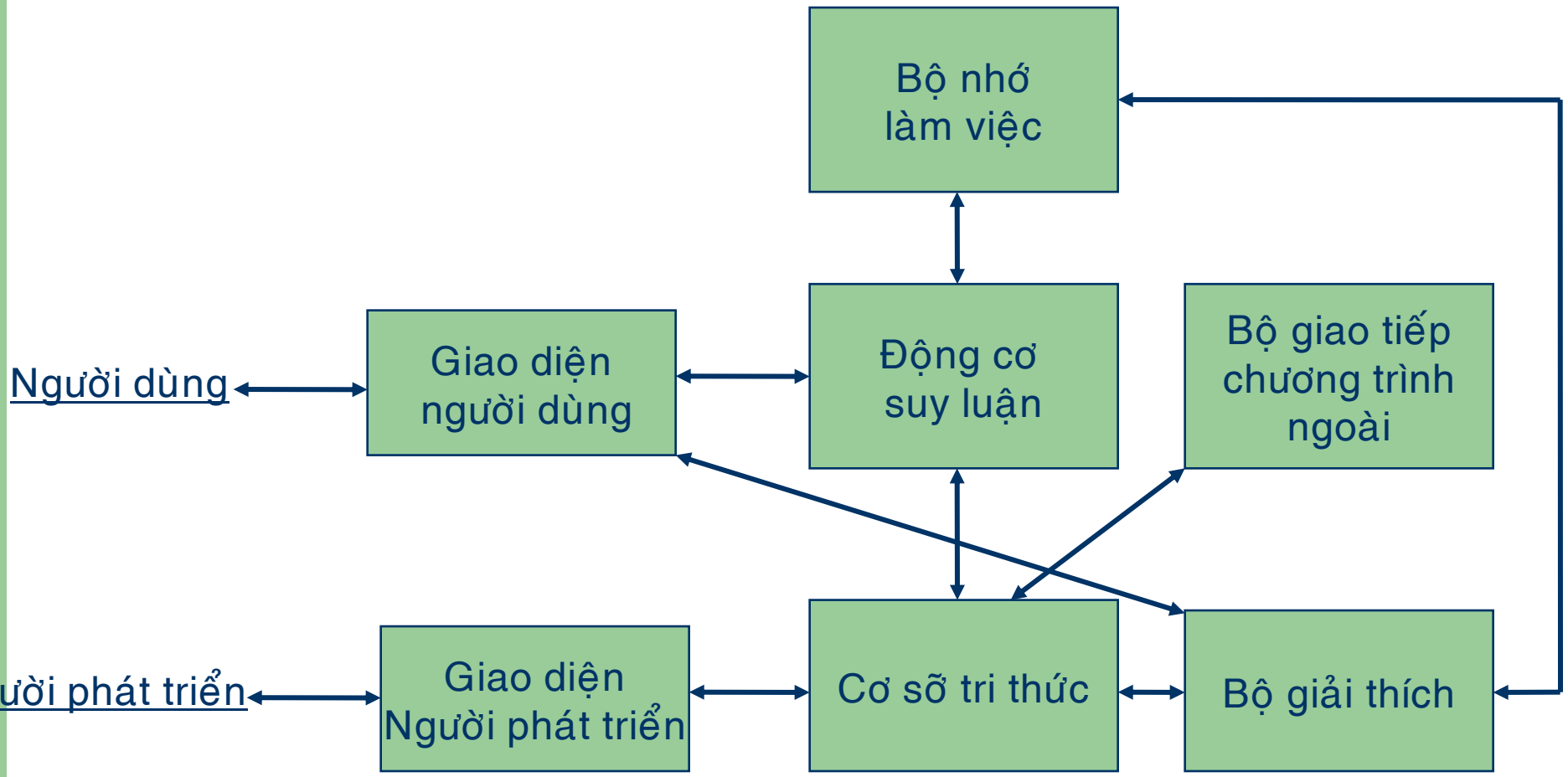
  ES dựa trên luật: có nền tảng xây dựng lá hệ luật sinh – chương trước.

  ES dựa trên luật cũng có những đặc trưng cơ bản như đã nêu trong phần trước cho các ES tổng quát, một vài đặc điểm:

  - ♣ Có CSTT chứa các luật.
  - ♣ Có bộ nhớ làm việc tạm thời.
  - ♣ Có động cơ suy luận.
  - ♣ Có một giao diện để giao tiếp với người dùng, người phát triển.
  - ♣ Có tiện ích giải thích.
  - ♣ Có khả năng giao tiếp với chương trình ngoài như: các DBMS, xử lý bảng tính,…

Hệ chuyên gia dựa trên luật

- Kiến trúc: (như hình sau)
- Nguyên lý hoạt động tương tự hệ luật sinh đã giới thiệu.



Hệ chuyên gia dựa trên luật

- Ưu điểm
  - ♣ Biểu diễn tri thức tự nhiên: IF… THEN.
  - ♣ Phân tách tri thức – điều khiển.
  - ♣ Tri thức là tập các luật có tính độc lập cao -> dễ thay đổi, chỉnh sữa.
  - ♣ Dễ mở rộng.
  - ♣ Tận dụng được tri thức heuristic.
  - ♣ Có thể dùng biến trong luật, tri xuất chương trình ngoài.

- Nhược điểm
  - ♣ Các fact muốn đồng nhất nhau, phải khớp nhau hoàn toàn → Các facts cùng một ý nghĩa phải giống nhau về cú pháp, ngôn ngữ tự nhiên không như vậy.
  - ♣ khó tìm mối qua hệ giữa các luật trong một chuỗi suy luận, vì chúng có thể nằm rải rác trong CSTT.
  - ♣ Có thể hoạt động chậm.
  - ♣ Làm cho nhà phát triển phải hình chung mọi cái ở dạng luật -→ không phải bài toán nào cũng có thể làm được như thế này.

Chương 7: BIỂU DIỄN TRI THỨC

- Biểu diễn tri thức trong AI: vai trò và ứng dụng
- Các kỹ thuật biểu diễn tri thức:
  - Semantic network
  - Lưu đồ phụ thuộc khái niệm
  - Frame
  - Script

ThS Nguyễn Cao Trí – caotri@dit.hcmut.edu.vn

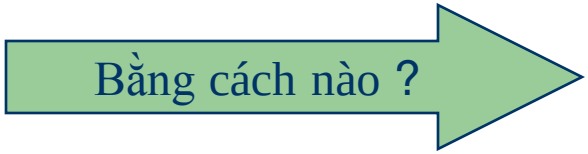
KS Lê Thành Sách – ltsach@dit.hcmut.edu.vn

Các lược đồ biểu diễn tri thức.

- Định nghĩa:

  Biểu diễn tri thức là phương pháp để mã hoá tri thức, nhằm thành lập cơ sỡ tri thức cho các hệ thống dựa trên tri thức (knowledge-based system).

Gồm: đối tượng và các quan hệ giữa chúng trong lĩnh vực.



Bằng cách: dùng các lược đồ biểu diễn (scheme).
→ Chọn dùng lược đồ cho loại tri thức là vấn đề quan trọng.

Gồm: Bảng ánh xạ giữa:
Đối tượng thực → đối tượng tính toán.
Quan hệ thực → quan hệ tính toán.

Các lược đồ biểu diễn tri thức.

- Chú ý:

  Cần phân biệt: Lược đồ biểu diễn (scheme) và môi trường hiện thực (medium), tương tự như việc phân biệt: cấu trúc dữ liệu (CTDL) và ngôn ngữ lập trình. Với một loại CTDL, ví dụ như: Bản ghi (record), chúng ta có hiện thực trong nhiều ngôn ngữ như: Pascal, C,…Tương tự, với một loại lược đồ nào đó chúng ta có thể chọn một trong các NNLT để hiện thực nó.

- Các loại lược đồ biểu diễn:

  ♣ ***Lược đồ logic.***

  Dùng các biểu thức trong logic hình thức ,như phép toán vị từ, để biểu diễn tri thức.

  Các luật suy diễn áp dụng cho loại lược đồ này rất rõ ràng, đã khảo sát trong chương 2 (như: MP, MT,…).

  Ngôn ngữ lập trình hiện thực tốt nhất cho loại lược đồ này là: PROLOG.

  ♣ ***Lược đồ thủ tục:***

  Biểu diễn tri thức như tập các chỉ thị lệnh để giải quyết vấn đề.

Các lược đồ biểu diễn tri thức.

♣ ***Lược đồ thủ tục:***

Ngược lại với các lược đồ dạng khai báo, như logic và mạng, các chỉ thị lệnh trong lược đồ thủ tục chỉ ra bằng cách nào giải quyết vấn đề.

Các luật trong CSTT của ES dựa trên luật là ví dụ về thủ tục giải quyết vấn đề.

Hệ luật sinh là ví dụ điển hình của loại lược đồ này.

♣ ***Lược đồ mạng.***

Biểu diễn tri thức như là đồ thị; các đỉnh như là các đối tượng hoặc khái niệm, các cung như là quan hệ giữa chúng.

Các ví dụ về loại lược đồ này gồm: mạng ngữ nghĩa, phụ thuộc khái niệm, đồ thị khái niệm → được khảo sát sau đây của chương này.

♣ ***Lược đồ cấu trúc:***

Là một mở rộng của lược đồ mạng; bằng cách cho phép các node có thể là một CTDL phức tạp gồm các khe(slot) có tên và trị hay một thủ tục. Chính vì vậy nó tích hợp cả dạng khai báo và thủ tục.

Kịch bản(script) , khung (frame), đối tượng (object) là ví dụ của lược đồ này → khảo sát sau.

Các chú ý về lược đồ.

- Khi xây dựng các lược đồ cần chú ý những vấn đề sau:
  - ♣ Các đối tượng và các quan hệ có thể biểu diễn cho cái gì trong lĩnh vực?

    Ví dụ: để biểu diễn cho ý "Nam cao 1mét 70",chúng ta có thể dùng: ***chieucao(nam,170).*** Vậy thì để diễn tả "An cao hơn Nam" chúng ta làm như thế nào, vì chiều cao của An lúc này không là một trị cụ thể nữa!
  - ♣ Bằng cách nào phân biệt giữa "nội hàm" và "ngoại diện" của một khái niệm.
  - ♣ Bằng cách nào thể hiện được meta-knowledge?
  - ♣ Bằng cách nào thể hiện tính phân cấp của tri thức.

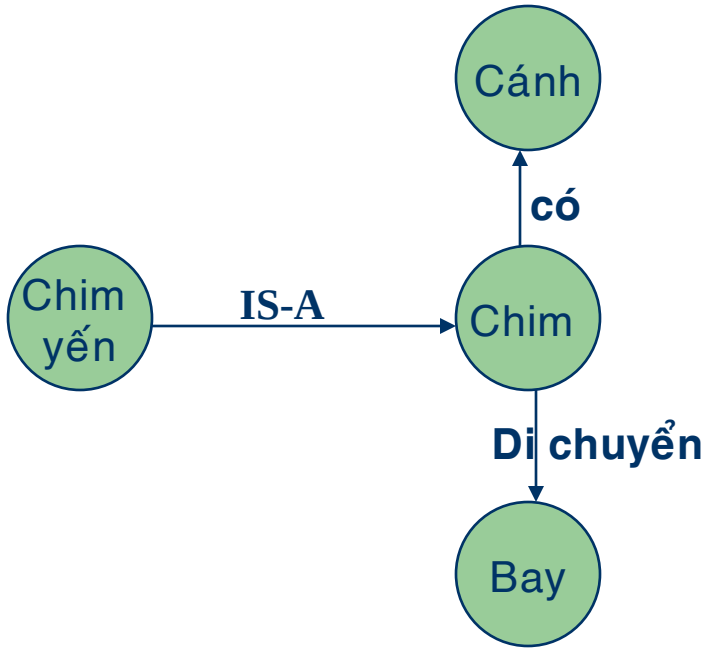
    Lúc biểu diễn tính phân cấp thì các hình thức : kế thừa, ngoại lệ, trị mặc định, ngoại lệ, đa thừa kế phải đặc tả như thế nào
  - ♣ Khi mô tả đối tượng, bằng cách nào có thể tích hợp một tri thức thủ tục vào bản thân mô tả, khi nào thủ tục được thực hiện,..

Mạng ngữ nghĩa

- Định nghĩa:

  Là một lược đồ biểu diễn kiểu mạng, dùng đồ thị để biểu diễn tri thức. Các đỉnh biểu diễn đối tượng; các cung biểu diễn quan hệ giữa chúng.

- Ví dụ:



Xem mạng bên:

- Có hai đỉnh biểu diễn đối tượng, và hai đỉnh còn lại biểu diễn thuộc tính.
- đỉnh có nhãn: "Chim" nối với hai đỉnh thuộc tính có nhãn: "Cánh", "Bay" nên có thể biểu diễn: "Một con chim thì có cánh và có hình thức di chuyển là bay".
-Đỉnh có nhãn "Chim **yến"** nối với đỉnh "Chim" thông qua cung đặc biệt "IS**-A"** nói lên: "Chim yến là một loài chim".Vì vậy chim yến có thể sỡ hữu các thuộc tính: có cánh, bay như một con chim thông thường.

Mạng ngữ nghĩa

- **Mở rộng mạng ngữ nghĩa:**

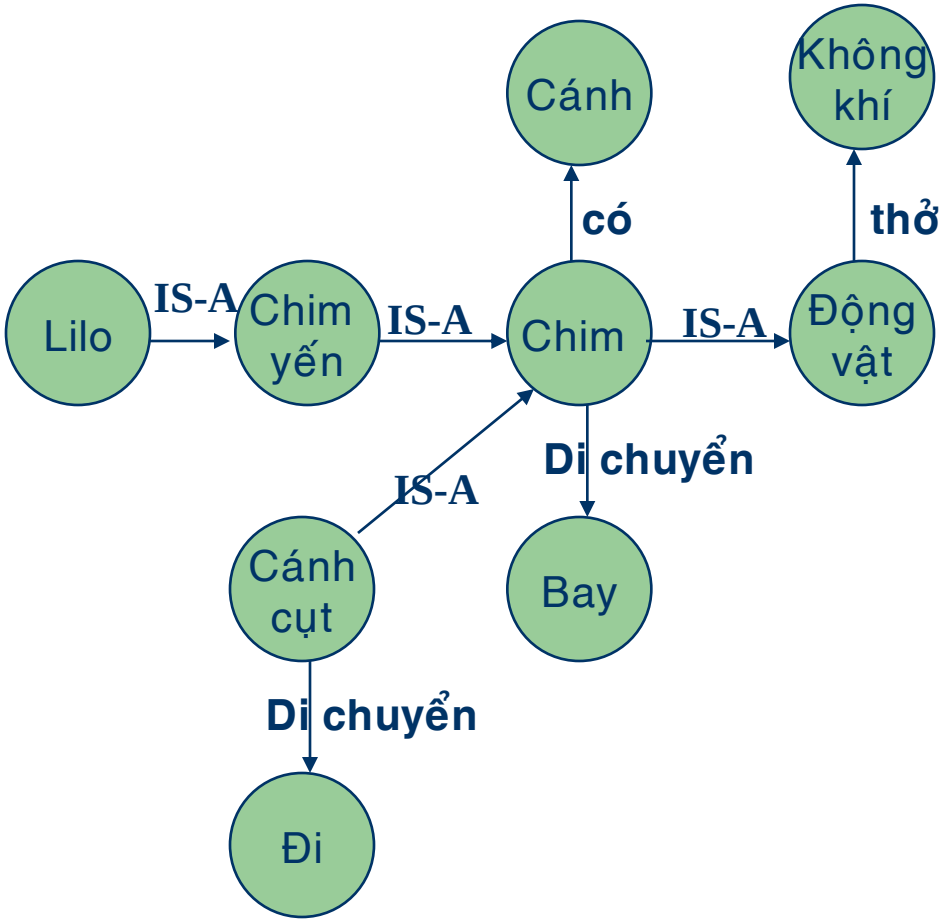
  Để mở rộng mạng thật đơn giản; chúng ta chỉ việc thêm các đỉnh và các cung quan hệ với các đỉnh có sẵn. Các đỉnh được thêm vào mạng hoặc là biểu diễn đối tượng hoặc là biểu diễn thuộc tính như ví dụ trước. Xét ví dụ sau đây minh họa việc mở rộng mạng đã có.

- **Tính thừa kế:**

  Là đặc điểm nổi bật của lược đồ mạng ngữ nghĩa. Mạng ngữ nghĩa định ra cung quan hệ đặc biệt "IS-A" để chỉ ra sự thừa kế. Ví dụ, nhờ tính thừa kế mà từ mạng bên chúng ta có thể suy ra: "Lilo là một động vật có thể bay và hít thở không khí."

- **Tính ngoại lệ:**

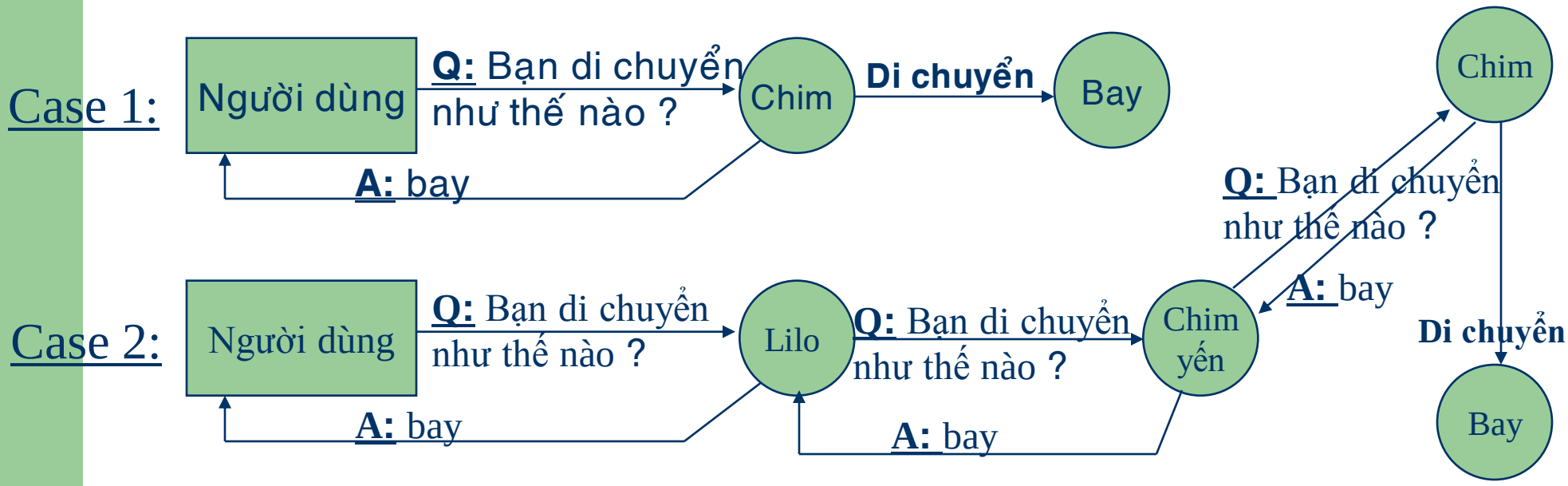
  Định nghĩa một cung quan hệ mới đến một đỉnh có trị khác.



Mạng ngữ nghĩa

- **Phép toán trên mạng ngữ nghĩa:**

  Giả sử chúng ta đã mã hoá mạng ở hình trước vào máy tính. Để dùng mạng, có thể đơn giản là chúng ta câu hỏi với một đỉnh nào đó. Ví dụ, với đỉnh "Chim" chúng ta đặt câu hỏi: "Bạn **di chuyển** như thế nào?". Để trả lời câu hòi chúng ta có thể hiện thực cách trả lời sau cho đỉnh: tìm kiếm cung quan hệ có nhãn "di **chuyển"** bắt đầu từ nó, như case 1,2 ở bên.



Lưu đồ về quan hệ phụ thuộc khái niệm.

- Trong quá trình nghiên cứu về cách hiểu ngôn ngữ tự nhiên, Schank và Rieger đã cố gắng thiết lập một tập các phần tử cơ bản để có thể biểu diễn cấu trúc ngữ nghĩa của các biểu thức ở ngôn ngữ tự nhiên theo một cách đồng nhất.

  Lý thuyết về phụ thuộc khái niệm có đề ra 4 khái niệm cơ bản để từ đó ngữ nghĩa được xây dựng, chúng là:

  - ACT (Action) : các hành động.
    : (các động từ trong câu)
  - PP (Picture Producers) : các đối tượng.
    : (các chủ từ, tân ngữ,..)
  - AA (Action Adder) : bổ nghĩa cho hành động.
    : (trạng từ)
  - PA (Picture Adder) : bổ nghĩa cho đối tượng.
    : (tính từ)

Lưu đồ về quan hệ phụ thuộc khái niệm.

- Tất cả các hành động được cho là có thể được mô tả bằng cách phân rã về một hoặc nhiều hành động như liệt kê sau đây:

1. ATRANS : chuyển đổi một quan hệ – VD: động từ: cho, biếu,…
2. PTRANS : chuyển đổi vị trí vật lý – VD: đi, chạy, di chuyển,..
3. PROPEL : tác động một lực vật lý lên đối tượng – VD: đẩy, chải,…
4. MOVE : di chuyển một phần thân thể bởi đối tượng – VD: đá..
5. GRASP : nắm lấy đối tượng khác. – VD: cầm, nắm, giữ,…
6. INGEST : ăn vào bụng một đối tượng bởi đt khác – VD: ăn, nuốt,..
7. EXPEL : tống ra từ thân thể của một đối tượng – VD: khóc,..
8. MTRANS : chuyển đổi thông tin tinh thần – VD: nói, tiết lộ,..
9. MBUILD : tạo ra một thông tin tinh thần mới – VD: quyết định, …
10. CONC : nghĩ về một ý kiến – VD: suy nghĩ, hình dung,…
11. SPEAK : tạo ra âm thanh – VD: nói, phát biểu,…
12. ATTEND: tập trng giác quan – VD: lắng nghe, nhìn,…

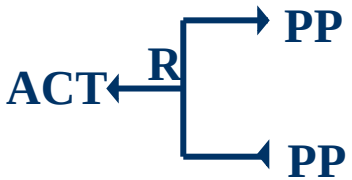
Lưu đồ về quan hệ phụ thuộc khái niệm.

- Quan hệ phụ thuộc khái niệm bao gồm một tập các luật cú pháp cho khái niệm, hình thành nên văn phạm về quan hệ ngữ nghĩa. Các quan hệ này sẽ được dùng vào việc biểu diễn bên trong cho câu trong ngôn ngữ tự nhiên. Danh sách các phụ thuộc khái niệm được liệt kê như bên.

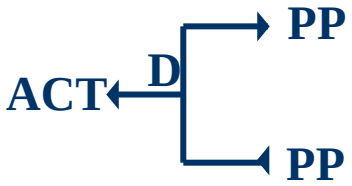
Đối tượng PP
thực hiện hành động ACT

Đối tượng PP có thuộc tính PA

Hành động ACT tác động lên PP



Đối tượng nhận và cho
trong hành động ACT
R: đối tượng nhận (recipient)

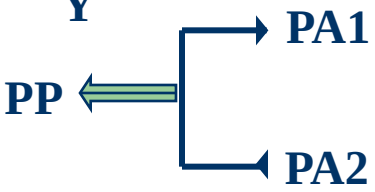


Hướng của đối tượng
trong hành động ACT
D: Hướng(Direction)

Quan hệ giữa hành động và
thiết bị phục vụ cho hành động.
(xem ví dụ phần sau)

Lưu đồ về quan hệ phụ thuộc khái niệm.

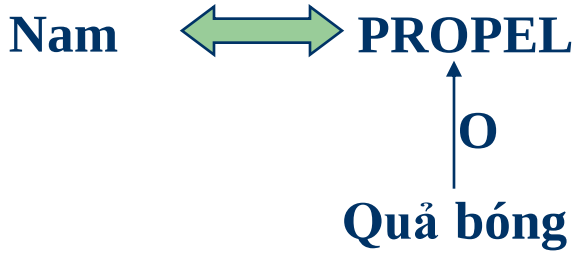
Biểu diễn quan hệ nhân quả.
X: nguyên nhân.
Y: kết quả.



Biểu diễn sự chuyển đổi trạng thái của PP từ PA2 sang PA1

Biểu diễn quan hệ sỡ hữu.
PP2 là PART OF hoặc POSSESSOR OF PP1

♣ Từ các phụ thuộc khái niệm cơ bản nêu trên. Chúng ta có thể kết hợp để có thể biểu diễn các câu trong NNTN, như ví dụ sau:



Ý nghĩa: "Nam đã tác dụng một lực vào quả bóng"

Lưu đồ về quan hệ phụ thuộc khái niệm.

- Các phụ thuộc khái niệm trên cho phép chúng biểu diễn quan hệ giữa: chủ từ với động từ (như phụ thuộc đầu tiên), hay giữa chủ từ và thuộc tính của nó,…. Lược đồ về quan hệ phụ thuộc khái niệm càng đưa ra cách thứa để biểu diễn thì, điều kiện,…, như bên phải.

p : quá khứ – ACT đã xảy ra trong quá khứ

VD:

Ý nghĩa: Nam ***đã*** tác dụng một lực (đẩy) vào cái bàn.

f : tương lai.

t : chuyển tiếp.

ts : bắt đầu chuyển tiếp.

tf : kết thúc chuyển tiếp.

k : đang diễn ra.

? : nghi vấn.

/ : phủ định.

C : điều kiện.

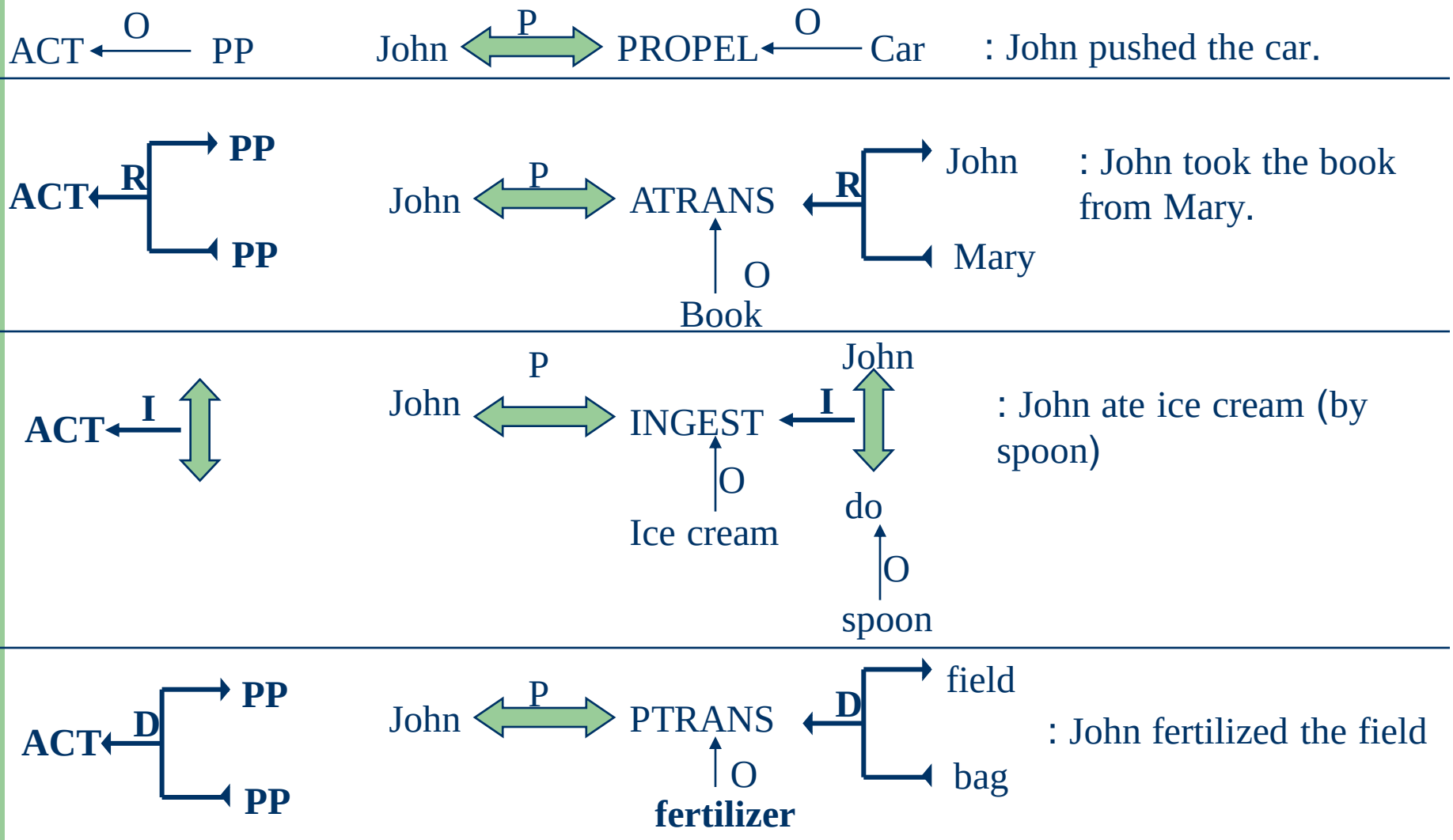
Nil: hiện tại. (không ghi chú gì)

Lưu đồ về quan hệ phụ thuộc khái niệm.

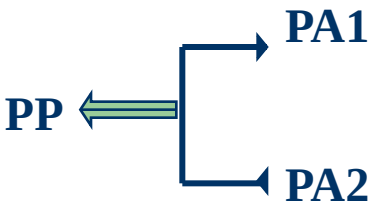
- Một số ví dụ về việc kết hợp các phụ thuộc khái niệm để biểu diễn câu:

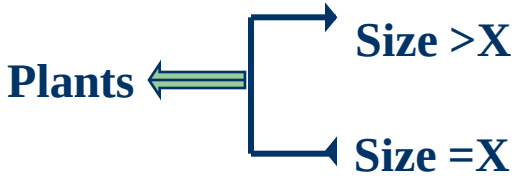
PP ⟺ ACT	**VD:**	Nam ⟺(P) PROPEL ←(O)— Lan	: Nam đã đánh Lan
		Nam ⟺(K) ATTEND ←(O)— Bài giảng	: Nam đang tập trung vào bài giảng.
PP ⟺ PA		Nam ⟺ Height (> average)	: Nam is tall.
PP ⟺ PP		Nam ⟺ doctor	: Nam is a doctor.
PP ←— PA		boy ←— nice	: A nice boy
PP ⟸ PP		dog ⟸(Poss-by) John	: John's dog.

Lưu đồ về quan hệ phụ thuộc khái niệm.



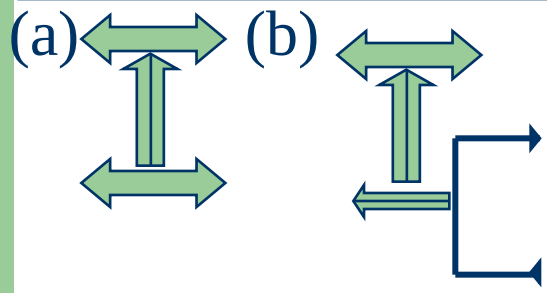
Lưu đồ về quan hệ phụ thuộc khái niệm.

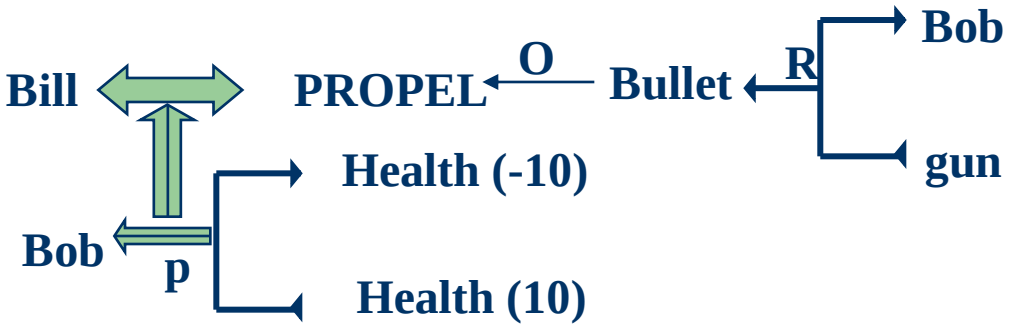




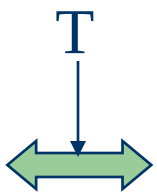
: The plants grew

(a) (b)





: Bill shot Bob



: John ran yesterday

Lưu đồ về quan hệ phụ thuộc khái niệm.

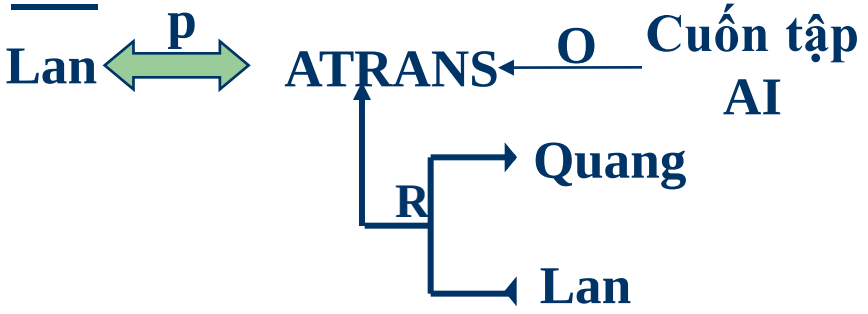
♣ Từ những kết hợp giữa các phụ thuộc khái niệm để biểu diễn các câu đơn giản ở trên, chúng ta có thể cũng có thể tạo ra biểu diễn cho các câu phức tạp hơn như ví dụ sau:

Câu: ***"Nam đã cấm Lan gởi cuốn tập AI cho Quang"***

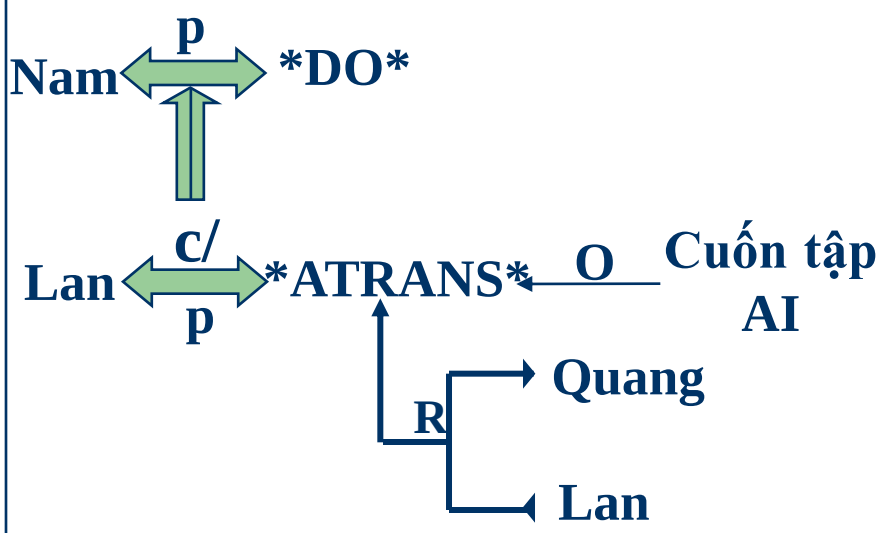
Nếu đặt C là mệnh đề: ***"Lan gởi cuốn tập cho Quang",*** thì câu trên có thể hiểu là: Nam cấm cái mệnh đề vừa nêu xảy ra.

Mà mệnh đề C được biểu diễn như **H1**, nên toàn bộ câu là như **H2**:

H1:



H2:



Lưu đồ về quan hệ phụ thuộc khái niệm.

- Ưu điểm
  - ♣ Cung cấp cách thức biểu diễn hình thức cho ngữ nghĩa của ngôn ngữ tự nhiên, ngữ nghĩa được biểu diễn theo dạng có quy tắc → giảm sự nhập nhằng.
  - ♣ Chính bản thân dạng biểu diễn chứa đựng ngữ nghĩa → tính **đồng nghĩa** tương ứng là sự **đồng nhất về cú pháp** của lược đồ biểu diễn → chứng minh tính đồng nghĩa ⇔ so trùng hai đồ thị biểu diễn.

- Nhược điểm:
  - ♣ Khó khăn trong việc phát triển chương trình để tự động thu giảm biểu diễn của câu bất kỳ về dạng quy tắc chuẩn.
  - ♣ Trả giá cho việc phân rã mọi cái về các thành phần cơ bản: ACT, PP, …
  - ♣ Các thành phần cơ bản không thích hợp để miêu tả những khái niệm tinh tế của ngôn ngữ tự nhiên, như các từ có nghĩa định tính: cao, đẹp,…

Đồ thị khái niệm

- Định nghĩa:

Đồ thị khái niệm là một đồ thị hữu hạn, liên thông, các đỉnh được chia làm hai loại: đỉnh khái niệm và đỉnh quan hệ.

Đỉnh khái niệm: dùng để biểu diễn các khái niệm cụ thể (cái, điện thoại, …) hay trừu tượng (tình yêu, đẹp, văn hoá,…). Đỉnh khái niệm được biểu diễn bởi hình chữ nhật có gán nhãn là khái niệm.

Đỉnh quan hệ: dùng để chỉ ra quan hệ giữa các khái niệm có nối đến nó.

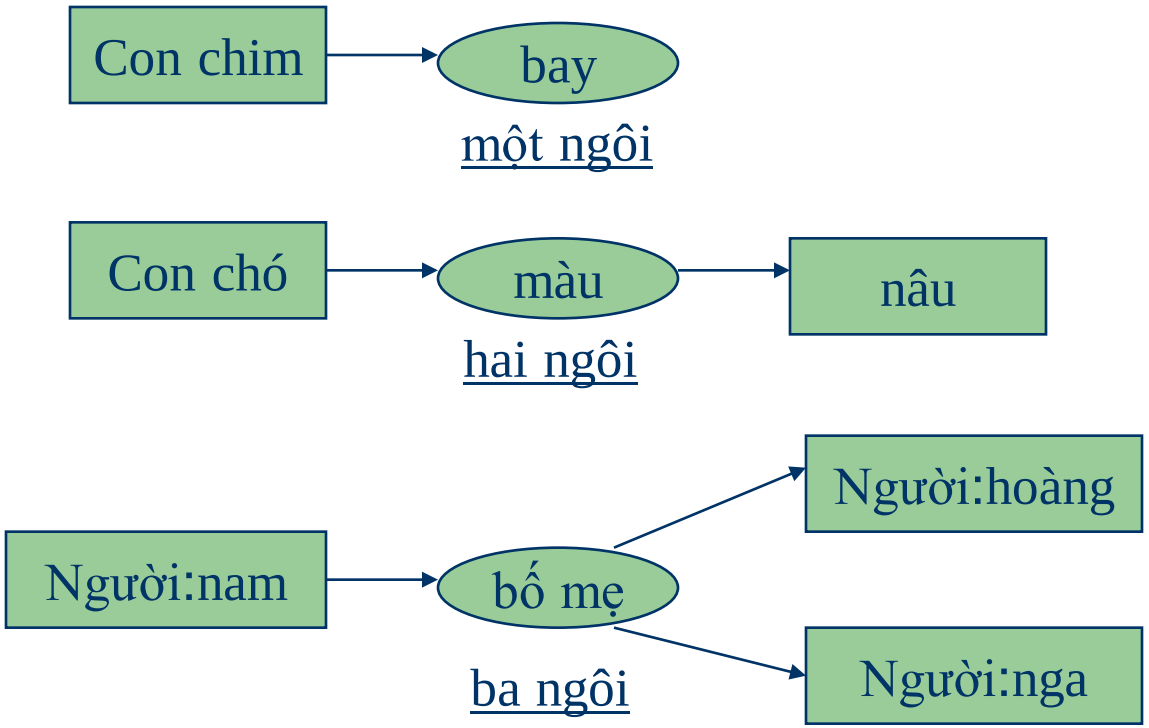
Trong đồ thị khái niệm: chỉ có khác loại mới nối được với nhau. Chính vì dùng đỉnh quan hệ nên các cung không cần phải được gán nhãn nữa.

♣ Mỗi đồ thị khái niệm biểu diễn một mệnh đề đơn.

♣ Cơ sỡ tri thức: chứa nhiều đồ thị khái niệm.

Đồ thị khái niệm

- Một số ví dụ:



Biểu diễn: "Con chim biết bay"

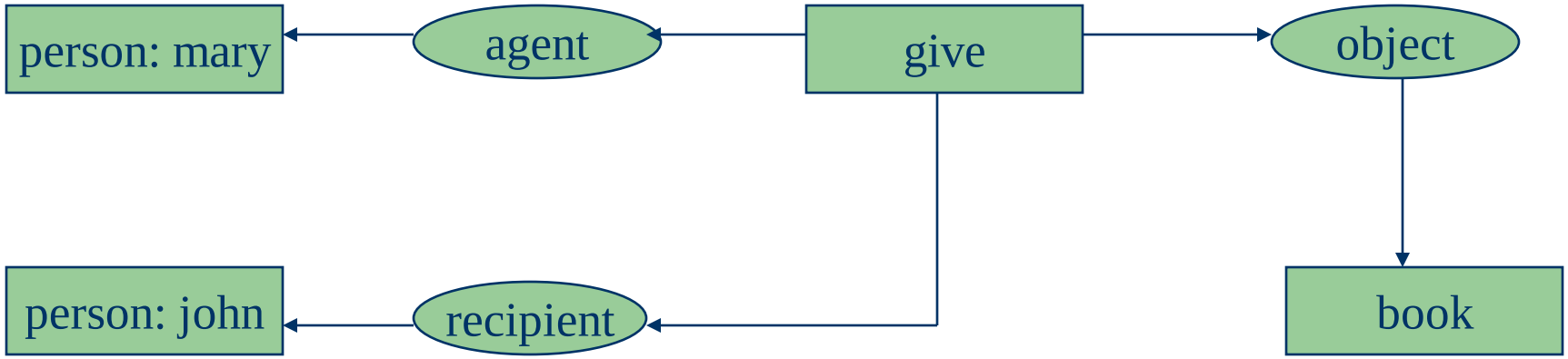
Biểu diễn: "con chó có màu nâu".

Biểu diễn: "Nam có bố mẹ là ông Hoàng và bà Nga".

** Một đỉnh quan hệ có thể là một hay nhiều ngôi.*

Đồ thị khái niệm

- Một số ví dụ:



Represent: "Mary give John the book"

Trong ví dụ trên, chú ý: động từ "give" có chủ từ thông qua đỉnh quan hệ "agent", tân ngữ trực tiếp thông qua đỉnh quan hệ "object", tân ngữ gián tiếp cũng là người nhận thông quan đỉnh quan hệ "recipient", hướng mũi tên cho các loại động từ tương tự có dạng như đồ thị trên.

Đồ thị khái niệm

- Loại, cá thể, tên:

Trong đồ thị khái niệm, mỗi đỉnh quan hệ biểu diễn cho một cá thể đơn lẽ thuộc một loại nào đó. Để nói lên quan hệ giữa "loại-cá thể", nên mỗi đỉnh khái niệm được quy định cách gán nhãn là:

"loại: tên_cá_thể"

tên_ cá_thể có thể là:

1. Một tên nào đó, như:

sinhviên: nam → một sinh viên có tên là Nam.

2. Một khoá để phân biệt, được viết theo cú pháp **#khoá,** như

sinhviên: #59701234 → một sinh viên có khoá là: 59701234.

3. Có thể dùng dấu sao (*) để chỉ ra một cá thể chưa xác định, như:

sinhviên: * , có tác dụng như sinhviên → chỉ ra một sinh viên bất kỳ

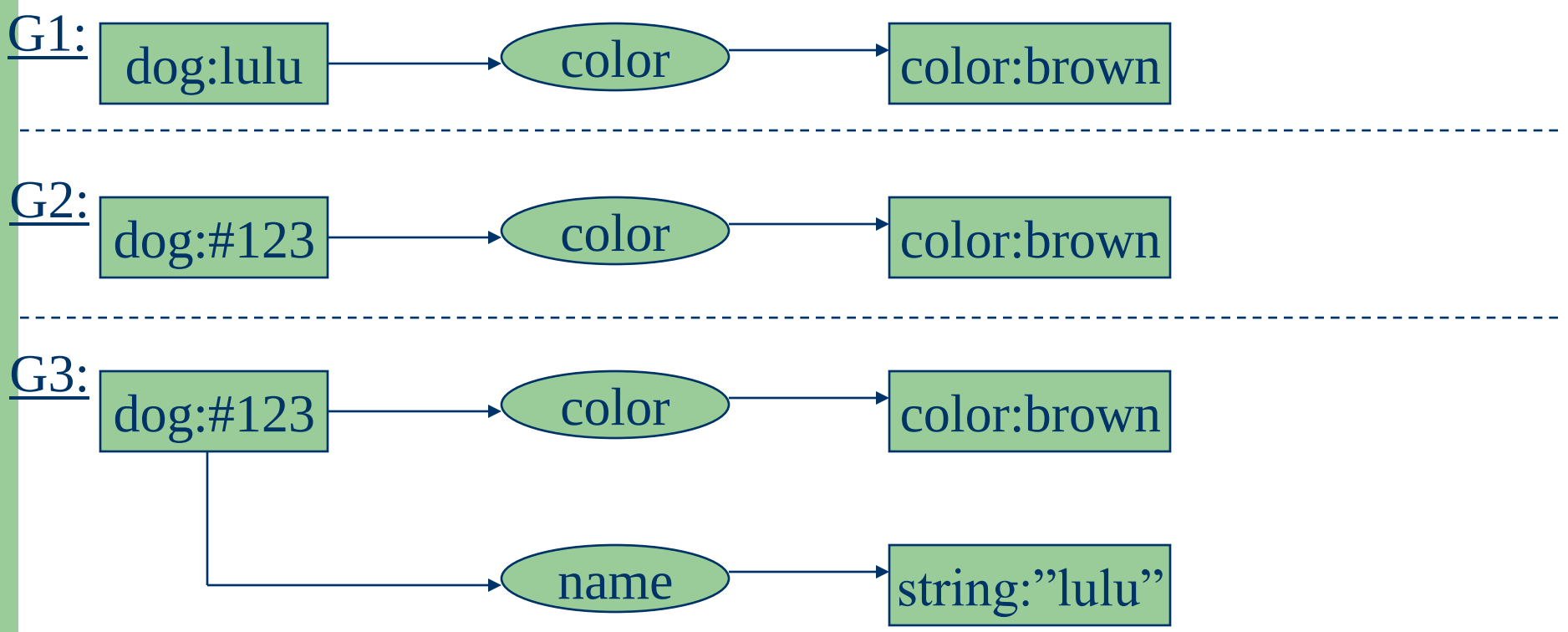
sinhviên:*X → sinh viên bất kỳ, tên sinh viên đã được lấy qua biến X.

sinhviên:ng* → sinh viên có tên bắt đầu bởi "ng"

Trường hợp 1 và 2, khái niệm được gọi là khái niệm cá thể, trường hợp 3 ta có khái niệm tổng quát.

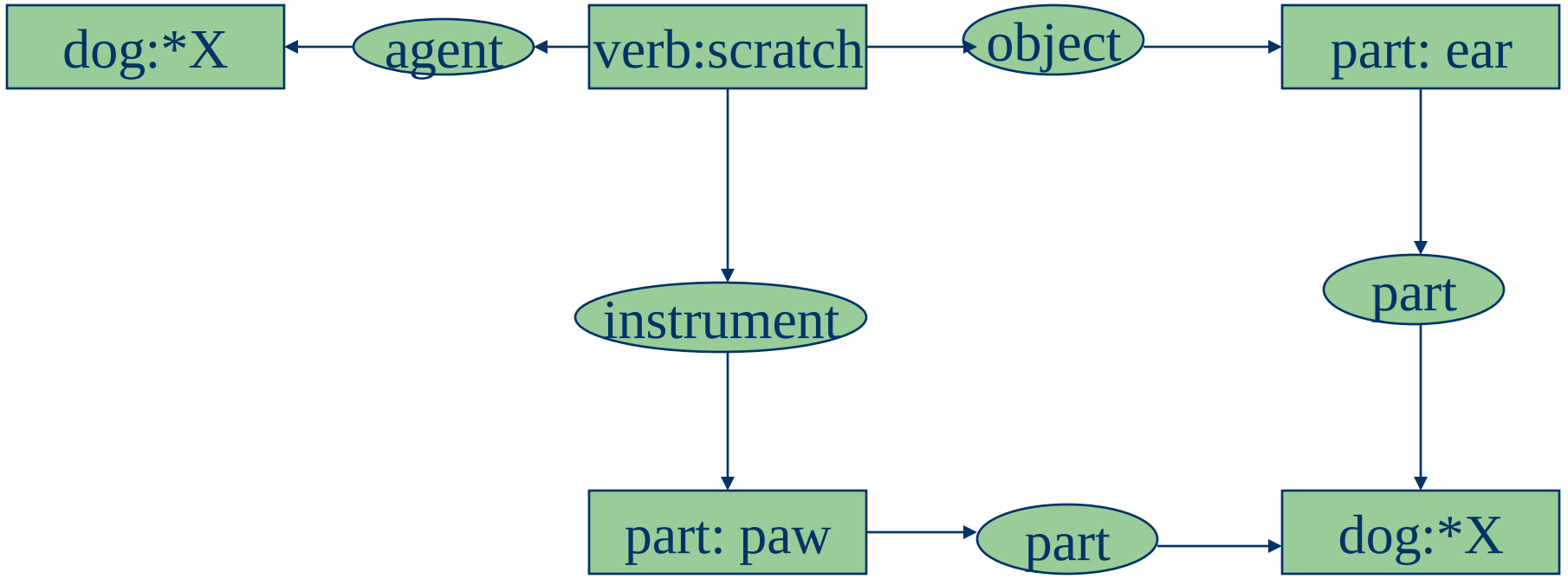
Đồ thị khái niệm

- Nếu dùng cách đặt tên như nói trên có thể nhín thấy 3 đồ thị sau có tác dụng biểu diễn như nhau nếu con có luu có khoá là #123.



Đồ thị khái niệm

- Biến có thể được dùng khi cần chỉ ra nhiều đỉnh khái niệm đồng nhất nhau trong một đồ thị như trường hợp sau.



Represent: "The dog scratches its ear with its paw"

Đồ thị khái niệm

- Phân cấp loại (type)

  Nếu có s và t là hai loại (type) thì:

  s ≤ t : → s: **subtype** của t

  → t : **supertype** của s

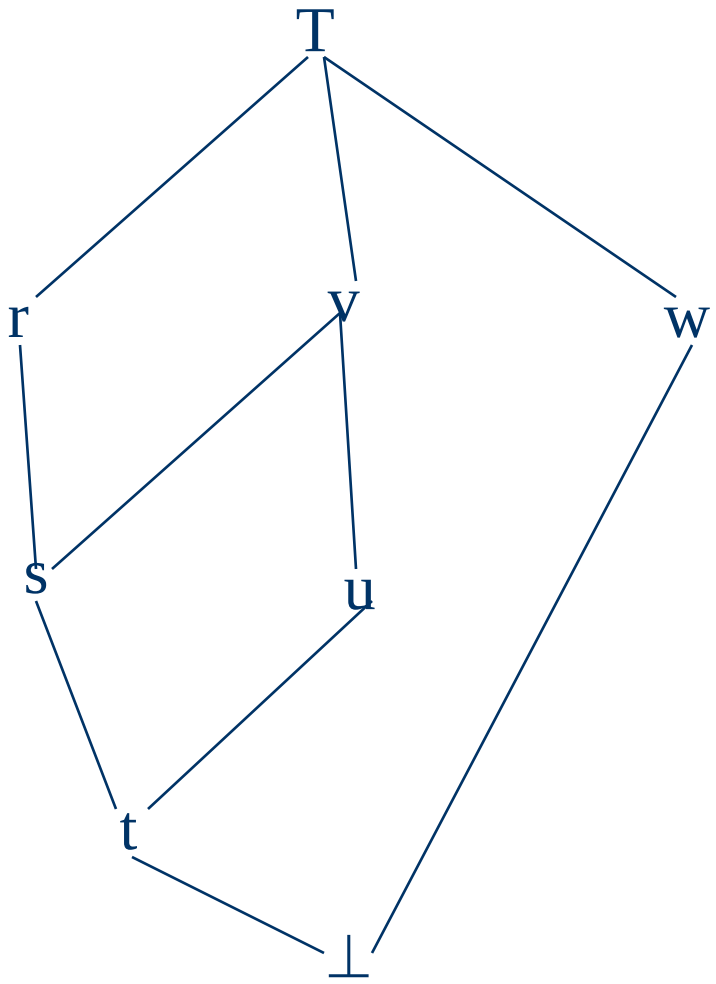
Ví dụ:

- sinhviên là subtype của người.
- người là super type của sinhviên.

→ nên viết: ***sinhviên ≤ người***

Trong sơ đồ phân cấp bên,

- s: được gọi là **common-subtype** của r và v.
- v : được gọi là **common-supertype** của s và u.
- T : supertype của mọi type
- ⊥ : subtype của mọi type

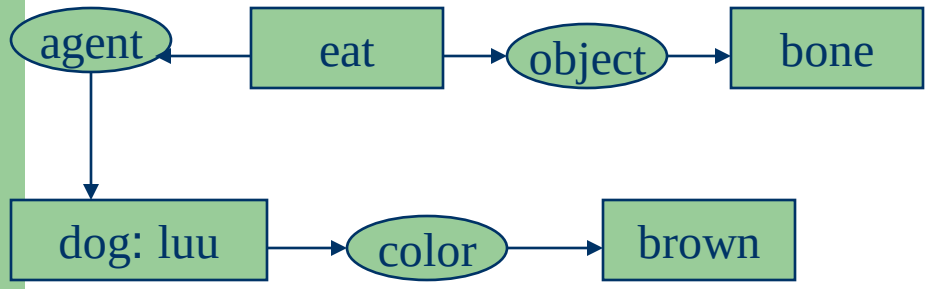


Đồ thị khái niệm

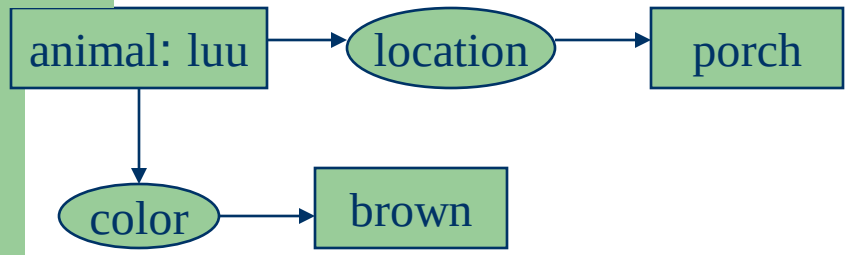
- Các phép toán trên đồ thị khái niệm.

  Xét hai đồ thị sau:

G1:



G2:



♣**Phép copy (nhân bản):** nhân bản một đồ thị.

♣**Phép Restriction (giới hạn):** tạo ra đồ thị mới bằng cách: từ một đồ thị đã có, thay thế một đỉnh khái niệm bởi một đỉnh khác cụ thể hơn, như hai trường hợp:

→ Một biến *, được thay thế bởi một khoá, hay một tên của cá thể.

VD: ***dog:*** → ***dog:#123*** hay ***dog:luu***

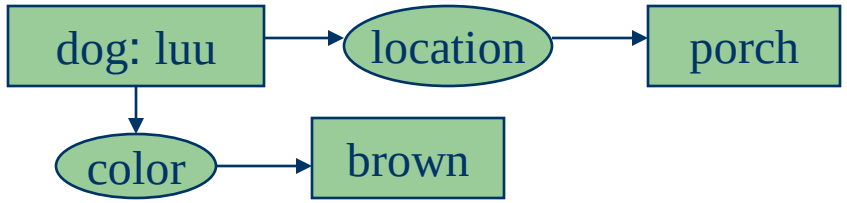
→ Một type được thay thế bởi subtype của nó.

VD: ***người: nam*** → ***sinhviên:nam***

Đồ thị khái niệm

Aùp dụng phép restriction trên đồ thị G2, có thể dẫn ra G3 như sau:

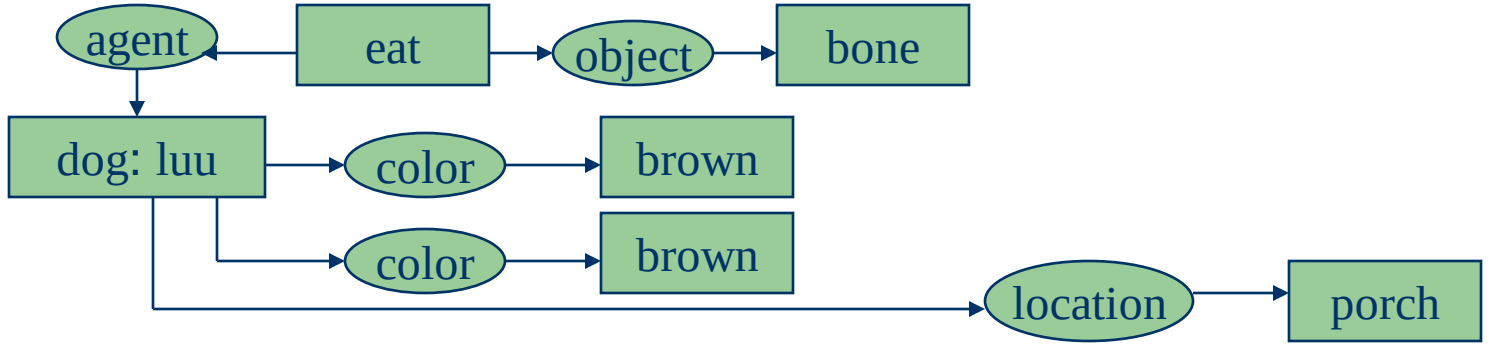
G3:



♣ **Phép Join (nối):** Nối hai đồ thị để được một đồ thị khác.

Nếu có đỉnh khái niệm C xuất hiện trên cả hai đồ thị X và Y, thì chúng ta có thể nối hai đồ thị trên đỉnh chung C nói trên, như từ G1 và G3 có thể tạo ra G4 như sau: (nối trên đỉnh chung là: *dog:lulu*)

G4:

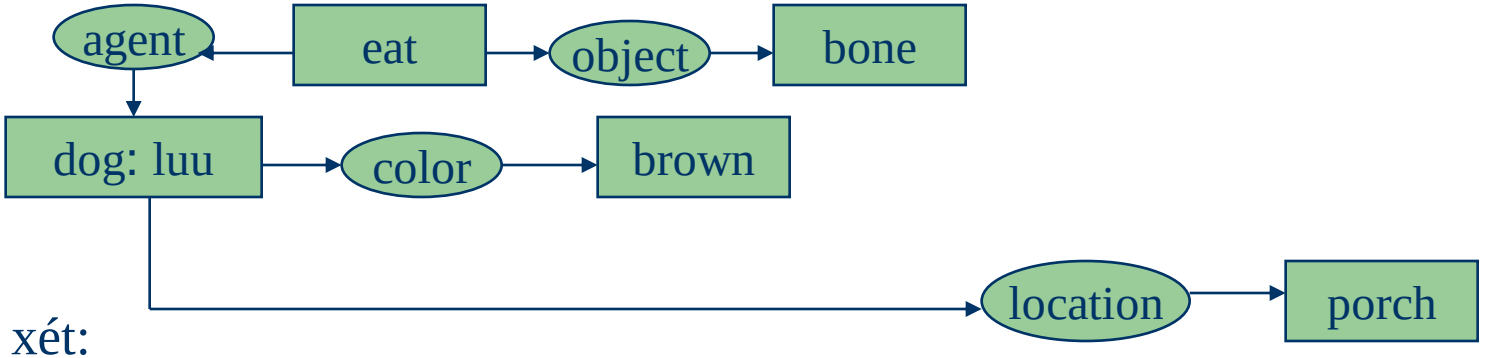


Đồ thị khái niệm

♣ **Phép simplify: (rút gọn)**

Nếu trên một đồ thị có hai đồ thị con giống nhau hoàn toàn thì chúng ta có thể bỏ đi một để tạo ra một đồ thị mới có khả năng biểu diễn không thay đổi. Từ G4 có thể sinh ra G5 cùng khả năng biểu diễn.

G5:



Nhận xét:

Phép Restriction và phép Join cho phép chúng ta thực hiện tính thừa kế trên đồ thị khái niệm. Khi thay một biến * bởi một cá thể cụ thể, lúc đó chúng ta cho phép cá thể thừa kế các tính chất từ loại(type) của nó, cũng tương tự khi ta thay thế một type bởi subtype của nó.

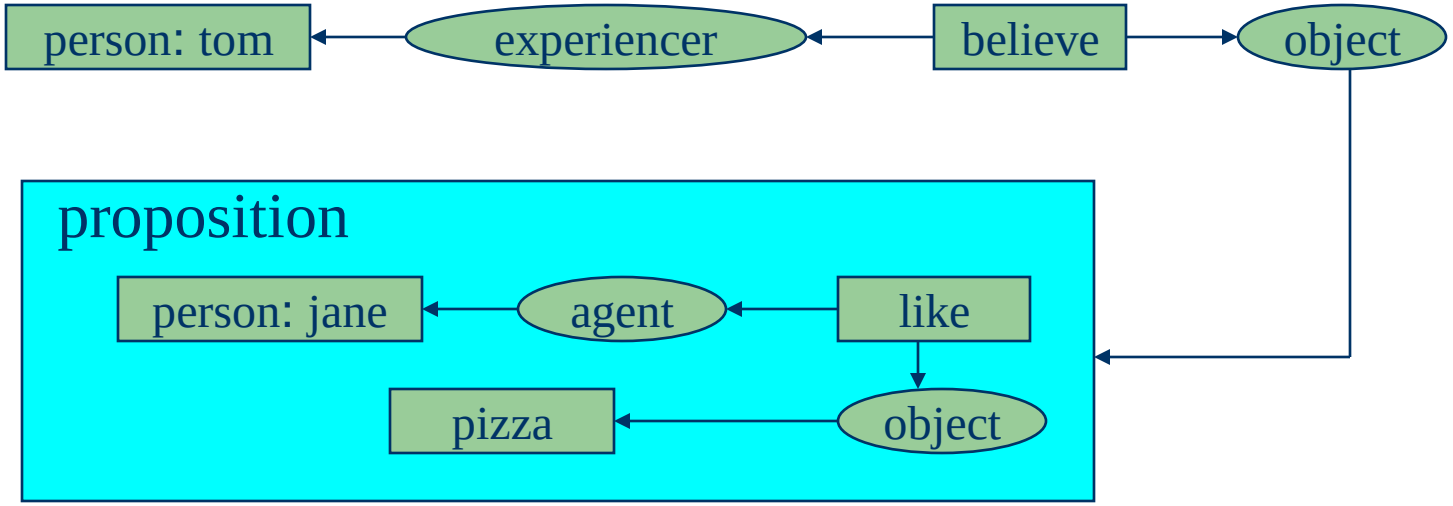
Đồ thị khái niệm

- Đỉnh mệnh đề:

  Để thuận tiện biểu diễn cho các câu gồm nhiều mệnh đề, đồ thị khái niệm đã được mở rộng để có thể chứa cả một mệnh đề trong một đỉnh khái niệm, lúc đó chúng ta gọi là đỉnh mệnh đề.

  Vậy đỉnh mệnh đề là một đỉnh khái niệm có chứa một đồ thị khái niệm khác. Xét đồ thị khái niệm mở rộng biểu diễn cho câu:

  "Tom believes that Jane likes pizza".

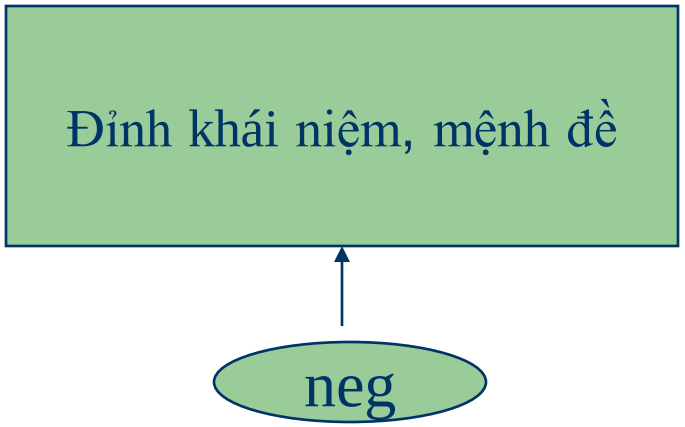


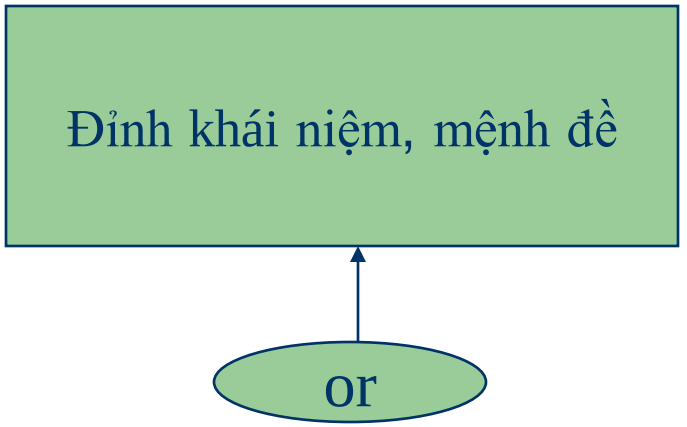
Đồ thị khái niệm

- Đồ thị khái niệm và logic.

  - Phép hội (and) của nhiều khái niệm, mệnh đề chúng ta có thể thực hiện dễ dàng cách cách nối nhiều đồ thị bởi phép toán join.

  - Phép phủ định(not) và phép tuyển(or) giữa các khái niệm hay mệnh đề cũng có thể được thể hiện bằng cách đưa vào đỉnh quan hệ có tên: neg(phủ định), or(tuyển) như dạng sau.

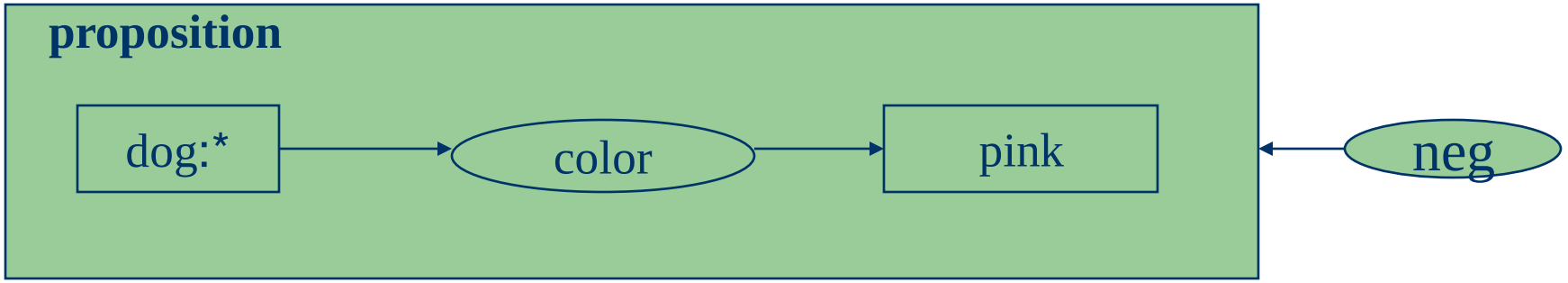




Đồ thị khái niệm

Ví dụ:

Câu: "There are no pink dogs", được biểu diễn:



Trong đồ thị khái niệm, các khái niệm tổng quát (đỉnh dùng biến * - như ***dog:****, hay chỉ có tên loại - như ***dog***) được xem như có lượng từ tồn tại (∃). Do vậy, mệnh đề trong ví dụ trên có biểu diễn vị từ là:

∃X∃Y(dog(X) ^ color(X,Y) ^ pink(Y)).

Và toàn bộ đồ thị (bao gồm đỉnh quan hệ :neg), có biểu diễn vị từ:

¬ ∃X∃Y(dog(X) ^ color(X,Y) ^ pink(Y)).

= **∀X∀Y(¬ (dog(X) ^ color(X,Y) ^ pink(Y))).**

Đồ thị khái niệm

- Giải thuật để chuyển một đồ thị khái niệm sang biểu diễn vị từ:

1. Gán một biến riêng biệt (X1, X2,…) cho mỗi khái niệm tổng quát.
2. Gán một hằng cho mỗi khái niệm cá thể trong đồ thị. Hằng này có thể là tên cá thể hay khoá của nó.
3. Biểu diễn một đỉnh khái niệm bởi một vị từ một ngôi; có tên là tên loại (type), đối số là biến hay hằng vừa gán trên.
4. Biểu diễn mỗi đỉnh quan hệ bởi một vị từ n ngôi; có tên là tên của đỉnh quan hệ, các thông số là biến hay hằng được gán cho các đỉnh khái niệm nối đến nó.
5. Hội của tất cả các câu trong bước 3 và 4. Tất cả các biến trong biểu thức thu được đều đính kèm lượng từ tồn tại.

Ví dụ: có đồ thị như sau

Được chuyển sang là:

∃X1(dog(luu) ^ color(X1) ^ brown(X1))

Lược đồ có cấu trúc - Frame

- Frame – khung.

  Là một cấu trúc dữ liệu cho phép biểu diễn tri thức ở dạng khái niệm hay đối tượng.

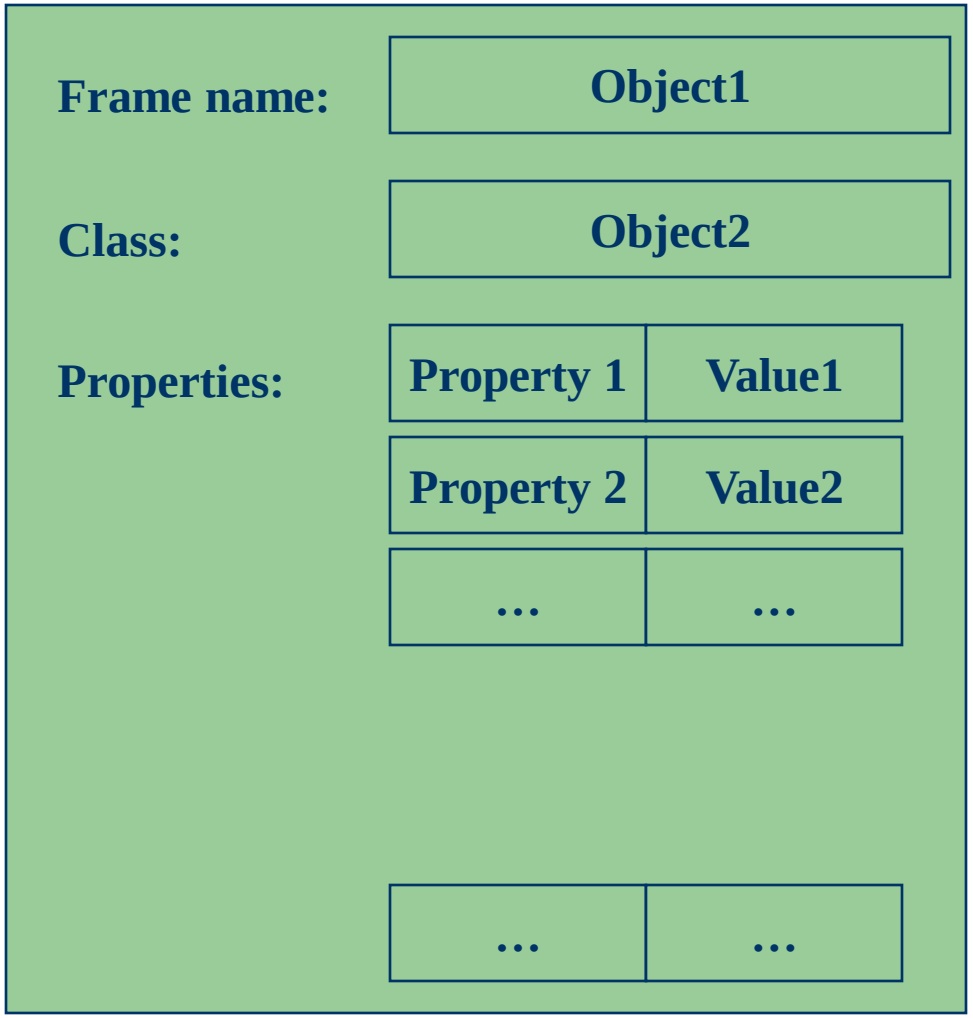
  Một khung có cấu trúc như hình vẽ bên.

- Cấu trúc của frame:

  Đặc tả cho một frame gồm các thành phần cơ bản sau:

  1. Frame name: tên của frame.

  - Nếu frame biểu diễn cho một cá thể nào đó, thì đây là tên của cá thể. Ví dụ: an, nam, lulu,..



Lược đồ có cấu trúc – Frame

- Cấu trúc của frame (tt):

- Nếu Frame biểu diễn cho một lớp, thì đây là tên lớp. Ví dụ: chim, động vật, …

2. Class: Tên loại.

- Nếu thành phần này xuất hiện, nó cho biết rằng frame mà chúng ta đang biểu diễn có loại là giá trị trường class. → ***Cho phép thành lập quan hệ thừa kế IS-A.***

Như ví dụ trên, chúng ta có:

Object1 IS-A Object2

3. Các thuộc tính (property): Khi biểu diễn một frame chúng ta có thể thiết lập một hay nhiều thuộc tính cho nó, như ví dụ sau:

Frame name:	**chim**
Properties:	
màu	**Chưa biết**
ăn	**Côn trùng**
Số cánh	**2**
bay	**true**
đói	**Chưa biết**
Hoạt động	**Chưa biết**

Lược đồ có cấu trúc – Frame

- Cấu trúc của frame (tt):

  - Khi chúng ta đặt tả thuộc tính cho một lớp; nếu chúng ta biết được giá trị chung cho tất cả các đối tượng thuộc lớp mà chúng ta đang biểu diễn thì điền vào trị cho thuộc tính đó, giá trị đó chúng ta gọi là ***giá trị mặc nhiên, như: ăn, số cánh ở trên*** ; nếu chúng ta chưa biết trị cụ thể (nhưng biết là có thuộc tính đó) thì chúng ta có thể bỏ trống ***(chưa biết) – như màu, hoạt động,..***:.

Các thuộc tính của frame nằm ở hai dạng cơ bản:

→ *Dạng tĩnh(static):* giá trị của nó không thay đổi trong quá trình hệ thống tri thức hoạt động.

→ *Dạng động(dynamic):* giá trị có thể chuyển đổi.

Khi phải tìm kiếm một frame, chúng ta có thể dựa vào ***frame name*** , cũng có thể dựa vào các thuộc tính được đặt tả cho frame.

Lược đồ có cấu trúc – Frame

- Cấu trúc của frame (tt):

4. *Các thủ tục:* Lược đồ frame càng cho phép tích hợp cách thức đặt như các thuộc tính như trên và các thủ tục vào một frame. Về hình thức, một thủ tục sẽ chiếm một khe tương tự như khe thuộc tính nói trên.

Thủ tục được dùng để: biểu diễn một hành động nào đó của đối tượng, điều khiển giá trị của thuộc tính như: kiểm tra ràng buộc về trị, kiểu, của thuộc tính mỗi khi cần trích, hay thay đổi nó.

Hai thủ tục phổ biến được đính kèm với một thuộc tính là:

IF_NEEDED và

IF_CHANGED.

→ *IF_NEEDED:*

Thủ tục này được thực thi mỗi khi chúng ta cần đến giá trị của thuộc tính (giống thủ tục GET trong VB).

Ví dụ: thủ tục sau (dạng if_needed) cho thuộc tính ***bay*** của frame ***chim*** nói trên.

```
If      self:số_cánh < 2
Then self:bay = false
If      self:số_cánh = 2
Then self:bay = true
```

Lược đồ có cấu trúc – Frame

- Cấu trúc của frame (tt):

→ *IF_CHANGED:*

Thủ tục này được thực thi mỗi khi giá trị của thuộc tính mà if_changed này được gắn vào thay đổi. (giống như SET, LET trong VB)

Ví dụ: gắn thủ tục sau cho thuộc tính ***đói*** của lớp ***chim*** nói trên.

If Seft:đói = true

Then Seft:hànhđộng = eating # seft:ăn

4. Các thông tin khác:

Một số khe khác của frame có thể chứa frame khác, link đến frame, mạng ngữ nghĩa, rules, hay các loại thông tin khác.

Chú ý: các ví dụ trên mô phỏng theo ngôn ngữ *Kappa PC*, trong đó, "Expert ***System - DurKin"***:

- Seft: từ khoá chỉ chính bản thân frame đang mô tả (như Me của VB, this của VC)

- # : dấu nối chuỗi(như & của VB, + của VC)

- Lược đồ frame cũng giống như các hệ thống hướng đối tượng. Chúng ta:

→ Có thể đặt tả frame lớp hay cá thể.

→ Có thể đặt tả tính thừa kế.

→ Mỗi khi tạo ra frame cá thể, có thể copy các thuộc tính, thủ tục của frame lớp; đồng thời có thể mở rộng thêm, hay định nghĩa lại một số thuộc tính, thủ tục.

Lược đồ có cấu trúc – Script

- Script – Kịch bản:

  Là một lược đồ biểu diễn có cấu trúc, dùng để biểu diễn một chuỗi các sự kiện trong một ngữ cảnh cụ thể. Nó như một phương tiện để tổ chức các phụ thuộc khái niệm – (đã giới thiệu trước) để mô tả một tình huống cụ thể.

  Script được dùng trong các hệ thống hiểu NNTN, tổ chức tri thức trong thành phần các tình huống mà hệ thống phải tìm hiểu.

- Cấu trúc của Script:

  1. Entry conditions:

  Các điều kiện phải true để script được gọi. Ví dụ: một cá nhân bị bệnh thì script nhập viện được gọi.

2. Results:

Kết quả thu được từ script khi nó hoàn thành.

3. Props:

Các đồ vật tham gia vào script, như: xe cứu thương, cán, bình oxy,…

4. Roles:

Các cá nhân tham gia vào script, như: bệnh nhân, bác sĩ, y tá, người nhà,…

5. Scenes:

Các cảnh chính trong script, như: di chuyển, cấp cứu, hồi sức,..

→ Một ví dụ về kịch bản đi nhà hàng như ví dụ sau:

Lược đồ có cấu trúc – Script

Script: RESTAURENT
Track: Coffe Shop
Entry conditions:
- S is hungry
- S has money

Results:
- S has less money
- O has more money
- S is not hungry
- S is pleased (optional)

Props: Tables, Menu, Food (F), Check, Money

Roles: Custumer (S), Waiter(W), Cook(C), Cashier(M), Owner(O)

Scene 1: (Entering)

S PTRANS S into restaurent.
S ATTEND eyes to tables
S MBUILD where to sit
S PTRANS S to table
S MOVE S to sitting position

Scene 2: (Ordering)

(Menu on table)
S PTRANS menu to S
(S ask for menu)
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'need menu' to W
W PTRANS W to menu

Lược đồ có cấu trúc – Script

W PTRANS W to table
W ATRANS menu to S

S MTRANS food list to S
(*) S MBUILD choice of F
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'I want F' to W

W PTRANS W to C
W MTRANS (ATRANS F) to C

C MTRANS 'no F' to W
W PTRANS W to S
W MTRANS 'no F' to S
(go back to *) or
(go to scene 4)

C DO (prepare F script)
to scene 3

Lược đồ có cấu trúc – Script

Scene 3: (Eating)

C ATRANS F to W
W ATRANS F to S
S INGEST F
(Option: return to scene 2 to order more;
otherwise: goto scene 4)

Scene 4: (exiting)

S MTRANS to W
W ATRANS check to S

W MOVE (write check) ;
W PTRANS W to S
W ATRANS check to S ;
S ATRANS tip to W
S PTRANS S to M;
S ATRANS money to M;
S PTRANS S to out of restaurent.

Thi & Kiểm tra

- Hình thức thi:
  - Câu hỏi trắc nghiệm
  - Một số bài tập
- Học viên được phép sử dụng tài liệu

Khoa Công Nghệ Thông Tin

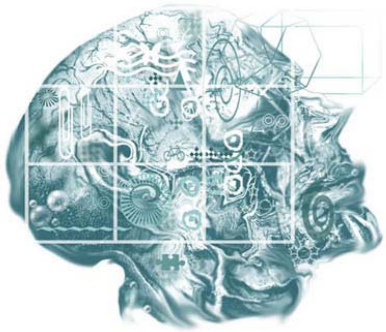
Đại Học Bách Khoa Tp.HCM

ThS Nguyễn Cao Trí caotri@dit.hcmut.edu.vn

Ks Lê Thành Sách ltsach@dit.hcmut.edu.vn

Giáo trình

Trí tuệ nhân tạo



Lưu hành nội bộ

Lời nói đầu

Trí tuệ nhân tạo (hay AI: Artificial Intelligence), là nỗ lực tìm hiểu những yếu tố trí tuệ. Lý do khác để nghiên cứu lĩnh vực này là cách để ta tự tìm hiểu bản thân chúng ta. Không giống triết học và tâm lý học, hai khoa học liên quan đến trí tuệ, còn AI cố gắng thiết lập các các yếu tố trí tuệ cũng như tìm biết về chúng. Lý do khác để nghiên cứu AI là để tạo ra các thực thể thông minh giúp ích cho chúng ta. AI có nhiều sản phẩm quan trọng và đáng lưu ý, thậm chí ngay từ lúc sản phẩm mới được hình thành. Mặc dù không dự báo được tương lai, nhưng rõ ràng máy tính điện tử với độ thông minh nhất định đã có ảnh hưởng lớn tới cuộc sống ngày nay và tương lai phát triển của văn minh nhân loại.

Tài liệu gồm các phần sau:

- Phần 1 : Tri thức và lập luận
- Phần 2 : Giải quyết vấn đề bằng tìm kiếm
- Phần 3 : Tiếp cận ký hiệu

Còn nhiều vấn đề khác chưa đề cập được trong phạm vi tài liệu này. Đề nghị các bạn đọc tìm hiểu thêm sau khi đã có những kiến thức cơ bản này.

Mục lục

Phần I

TRI THỨC VÀ LẬP LUẬN

Chương I

LOGIC MỆNH ĐỀ

Trong chương này chúng ta sẽ trình bày các đặc trưng của ngôn ngữ biểu diễn tri thức. Chúng ta sẽ nghiên cứu logic mệnh đề, một ngôn ngữ biểu diễn tri thức rất đơn giản, có khả năng biểu diễn hẹp, nhưng thuận lợi cho ta làm quen với nhiều khái niệm quan trọng trong logic, đặc biệt trong logic vị từ cấp một sẽ được nghiên cứu trong các chương sau.

I. Biểu diễn tri thức

Con người sống trong môi trường có thể nhận thức được thế giới nhờ các giác quan (tai, mắt và các bộ phận khác), sử dụng các tri thức tích luỹ được và nhờ khả năng lập luận, suy diễn, con người có thể đưa ra các hành động hợp lý cho công việc mà con người đang làm. Một mục tiêu của Trí tuệ nhân tạo ứng dụng là thiết kế các tác nhân thông minh (intelligent agent) cũng có khả năng đó như con người. Chúng ta có thể hiểu tác nhân thông minh là bất cứ cái gì có thể nhận thức được môi trường thông qua các bộ cảm nhận (sensors) và đưa ra hành động hợp lý đáp ứng lại môi trường thông qua bộ phận hành động (effectors). Các robots, các softbot (software robot), các hệ chuyên gia,... là các ví dụ về tác nhân thông minh. Các tác nhân thông minh cần phải có tri thức về thế giới hiện thực mới có thể đưa ra các quyết định đúng đắn.

Thành phần trung tâm của các tác nhân dựa trên tri thức (knowledge-based agent), còn được gọi là hệ dựa trên tri thức (knowledge-based system) hoặc đơn giản là hệ tri thức, là cơ sở tri thức. Cơ sở tri thức (CSTT) là một tập hợp các tri thức được biểu diễn dưới dạng nào đó. Mỗi khi nhận được các thông tin đưa vào, tác nhân cần có khả năng suy diễn để đưa ra các câu trả lời, các hành động hợp lý, đúng đắn. Nhiệm vụ này được thực hiện bởi bộ suy diễn. Bộ suy diễn là thành phần cơ bản khác của các hệ tri thức. Như vậy hệ tri thức bảo trì một CSTT và được trang bị một thủ tục suy diễn. Mỗi khi tiếp nhận được các sự kiện từ môi trường, thủ tục suy diễn thực hiện quá trình liên kết các sự kiện với các tri thức trong CSTT để rút ra các câu trả lời, hoặc các hành động hợp lý mà tác nhân cần thực hiện. Đương nhiên là, khi ta thiết kế một tác nhân giải quyết một vấn đề nào đó thì CSTT sẽ chứa các tri thức về miền đối tượng cụ thể đó. Để máy tính có thể sử dụng được tri thức, có thể xử lý tri thức, chúng ta cần biểu diễn tri thức dưới dạng thuận tiện cho máy tính. Đó là mục tiêu của biểu diễn tri thức.

Tri thức được mô tả dưới dạng các câu trong ngôn ngữ biểu diễn tri thức. Mỗi câu có thể xem như sự mã hóa của một sự hiểu biết của chúng ta về thế giới hiện thực. Ngôn ngữ biểu diễn tri thức (cũng như mọi ngôn ngữ hình thức khác) gồm hai thành phần cơ bản là cú pháp và ngữ nghĩa.

• Cú pháp của một ngôn ngữ bao gồm các ký hiệu về các quy tắc liên kết các ký hiệu (các luật cú pháp) để tạo thành các câu (công thức) trong ngôn ngữ. Các câu ở đây là biểu diễn ngoài, cần phân biệt với biểu diễn bên trong máy tính. Các câu sẽ được chuyển thành các cấu trúc dữ liệu thích hợp được cài đặt trong một vùng nhớ nào đó

của máy tính, đó là biểu diễn bên trong. Bản thân các câu chưa chứa đựng một nội dung nào cả, chưa mang một ý nghĩa nào cả.

• Ngữ nghĩa của ngôn ngữ cho phép ta xác định ý nghĩa của các câu trong một miền nào đó của thế giới hiện thực. Chẳng hạn, trong ngôn ngữ các biểu thức số học, dãy ký hiệu (x+y)*z là một câu viết đúng cú pháp. Ngữ nghĩa của ngôn ngữ này cho phép ta hiểu rằng, nếu x, y, z, ứng với các số nguyên, ký hiệu + ứng với phép toán cộng, còn * ứng với phép chia, thì biểu thức (x+y)*z biểu diễn quá trình tính toán: lấy số nguyên x cộng với số nguyên y, kết quả được nhân với số nguyên z.

• Ngoài hai thành phần cú pháp và ngữ nghĩa, ngôn ngữ biểu diễn tri thức cần được cung cấp cơ chế suy diễn. Một luật suy diễn (rule of inference) cho phép ta suy ra một công thức từ một tập nào đó các công thức. Chẳng hạn, trong logic mệnh đề, luật modus ponens từ hai công thức A và $A \Rightarrow B$ suy ra công thức B. Chúng ta sẽ hiểu lập luận hoặc suy diễn là một quá trình áp dụng các luật suy diễn để từ các tri thức trong cơ sở tri thức và các sự kiện ta nhận được các tri thức mới. Như vậy chúng ta xác định:

- Ngôn ngữ biểu diễn tri thức = Cú pháp + Ngữ nghĩa + Cơ chế suy diễn.
- Một ngôn ngữ biểu diễn tri thức tốt cần phải có khả năng biểu diễn rộng, tức là có thể mô tả được mọi điều mà chúng ta muốn nói. Nó cần phải hiệu quả theo nghĩa là, để đi tới các kết luận, thủ tục suy diễn đòi hỏi ít thời gian tính toán và ít không gian nhớ. Người ta cũng mong muốn ngôn ngữ biểu diễn tri thức gần với ngôn ngữ tự nhiên.
- Trong tài liệu này, chúng ta sẽ tập trung nghiên cứu logic vị từ cấp một (first-order predicate logic hoặc first-order predicate calculus) - một ngôn ngữ biểu diễn tri thức, bởi vì logic vị từ cấp một có khả năng biểu diễn tương đối tốt, và hơn nữa nó là cơ sở cho nhiều ngôn ngữ biểu diễn tri thức khác, chẳng hạn toán hoàn cảnh (situation calculus) hoặc logic thời gian khoảng cấp một (first-order interval tempral logic). Nhưng trước hết chúng ta sẽ nghiên cứu logic mệnh đề (propositional logic hoặc propositional calculus). Nó là ngôn ngữ rất đơn giản, có khả năng biểu diễn hạn chế, song thuận tiện cho ta đưa vào nhiều khái niệm quan trọng trong logic.

II. Cú pháp và ngữ nghĩa của logic mệnh đề

II.1 Cú pháp

Cú pháp của logic mệnh đề rất đơn giản, nó cho phép xây dựng nên các công thức. Cú pháp của logic mệnh đề bao gồm tập các ký hiệu và tập các luật xây dựng công thức.

- Các ký hiệu
  - • Hai hằng logic True và False.
  - • Các ký hiệu mệnh đề (còn được gọi là các biến mệnh đề): P, Q,...
  - • Các kết nối logic $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
  - • Các dấu mở ngoặc (và đóng ngoặc).
- Các quy tắc xây dựng các công thức
  - • Các biến mệnh đề là công thức.
  - • Nếu A và B là công thức thì:

($A \wedge B$) (đọc "A hội B" hoặc "A và B")

($A \vee B$) (đọc "A tuyển B" hoặc "A hoặc B")

($\neg A$) (đọc "phủ định A")

($A \Rightarrow B$) (đọc "A kéo theo B" hoặc "nếu A thì B")

($A \Leftrightarrow B$) (đọc "A và B kéo theo nhau")

là các công thức.

Sau này để cho ngắn gọn, ta sẽ bỏ đi các cặp dấu ngoặc không cần thiết. Chẳng hạn, thay cho $((A \vee B) \wedge C)$ ta sẽ viết là $(A \vee B) \wedge C$.

Các công thức là các ký hiệu mệnh đề sẽ được gọi là các câu đơn hoặc câu phân tử. Các công thức không phải là câu đơn sẽ được gọi là câu phức hợp. Nếu P là ký hiệu mệnh đề thì P và TP được gọi là literal, P là literal dương, còn TP là literal âm. Câu phức hợp có dạng $A_1 \vee ... \vee A_m$ trong đó A_i là các literal sẽ được gọi là câu tuyển (clause).

II.2 Ngữ nghĩa

Ngữ nghĩa của logic mệnh đề cho phép ta xác định thiết lập ý nghĩa của các công thức trong thế giới hiện thực nào đó. Điều đó được thực hiện bằng cách kết hợp mệnh đề với sự kiện nào đó trong thế giới hiện thực. Chẳng hạn, ký hiệu mệnh đề P có thể ứng với sự kiện "Paris là thủ đô nước Pháp" hoặc bất kỳ một sự kiện nào khác. Bất kỳ một sự kết hợp các kí hiệu mệnh đề với các sự kiện trong thế giới thực được gọi là một minh họa (interpretation). Chẳng hạn minh họa của kí hiệu mệnh đề P có thể là một sự kiện (mệnh đề) "Paris là thủ đô nước Pháp ". Một sự kiện chỉ có thể đúng hoặc sai. Chẳng hạn, sự kiện "Paris là thủ đô nước Pháp " là đúng, còn sự kiện "Số Pi là số hữu tỉ " là sai.

Một cách chính xác hơn, cho ta hiểu một minh họa là một cách gán cho mỗi ký hiệu mệnh đề một giá trị chân trị True hoặc False. Trong một minh họa, nếu kí hiệu mệnh đề P được gán giá trị chân trị True/False (P ←True/ P←False) thì ta nói mệnh đề P đúng/sai trong minh họa đó. Trong một minh họa, ý nghĩa của các câu phức hợp được xác định bởi ý nghĩa của các kết nối logic. Chúng ta xác định ý nghĩa của các kết nối logic trong các bảng chân trị (xem hình 1.1)

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q	P $\Leftrightarrow$ Q
False	False	True	False	False	True	True
False	True	True	False	True	True	Fasle
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Hình 1.1 Bảng chân trị của các kết nối logic

Ý nghĩa của các kết nối logic $\wedge$, $\vee$ và $\neg$ được xác định như các từ "và","hoặc là" và "phủ định" trong ngôn ngữ tự nhiên. Chúng ta cần phải giải thích thêm về ý nghĩa của phép kéo theo P $\Rightarrow$ Q (P kéo theo Q), P là giả thiết, còn Q là kết luận. Trực quan cho phép ta xem rằng, khi P là đúng và Q là đúng thì câu "P kéo theo Q " là đúng, còn khi

P là đúng Q là sai thì câu "P kéo theo Q" là sai. Nhưng nếu P sai và Q đúng , hoặc P sai Q sai thì "P kéo theo Q" là đúng hay sai ? Nếu chúng ta xuất phát từ giả thiết sai, thì chúng ta không thể khẳng định gì về kết luận. Không có lý do gì để nói rằng, nếu P sai và Q đúng hoặc P sai và Q sai thì "P kéo theo Q" là sai. Do đó trong trường hợp P sai thì "P kéo theo Q " là đúng dù Q là đúng hay Q là sai.

Bảng chân trị cho phép ta xác định ngẫu nhiên các câu phức hợp. Chẳng hạn ngữ nghĩa của các câu $P \wedge Q$ trong minh họa $\{P \leftarrow True , Q \leftarrow False \}$ là False. Việc xác định ngữ nghĩa của một câu $(P \vee Q) \wedge \neg S$ trong một minh họa được tiến hành như sau: đầu tiên ta xác định giá trị chân trị của $P \vee Q$ và $\neg S$, sau đó ta sử dụng bảng chân trị $\wedge$ để xác định giá trị $(P \vee Q) \wedge \neg S$.

- Một công thức là hợp lệ (valid) nếu và chỉ nếu nó đúng trong mọi minh hoạ chẳng hạn câu $P \vee \neg P$, ngược lại nó là không hợp lệ.
- Một công thức được gọi là thoả được (satisfiable) hay bền vững (consistent) nếu nó đúng trong một minh họa nào đó. Chẳng hạn công thức $(P \vee Q) \wedge \neg S$ là thoả được, vì nó có giá trị True trong minh họa $\{P \leftarrow True, Q \leftarrow False, S \leftarrow False\}$.
- Một công thức được gọi là không thoả được (insatisfiable) hay không bền vững (inconsistent), nếu nó là sai trong mọi minh họa (mâu thuẫn), chẳng hạn công thức $P \wedge \neg P$.

Chúng ta sẽ gọi một mô hình (model) của một công thức là một minh họa sao cho công thức là đúng trong minh họa này. Như vậy một công thức thoả được là công thức có một mô hình. Chẳng hạn, minh họa $\{P \leftarrow False , Q \leftarrow False , S \leftarrow True \}$ là một mô hình của công thức $(P \Rightarrow Q) \wedge S$.

Bằng cách lập bảng chân trị (phương pháp bảng chân trị) là ta có thể xác định được một công thức có thoả được hay không. Trong bảng này, mỗi biến mệnh đề đứng đầu với một cột, công thức cần kiểm tra đứng đầu một cột, mỗi dòng tương ứng với một minh họa. Chẳng hạn hình 1.2 là bảng chân trị cho công thức $(P \Rightarrow Q) \wedge S$. Trong bảng chân trị này ta cần đưa vào các cột phụ ứng với các công thức con của các công thức cần kiểm tra để việc tính giá trị của công thức này được dễ dàng. Từ bảng chân trị ta thấy rằng công thức $(P \Rightarrow Q) \wedge S$ là thoả được nhưng không hợp lệ và.

P	Q	S	$P \Rightarrow Q$	$(P \Rightarrow Q) \wedge S$
False	False	False	True	False
False	False	True	True	True
False	True	False	True	False
False	True	True	True	True
True	False	False	False	False
True	False	True	False	False
True	True	False	True	False
True	True	True	True	True

Hình 1.2 Bảng chân trị cho công thức $(P \Rightarrow Q) \wedge S$

Cần lưu ý rằng, một công thức chứa n biến, thì số các minh họa của nó là 2^n , tức là bảng chân trị có 2^n dòng. Như vậy việc kiểm tra một công thức có thoả được hay không bằng phương pháp bảng chân trị, đòi hỏi thời gian mũ. Cook (1971) đã chứng minh rằng, vấn đề kiểm tra một công thức trong logic mệnh đề có thoả được hay không là vấn đề NP-đầy đủ.

Chúng ta sẽ nói rằng thoả được/ không thoả được nếu hội của chúng $G_1 \wedge ..... \wedge G_m$ là vững chắc (thoả được, không thoả được). Một mô hình của tập công thức G là mô hình của tập công thức $G_1 \wedge ..... \wedge G_m$.

Định nghĩa: Cho các công thức $F_1, F_2,.., F_n$ và công thức G, G là hệ quả logic của $F_1, F_2,.., F_n$ nếu và chỉ nếu cho bất kỳ minh hoạ I nào trong đó $F_1 \wedge F_2 \wedge ... \wedge F_n$ là đúng thì G cũng đúng, $F_1, F_2,.., F_n$ là tiền đề của G.

Định lý: Cho các công thức $F_1, F_2,.., F_n$ và công thức G, G là hệ quả logic của $F_1, F_2,.., F_n$ nếu và chỉ nếu công thức $(F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G$ là hợp lệ.

Định lý: Cho các công thức $F_1, F_2,.., F_n$ và công thức G, G là hệ quả logic của $F_1, F_2,.., F_n$ nếu và chỉ nếu công thức $(F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G)$ là không bền vững.

Ví dụ: Nếu quốc hội từ chối ban hành luật mới thì tổng thống từ chức và cuộc đình công kéo dài hơn một tuần trừ khi nó chấm dứt ngay. Câu hỏi rằng liệu cuộc đình công sẽ chấm dứt ngay nếu quốc hội từ chối ban hành và cuộc đình công không kéo dài hơn một tuần?

Ta đặt:

P: quốc hội từ chối ban hành.

Q: tổng thống từ chức.

R: cuộc đình công kéo dài hơn một tuần.

S: cuộc đình công chấm dứt ngay.

Ta có

F_1: $P \rightarrow (Q \wedge R) \vee S \equiv$ Nếu quốc hội từ chối ban hành luật mới thì tổng thống từ chức và cuộc đình công kéo dài hơn một tuần trừ khi nó chấm dứt ngay.

F_2: $P \equiv$ Quốc hội từ chối ban hành.

F_3: $\neg R \equiv$ Cuộc đình công không kéo dài hơn một tuần.

Ta chứng minh rằng S là hệ quả logic của F_1, F_2 và F_3.

III. Dạng chuẩn tắc

Trong mục này chúng ta sẽ xét việc chuẩn hóa các công thức, đưa các công thức về dạng thuận lợi cho việc lập luận, suy diễn. Trước hết ta sẽ xét các phép biến đổi tương đương. Sử dụng các phép biển đổi này, ta có thể đưa một công thức bất kỳ về các dạng chuẩn tắc.

- Sự tương đương của các công thức

Hai công thức A và B được xem là tương đương nếu chúng có cùng một giá trị chân trị trong mọi minh họa. Để chỉ A tương đương với B ta viết A≡ B bằng phương pháp bảng chân trị, dễ dàng chứng minh được sự tương đương của các công thức sau đây :

- $A \Rightarrow B \equiv \neg A \vee B$
- $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
- $\neg (\neg A) \equiv A$

- Luật De Morgan
  - $\neg(A \vee B) \equiv \neg A \wedge \neg B$
  - $\neg(A \wedge B) \equiv \neg A \vee \neg B$
- Luật giao hoán
  - $A \vee B \equiv B \vee A$
  - $A \wedge B \equiv B \wedge A$
- Luật kết hợp
  - $(A \vee B) \vee C \equiv A \vee (B \vee C)$
  - $(A \vee B) \vee C \equiv A \vee (B \vee C)$
- Luật phân phối
  - $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
  - $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

➢ **Dạng chuẩn tắc :**

Các công thức tương đương có thể xem như các biểu diễn khác nhau của cùng một sự kiện. Để dễ dàng viết các chương trình máy tính thao tác trên các công thức, chúng ta sẽ chuẩn hóa các công thức, đưa chúng về dạng biểu diễn chuẩn được gọi là dạng chuẩn hội. Một công thức ở dạng chuẩn hội, có dạng $A_1 \wedge ... . \wedge A_m$ trong đó các A_i là literal. Chúng ta có thể biến đổi một công thức bất kỳ về công thức ở dạng chuẩn hội bằng cách áp dụng các thủ tục sau.

• Bỏ các dấu kéo theo ($\Rightarrow$) bằng cách thay $(A \Rightarrow B)$ bởi $(\neg A \vee B)$.

• Chuyển các dấu phủ định ($\neg$) vào sát các kết hiệu mệnh đề bằng cách áp dụng luật De Morgan và thay $\neg(\neg A)$ bởi A.

• Áp dụng luật phân phối, thay các công thức có dạng $A \vee (B \wedge C)$ bởi $(A \vee B) \wedge (A \vee B)$.

Ví dụ : Ta chuẩn hóa công thức $(P \Rightarrow Q) \vee \neg(R \vee \neg S)$:

$(P \Rightarrow Q) \vee \neg(R \vee \neg S) \equiv (\neg P \vee Q) \vee (\neg R \wedge S) \equiv ((\neg P \vee Q) \vee \neg R) \wedge ((\neg P \vee Q) \vee S) \equiv (\neg P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee S)$. Như vậy công thức $(P \Rightarrow Q) \vee \neg(R \vee \neg S)$ được đưa về dạng chuẩn hội $(\neg P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee S)$.

Khi biểu diễn tri thức bởi các công thức trong logic mệnh đề, cơ sở tri thức là một tập nào đó các công thức. Bằng cách chuẩn hoá các công thức, cơ sở tri thức là một tập nào đó các câu tuyển.

➢ **Các câu Horn :**

Ở trên ta đã chỉ ra, mọi công thức đều có thể đưa về dạng chuẩn hội, tức là các hội của các tuyển, mỗi câu tuyển có dạng

$\neg P_1 \vee ..... \vee \neg P_m \vee Q_1 \vee ..... Q_n$

trong đó P_i , Q_i là các ký hiệu mệnh đề (literal dương) câu này tương đương với câu

$P_1 \wedge ..... \wedge P_m \Rightarrow Q_1 \vee ..... \vee Q_n \approx p_1 \wedge .... \wedge p_m \Rightarrow Q$

Dạng câu này được gọi là câu Kowalski (do nhà logic Kowalski đưa ra năm 1971).

Khi $n \leq 1$, tức là câu Kowalski chỉ chứa nhiều nhất một literal dương ta có dạng một câu đặc biệt quan trọng được gọi là câu Horn (mang tên nhà logic Alfred Horn năm 1951).

Nếu m>0, n=1, câu Horn có dạng :

$P_1 \wedge ..... \wedge P_m \Rightarrow Q$

Trong đó P_i , Q là các literal dương. Các P_i được gọi là các điều kiện (hoặc giả thiết), còn Q

được gọi là kết luận (hoặc hệ quả). Các câu Horn dạng này còn được gọi là các luật if ... then và được biểu diễn như sau :

If P_1 and ....and P_m then Q.

Khi m=0, n=1 câu Horn trở thành câu đơn Q, hay sự kiện Q. Nếu m>0, n=0 câu Horn trở thành dạng $\neg P_1 \vee ..... \vee \neg P_m$ hay tương đương $\neg(P_1 \wedge ... \wedge P_m)$. Cần chú ý rằng, không phải mọi công thức đều có thể biểu diễn dưới dạng hội của các câu Horn. Tuy nhiên trong các ứng dụng, cơ sở tri thức thường là một tập nào đó các câu Horn (tức là một tập nào đó các luật if-then).

IV. Luật suy diễn

Một công thức H được xem là hệ quả logic (logical consequence) của một tập công thức $G=\{G_1,.....,G_m\}$ nếu trong bất kỳ minh họa nào mà $\{G_1,.....,G_m\}$ đúng thì H cũng đúng, hay nói cách khác bất kỳ một mô hình nào của G cũng là mô hình của H.

Khi có một cơ sở tri thức, ta muốn sử dụng các tri thức trong cơ sở này để suy ra tri thức mới mà nó là hệ quả logic của các công thức trong cơ sở tri thức. thực hiện bằng các thực hiện các luật suy diễn (rule of inference). Luật suy diễn giống như một thủ tục mà chúng ta sử dụng để sinh ra một công thức mới từ các công thức đã có. Một luật suy diễn gồm hai phần: một tập các điều kiện và một kết luận. Chúng ta sẽ biểu diễn các luật suy diễn dưới dạng "phân số ", trong đó tử số là danh sách các điều kiện, còn mẫu số là kết luận của luật, tức là mẫu số là công thức mới được suy ra từ các công thức ở tử số.

Sau đây là một số luật suy diễn quan trọng trong logic mệnh đề. Trong các luật này α, αi , β, γ là các công thức :

- Luật Modus Ponens

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

Từ một kéo theo và giả thiết của kéo theo, ta suy ra kết luận của nó.

- Luật Modus Tollens

$$\frac{\alpha \Rightarrow \beta, \neg\beta}{\neg\alpha}$$

Từ một kéo theo và phủ định kết luận của nó, ta suy ra phủ định giả thiết của kéo theo.

- Luật bắc cầu

$$\frac{\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\alpha \Rightarrow \gamma}$$

Từ hai kéo theo, mà kết luận của nó là của kéo theo thứ nhất trùng với giả thiết của kéo theo thứ hai, ta suy ra kéo theo mới mà giả thiết của nó là giả thiết của kéo theo thứ nhất, còn kết luận của nó là kết luận của kéo theo thứ hai.

- Luật loại bỏ hội

$$\frac{\alpha_1 \wedge ..... \wedge \alpha_i \wedge ..... \wedge \alpha_m}{\alpha_i}$$

Từ một hội ta đưa ra một nhân tử bất kỳ của hội.

- Luật đưa vào hội

$$\frac{\alpha_1,.....,\alpha_i,.....\alpha_m}{\alpha_1 \wedge ..... \wedge \alpha_i \wedge ..... \wedge \alpha_m}$$

Từ một danh sách các công thức, ta suy ra hội của chúng.

- Luật đưa vào tuyển

$$\frac{\alpha_i}{\alpha_1 \vee ..... \vee \alpha_i. \vee ..... \vee \alpha_m}$$

Từ một công thức, ta suy ra một tuyển mà một trong các hạng tử của các tuyển là công thức đó.

- Luật giải

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Từ hai tuyển, một tuyển chứa một hạng tử đối lập với một hạng tử trong tuyển kia, ta suy ra tuyển của các hạng tử còn lại trong cả hai tuyển.

Một luật suy diễn được xem là tin cậy (secured) nếu bất kỳ một mô hình nào của giả thiết của luật cũng là mô hình kết luận của luật. Chúng ta chỉ quan tâm đến các luật suy diễn tin cậy.

Bằng phương pháp bảng chân trị, ta có thể kiểm chứng được các luật suy diễn nêu trên đều là tin cậy. Bảng chân trị của luật giải được cho trong hình 1.3. Từ bảng này ta thấy rằng , trong bất kỳ một minh họa nào mà cả hai giả thiết $\alpha \vee \beta$, $\neg\beta \vee \gamma$ đúng thì kết luận $\alpha \vee \gamma$ cũng đúng. Do đó luật giải là luật suy điễn tin cậy.

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	False	False	False	True	False
False	False	True	False	True	True
False	True	False	True	False	False
False	True	True	True	True	True
True	False	False	True	True	True
True	False	True	True	True	True
True	True	False	True	False	True
True	True	True	True	True	True

Hình 1.3 Bảng chân trị chứng minh tính tin cậy của luật giải.

Ta có nhận xét rằng, luật giải là một luật suy diễn tổng quát, nó bao gồm luật Modus Ponens, luật Modus Tollens, luật bắc cầu như các trường hợp riêng.

- **Tiên đề định lý chứng minh**

Giả sử chúng ta có một tập nào đó các công thức. Các luật suy diễn cho phép ta từ các công thức đã có suy ra công thức mới bằng một dãy áp dụng các luật suy diễn. Các công thức đã cho được gọi là các tiên đề. Các công thức được suy ra được gọi là các định lý. Dãy các luật được áp dụng để dẫn tới định lý được gọi là một chứng minh của định lý. Nếu các luật suy diễn là tin cậy, thì các định lý là hệ quả logic của các tiên đề.

Ví dụ: Giả sử ta có các công thức sau :

$Q \wedge S \Rightarrow G \wedge H$ (1)

$P \Rightarrow Q$ (2)

$R \Rightarrow S$ (3)

P (4)

R (5)

Từ công thức (2) và (4), ta suy ra Q (Luật Modus Ponens) . Lại áp dụng luật Modus Ponens, từ (3) và (5) ta suy ra S . Từ Q, S ta suy ra $Q \wedge S$ (luật đưa vào hội). Từ (1) và $Q \wedge S$ ta suy ra $G \vee H$. Công thức $G \vee H$ đã được chứng minh.

Trong các hệ tri thức, chẳng hạn các hệ chuyên gia, hệ lập trình logic,..., sử dụng các luật suy diễn người ta thiết kế lên các thủ tục suy diễn (còn được gọi là thủ tục chứng minh) để từ các tri thức trong cơ sở tri thức ta suy ra các tri thức mới đáp ứng nhu cầu của người sử dụng.

Một hệ hình thức (formal system) bao gồm một tập các tiên đề và một tập các luật suy diễn nào đó (trong ngôn ngữ biểu diễn tri thức nào đó).

Một tập luật suy diễn được xem là đầy đủ, nếu mọi hệ quả logic của một tập các tiên đề đều chứng minh được bằng cách chỉ sử dụng các luật của tập đó.

- **Phương pháp chứng minh bác bỏ**

Phương pháp chứng minh bác bỏ (refutation proof hoặc proof by contradiction) là một phương pháp thường xuyên được sử dụng trong các chứng minh toán học. Tư tưởng của phương pháp này là như sau: Để chứng minh P đúng, ta giả sử P sai (thêm $\neg P$ vào các giả thiết) và dẫn tới một mâu thuẫn. Sau đây ta sẽ trình bầy cơ sở này.

Giả sử chúng ta có một tập hợp các công thức $G = \{G_1,.....,G_m\}$ ta cần chứng minh công thức H là hệ quả logic của G . Điều đó tương đương với chứng minh công thức $G_1 \wedge .... \wedge G_m \Rightarrow H$ là vững chắc. Thay cho chứng minh $G_1 \wedge ..... \wedge G_m \Rightarrow H$ là vững chắc, ta chứng minh $G_1 \wedge .... \wedge G_m \wedge \neg H$ là không thỏa mãn được.

Tức là ta chứng minh tập $G' = (G_1,.....,G_m,\neg H)$ là không thỏa được nếu từ G'ta suy ra hai mệnh đề đối lập nhau. Việc chứng minh công thức H là hệ quả logic của tập các tiêu đề G bằng cách chứng minh tính không thỏa được của tập các tiêu đề được thêm vào phủ định của công thức cần chứng minh, được gọi là chứng minh bác bỏ.

V. Luật giải, chứng minh bác bỏ bằng luật giải

Để thuận tiện cho việc sử dụng luật giải, chúng ta sẽ cụ thể hoá luật giải trên các dạng câu đặc biệt quan trọng.

•Luật giải trên các câu tuyển

$A_1 \vee ... \vee A_m \vee C$

$\neg C \vee B_1 ... \vee B_n$

$A_1 \vee ... \vee A_m \vee B_1 \vee ... \vee B_n$

trong đó A_i, B_j và C là các literal.

•Luật giải trên các câu Horn

Giả sử P_i, R_j, Q và S là các literal. Khi đó ta có các luật sau :

$P_1 \wedge ... \wedge P_m \wedge S \Rightarrow Q$,

$R_1 \wedge ... \wedge R_n \Rightarrow S$

$P_1 \wedge ... \wedge P_m \wedge R_1 \wedge ... \wedge R_n \Rightarrow Q$

Một trường hợp riêng hay được sử dụng của luật trên là :

$P_1 \wedge ... \wedge P_m \wedge S \Rightarrow Q$,

S

$P_1 \wedge ... \wedge P_m \Rightarrow Q$

Khi ta có thể áp dụng luật giải cho hai câu, thì hai câu này được gọi là hai câu giải được và kết quả nhận được khi áp dụng luật giải cho hai câu đó được gọi là giải thức của chúng. Giải thức của hai câu A và B được kí hiệu là res(A, B). Chẳng hạn, hai câu tuyển giải được nếu một câu chứa một literal đối lập với một literal trong câu kia. Giải thức của hai literal đối lập nhau (P và $\neg P$) là câu rỗng, chúng ta sẽ ký hiệu câu rỗng là [] , câu rỗng không thoả được.

Giả sử G là một tập các câu tuyển (Bằng cách chuẩn hoá ta có thể đưa một tập các công thức về một tập các câu tuyển). Ta sẽ ký hiệu R(G) là tập câu bao gồm các câu thuộc G và tất cả các câu nhận được từ G bằng một dãy áp dụng luật giải.

Luật giải là luật đầy đủ để chứng minh một tập câu là không thỏa được. Điều này được suy từ định lý sau :

Định lý giải:

Một tập câu tuyển là không thỏa được nếu và chỉ nếu câu rỗng [] $\in$ R(G).

Định lý giải có nghĩa rằng, nếu từ các câu thuộc G, bằng cách áp dụng luật giải ta dẫn tới câu rỗng thì G là không thỏa được, còn nếu không thể sinh ra câu rỗng bằng luật giải thì G thỏa được. Lưu ý rằng, việc dẫn tới câu rỗng có nghĩa là ta đã dẫn tới hai literal đối lập nhau P và ¬P (tức là dẫn tới mâu thuẫn).

Từ định lý giải, ta đưa ra thủ tục sau đây để xác định một tập câu tuyển G là thỏa được hay không. Thủ tục này được gọi là thủ tục giải.

Procedure Resolution ;

Input : tập G các câu tuyển ;

Begin

1.**Repeat**

1.1 Chọn hai câu A và B thuộc G ;

1.2 **if** A và B giải được **then** tính Res (A,B) ;

1.3 **if** Res (A,B) là câu mới **then** thêm Res (A,B) vào G ;

until nhận được [] hoặc không có câu mới xuất hiện ;

2. **if** nhận được câu rỗng **then** thông báo G không thoả được

else thông báo G thoả được ;

end;

Chúng ta có nhận xét rằng, nếu G là tập hữu hạn các câu thì các literal có mặt trong các câu của G là hữu hạn. Do đó số các câu tuyển thành lập được từ các literal đó là hữu hạn. Vì vậy chỉ có một số hữu hạn câu được sinh ra bằng luật giải. Thủ tục giải sẽ dừng lại sau một số hữu hạn bước.

Chỉ sử dụng luật giải ta không thể suy ra mọi công thức là hệ quả logic của một tập công thức đã cho. Tuy nhiên, sử dụng luật giải ta có thể chứng minh được một công thức bất kì có là hệ quả của một tập công thức đã cho hay không bằng phương pháp chứng minh bác bỏ. Vì vậy luật giải được xem là luật đầy đủ cho bác bỏ.

Sau đây là thủ tục chứng minh bác bỏ bằng luật giải

Procedure Refutation_Proof ;

Input : Tập G các công thức ;

Công thức cần chứng minh H;

Begin

1. Thêm ¬H vào G ;

2. Chuyển các công thức trong G về dạng chuẩn hội ;

3. Từ các dạng chuẩn hội ở bước hai, thành lập tạp các câu tuyển G' ;

4. Áp dụng thủ tục giải cho tập câu G' ;

5. if G' không thoả được then thông báo H là hệ quả logic

else thông báo H không là hệ quả logic của G ;

end;

Ví dụ: Giả giử G là tập hợp các câu tuyển sau :

$\neg A \vee \neg B \vee P$ (1)

$\neg C \vee \neg D \vee P$ (2)

$\neg E \vee C$ (3)

A (4)

E (5)

D (6)

Giả sử ta cần chứng minh P. Thêm vào G câu sau:

$\neg P$ (7)

Áp dụng luật giải cho câu (2) và (7) ta được câu:

$\neg C \vee \neg D$ (8)

Từ câu (6) và (8) ta nhận được câu:

$\neg C$ (9)

Từ câu (3) và (9) ta nhận được câu:

$\neg E$ (10)

Tới đây đã xuất hiện mâu thuẫn, vì câu (5) và (10) đối lập nhau. Từ câu (5) và (10) ta nhận được câu rỗng []. Vậy P là hệ quả logic của các câu (1) -- (6).

VI. Bài tập

1. Thể hiện các câu sau bằng mệnh đề logic
   a. Một quan hệ là tương đương nếu và chỉ nếu nó phản xạ, bắt cầu và đối xứng.
   b. Nếu độ ẩm quá cao thì trời sẽ mưa vào buổi chiều hay buổi tối.
   c. Ung thư sẽ không chữa trị được trừ khi tìm ra nguyên nhân và thuốc điều trị mới.
   d. Để leo lên núi cao thì đòi hỏi sự dũng cảm và kỹ năng leo núi.
   e. Nếu ông ta vận động nhiều thì có thể sẽ được bầu.
2. Đặt

   P : Ông ta cần một bác sĩ Q : Ông ta cần một luật sư

R : Ông ta gặp tai nạn S : Ông ta bị bệnh

U : Ông ta bị thương

Phát biểu ý nghĩa các công thức sau:

$(S \rightarrow P) \wedge (R \rightarrow Q)$ $P \rightarrow (S \vee U)$ $(P \wedge Q) \rightarrow R$

$(P \wedge Q) \leftrightarrow (S \wedge U)$ $\neg (S \vee U) \rightarrow \neg P$

3. Tạo bảng chân trị của công thức : $(\neg P \vee Q) \wedge (\neg (P \wedge \neg Q))$
4. Cho các công thức sau, xác định công thức nào là hợp lệ, không hợp lệ, bền vững, không bền vững hoặc kết hợp các tính chất trên:
   a. $\neg (\neg P) \rightarrow P$
   b. $P \rightarrow (P \wedge Q)$
   c. $\neg (P \vee Q) \vee \neg Q$
   d. $(P \vee Q) \vee P$
   e. $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$
   f. $(P \rightarrow Q) \rightarrow (Q \rightarrow P)$
   g. $P \vee (P \rightarrow Q)$
   h. $(P \wedge (Q \rightarrow P)) \rightarrow P$
   i. $P \vee (Q \rightarrow \neg P)$
   j. $(P \vee \neg Q) \wedge (\neg P \vee Q)$
   k. $\neg P \wedge (\neg (P \rightarrow Q))$
   l. $P \rightarrow \neg P$
   m. $\neg P \rightarrow P$
5. Chuyển các công thức sau ra dạng câu tuyển
   a. $(\neg P \wedge Q) \rightarrow R$
   b. $P \rightarrow ((Q \wedge R) \rightarrow S)$
   c. $\neg (P \vee \neg Q) \wedge (S \rightarrow T)$
   d. $(P \rightarrow Q) \rightarrow R$
   e. $\neg (P \wedge Q) \wedge (P \vee Q)$

6. Chuyển các công thức sau ra dạng câu hội
   a. $P \vee (\neg P \wedge Q \wedge R)$
   b. $\neg (P \rightarrow Q) \vee (P \vee Q)$
   c. $\neg(P \rightarrow Q)$
   d. $(\neg P \wedge Q) \vee (P \wedge \neg Q)$
7. Có công thức nào mà nó có thể ở dạng tuyển lẫn dạng hội? Nếu có, hãy cho ví dụ
8. Chứng minh các công thức sau là tương đương
   a. $(P \rightarrow Q) \wedge (P \rightarrow R) \equiv (P \rightarrow (Q \wedge R))$
   b. $(P \rightarrow Q) \rightarrow (P \wedge Q) \equiv (\neg P \rightarrow Q) \wedge (Q \rightarrow P)$
   c. $P \wedge Q \wedge (\neg P \vee \neg Q) \equiv \neg P \wedge \neg Q \wedge (P \vee Q)$
   d. $P \vee (P \rightarrow (P \wedge Q)) \equiv \neg P \vee \neg Q \vee (P \wedge Q)$
9. Chứng minh rằng $(\neg Q \rightarrow \neg P)$ là hệ quả lôgic của $(P \rightarrow Q)$
10. Xem xét các câu sau:

F_1: Sơn không thể là sinh viên giỏi trừ khi anh ta thông minh và cha Sơn hỗ trợ Sơn.

F_2: Nếu Sơn là sinh viên giỏi thì do cha Sơn hỗ trợ anh ta.

Chứng minh rằng F_2 là hệ quả lôgic của F_1

11. Xem xét câu phát biểu sau:

Nếu quốc hội từ chối ban hành luật mới thì cuộc đình công sẽ không kết thúc trừ khi nó kéo dài hơn một tuần và tổng thống từ chức. Giả sử quốc hội từ chối ban hành luật mới, cuộc đình công kết thúc và tổng thống không từ chức. Chứng minh điều mâu thuẩn sẻ xãy ra.

12. Chứng minh rằng F_5 là hệ quả lôgic của F_1, F_2 , F_3và F_4

F_1 : Nếu tổng thống không có đủ quyền và ông ta vô trách nhiệm thì trật tự không được vãn hồi hay cuộc bạo loạn sẽ lan rộng trừ khi những kẻ bạo loạn cảm thấy mệt mỏi và các nhà chức trách địa phương có những hành động hoà giải.
F_2: Trật tự được vãn hồi.
F_3: Những kẻ bạo loạn cảm thấy mệt mỏi
F_4: Tổng thống vô trách nhiệm và cuộc bạo loạn không lan rộng
F_5: Nếu tổng thống không có đủ quyền thì các nhà chức trách địa phương có những hành động hoà giải.

13. Chứng minh tập hợp S= $\{P \vee Q, \neg Q \vee R, \neg P \vee Q, \neg R\}$ sẽ cho câu rỗng [] bởi luật giải.

Chương II

LOGIC VỊ TỪ CẤP MỘT

Logic mệnh đề cho phép ta biểu diễn các sự kiện, mỗi kí hiệu trong logic mệnh đề được minh họa như là một sự kiện trong thế giới hiện thực, sử dụng các kết nối logic ta có thể tạo ra các câu phức hợp biểu diễn các sự kiện mang ý nghĩa phức tạp hơn. Như vậy khả năng biểu diễn của logic mệnh đề chỉ giới hạn trong phạm vi thế giới các sự kiện.

Thế giới hiện thực bao gồm các đối tượng, mỗi đối tượng có những tính chất riêng để phân biệt nó với các đối tượng khác. Các đối tượng lại có quan hệ với nhau. Các mối quan hệ rất đa dạng và phong phú. Chúng ta có thể liệt kê ra rất nhiều ví dụ về đối tượng, tính chất, quan hệ.

- Đối tượng : một cái bàn, một cái nhà, một cái cây, một con người, một con số. ...
- Tính chất : Cái bàn có thể có tính chất : có bốn chân, làm bằng gỗ, không có ngăn kéo. Con số có thể có tính chất là số nguyên, số hữu tỉ, là số chính phương. ..
- Quan hệ : cha con, anh em, bè bạn (giữa con người); lớn hơn nhỏ hơn, bằng nhau (giữa các con số) ; bên trong, bên ngoài nằm trên nằm dưới (giữa các đồ vật)...
- Hàm : Một trường hợp riêng của quan hệ là quan hệ hàm. Chẳng hạn, vì mỗi người có một mẹ, do đó ta có quan hệ hàm ứng mỗi người với mẹ của nó.

Logic vị từ cấp một là mở rộng của logic mệnh đề. Nó cho phép ta mô tả thế giới với các đối tượng, các thuộc tính của đối tượng và các mối quan hệ giữa các đối tượng. Nó sử dụng các biến (biến đối tượng) để chỉ một đối tượng trong một miền đối tượng nào đó. Để mô tả các thuộc tính của đối tượng, các quan hệ giữa các đối tượng, trong logic vị từ, người ta dựa vào các vị từ (predicate). Ngoài các kết nối logic như trong logic mệnh đề, logic vị từ cấp một còn sử dụng các lượng tử. Chẳng hạn, lượng tử $\forall$ (với mọi) cho phép ta tạo ra các câu nói tới mọi đối tượng trong một miền đối tượng nào đó.

Chương này dành cho nghiên cứu logic vị từ cấp một với tư cách là một ngôn ngữ biểu diễn tri thức. Logic vị từ cấp một đóng vai trò cực kì quan trọng trong biểu diễn tri thức, vì khả năng biểu diễn của nó (nó cho phép ta biểu diễn tri thức về thế giới với các đối tượng, các thuộc tính của đối tượng và các quan hệ của đối tượng), và hơn nữa, nó là cơ sở cho nhiều ngôn ngữ logic khác.

I. Cú pháp

Các ký hiệu

• Các ký hiệu hằng: a, b, c, An, Ba, John,...

• Các ký hiệu biến: x, y, z, u, v, w,...

• Các ký hiệu vị từ: P, Q, R, S, Like, Havecolor, Prime,...

Mỗi vị từ là vị từ của n biến ($n \geq 0$). Chẳng hạn Like là vị từ của hai biến, Prime là vị từ một biến. Các ký hiệu vị từ không biến là các ký hiệu mệnh đề.

• Các ký hiệu hàm: f, g, cos, sin, mother, husband, distance,...

Mỗi hàm là hàm của n biến ($n \geq 1$). Chẳng hạn, cos, sin là hàm một biến, distance là hàm của hai biến.

• Các ký hiệu kết nối logic: $\wedge$ (hội), $\vee$ (tuyển), $\neg$ (phủ định), $\Rightarrow$ (kéo theo), $\Leftrightarrow$ (kéo theo nhau).

• Các ký hiệu lượng tử: $\forall$ (với mọi), $\exists$ (tồn tại).

• Các ký hiệu ngăn cách: dấu phẩy, dấu mở ngoặc và dấu đóng ngoặc.

Các hạng thức

Các hạng thức (term) là các biểu thức mô tả các đối tượng. Các hạng thức được xác định đệ quy như sau:

• Các ký hiệu hằng và các ký hiệu biến là hạng thức.

• Nếu $t_1, t_2, t_3, ..., t_n$ là n hạng thức và f là một ký hiệu hàm n biến thì $f(t_1, t_2, t_3, ..., t_n)$ là hạng thức. Một hạng thức không chứa biến được gọi là một hạng thức cụ thể (ground term).

Chẳng hạn, A_n là ký hiệu hằng, mother là ký hiệu hàm một biến, thì mother (A_n) là một hạng thức cụ thể.

Các công thức phân tử

Chúng ta sẽ biểu diễn các tính chất của đối tượng, hoặc các quan hệ của đối tượng bởi các công thức phân tử (câu đơn).

Các công thức phân tử được xác định đệ quy như sau.

• Các ký hiệu vị từ không biến (các ký hiệu mệnh đề) là câu đơn.

• Nếu $t_1, t_2, t_3, ..., t_n$ là n hạng thức và p là vị từ của n biến thì $p(t_1, t_2, t_3, ..., t_n)$ là câu đơn.

Chẳng hạn, Hoa là một ký hiệu hằng, Love là một vị từ của hai biến, husband là hàm của một biến, thì Love (Hoa, husband(Hoa)) là một câu đơn.

Các công thức

Từ công thức phần tử, sử dụng các kết nối logic và các lượng tử, ta xây dựng nên các công thức (các câu).

Các công thức được xác định đệ quy như sau:

• Các công thức phân tử là công thức.

• Nếu G và H là các công thức, thì các biểu thức $(G \wedge H)$, $(G \vee H)$, $(\neg G)$, $(G \Rightarrow H)$, $(G \Leftrightarrow H)$ là công thức.

• Nếu G là một công thức và x là biến thì các biểu thức $(\forall x\ G)$, $(\exists x\ G)$ là công thức.

Các công thức không phải là công thức phân tử sẽ được gọi là các câu phức hợp. Các công thức không chứa biến sẽ được gọi là công thức cụ thể. Khi viết các công thức ta sẽ bỏ đi các dấu ngoặc không cần thiết, chẳng hạn các dấu ngoặc ngoài cùng.

• Lượng tử phổ dụng ($\forall$) cho phép mô tả tính chất của cả một lớp các đối tượng, chứ không phải của một đối tượng, mà không cần phải liệt kê ra tất cả các đối tượng trong lớp. Chẳng hạn sử dụng vị từ Elephant(x) (đối tượng x là con voi) và vị từ Color(x, Gray) (đối tượng x có màu xám) thì câu " tất cả các con voi đều có màu xám" có thể biểu diễn bởi công thức $\forall$ x (Elephant(x) $\Rightarrow$ Color(x, Gray)).

• Lượng tử tồn tại ($\exists$) cho phép ta tạo ra các câu nói đến một đối tượng nào đó trong một lớp đối tượng mà nó có một tính chất hoặc thoả mãn một quan hệ nào đó. Chẳng hạn bằng cách sử dụng các câu đơn Student(x) (x là sinh viên) và Inside(x, P301), (x ở trong phòng 301), ta có thể biểu diễn câu " Có một sinh viên ở phòng 301" bởi biểu thức $\exists$ x (Student(x) $\wedge$ Inside(x, P301)).

Một công thức là công thức phân tử hoặc phủ định của công thức phân tử được gọi là literal. Chẳng hạn, Play(x, Football), $\neg$Like(Lan, Rose) là các literal. Một công thức là tuyển của các literal sẽ được gọi là câu tuyển. Chẳng hạn, Male(x) $\vee$ $\neg$Like(x, Foodball) là câu tuyển.

Trong công thức $\forall$ x G, hoặc $\exists$ x G trong đó G là một công thức nào đó, thì mỗi xuất hiện của biến x trong công thức G được gọi là xuất hiện buộc. Một công thức mà tất cả các biến đều là xuất hiện buộc thì được gọi là công thức đóng.

Ví dụ: Công thức $\forall$ xP(x, f(a, x)) $\wedge$ $\exists$ y Q(y) là công thức đóng, còn công thức $\forall$ x P(x, f(y, x)) không phải là công thức đóng, vì sự xuất hiện của biến y trong công thức này không chịu ràng buộc bởi một lượng tử nào cả (sự xuất hiện của y gọi là sự xuất hiện tự do).

Sau này chúng ta chỉ quan tâm tới các công thức đóng.

II. Ngữ nghĩa

Cũng như trong logic mệnh đề, nói đến ngữ nghĩa là chúng ta nói đến ý nghĩa của các công thức trong một thế giới hiện thực nào đó mà chúng ta sẽ gọi là một minh họa.

Để xác định một minh hoạ, trước hết ta cần xác định một miền đối tượng (nó bao gồm tất cả các đối tượng trong thế giới hiện thực mà ta quan tâm).

Trong một minh hoạ, các ký hiệu hằng sẽ được gắn với các đối tượng cụ thể trong miền đối tượng các ký hiệu hàm sẽ được gắn với một hàm cụ thể nào đó. Khi đó, mỗi hạng thức cụ thể sẽ chỉ định một đối tượng cụ thể trong miền đối tượng. Chẳng hạn, nếu An là một ký hiệu hằng, Father là một ký hiệu hàm, nếu trong minh hoạ An ứng với một người cụ thể nào đó, còn Father(x) gắn với hàm; ứng với mỗi x là cha của nó, thì hạng thức Father(An) sẽ chỉ người cha của An .

- Ngữ nghĩa của các câu đơn

Trong một minh hoạ, các ký hiệu vị từ sẽ được gắn với một thuộc tính, hoặc một quan hệ cụ thể nào đó. Khi đó mỗi công thức phân tử (không chứa biến) sẽ chỉ định một sự kiện cụ thể. Đương nhiên sự kiện này có thể là đúng (True) hoặc sai (False). Chẳng hạn, nếu trong minh hoạ, ký hiệu hằng Lan ứng với một cô gái cụ thể nào đó, còn Student(x) ứng với thuộc tính "x là sinh viên" thì câu Student (Lan) có giá trị chân trị là True hoặc False tuỳ thuộc trong thực tế Lan có phải là sinh viên hay không.

- Ngữ nghĩa của các câu phức hợp

Khi đã xác định được ngữ nghĩa của các câu đơn, ta có thể thực hiện được ngữ nghĩa của các câu phức hợp (được tạo thành từ các câu đơn bằng cách liên kết các câu đơn bởi các kết nối logic) như trong logic mệnh đề.

Ví dụ: Câu Student(Lan) $\wedge$ Student(An) nhận giá trị True nếu cả hai câu Student(Lan) và Student(An) đều có giá trị True, tức là cả Lan và An đều là sinh viên.

Câu Like(Lan, Rose) $\vee$ Like(An, Tulip) là đúng nếu câu Like(Lan, Rose) là đúng hay câu Like(An, Tulip) là đúng.

- Ngữ nghĩa của các câu chứa các lượng tử

Ngữ nghĩa của các câu $\forall$ x G, trong đó G là một công thức nào đó, được xác định như là ngữ nghĩa của công thức là hội của tất cả các công thức nhận được từ công thức G bằng cách thay x bởi một đối tượng trong miền đối tượng. Chẳng hạn, nếu miền đối tượng gồm ba người {Lan, An, Hoa} thì ngữ nghĩa của câu $\forall$ x Student(x) được xác định là ngữ nghĩa của câu Student(Lan) $\wedge$ Student(An) $\wedge$ Student(Hoa). Câu này đúng khi và chỉ khi cả ba câu thành phần đều đúng, tức là cả Lan, An, Hoa đều là sinh viên.

Như vậy, công thức $\forall$ x G là đúng nếu và chỉ nếu mọi công thức nhận được từ G bằng cách thay x bởi mọi đối tượng trong miền đối tượng đều đúng, tức là G đúng cho tất cả các đối tượng x trong miền đối tượng.

Ngữ nghĩa của công thức $\exists$ x G được xác định như là ngữ nghĩa của công thức là tuyển của tất cả các công thức nhận được từ G bằng cách thay x bởi một đối tượng trong miền đối tượng. Chẳng hạn, nếu ngữ nghĩa của câu Younger(x, 20) là " x trẻ hơn 30 tuổi " và miền đối tượng gồm ba người {Lan, An, Hoa} thì ngữ nghĩa của câu $\exists$ x Yourger(x, 20) là ngữ nghĩa của câu Yourger(Lan, 20) $\vee$ Yourger(An, 20) $\vee$ Yourger(Hoa, 20). Câu này nhận giá trị True nếu và chỉ nếu ít nhất một trong ba người Lan, An, Hoa trẻ hơn 20.

Như vậy công thức $\exists$ x G là đúng nếu và chỉ nếu một trong các công thức nhận được từ G bằng cách thay x bằng một đối tượng trong miền đối tượng là đúng.

Bằng các phương pháp đã trình bày ở trên, ta có thể xác định được giá trị chân trị (True,False) của một công thức bất kỳ trong một minh hoạ. (lưu ý rằng, ta chỉ quan tâm tới các công thức đúng).

Sau khi đã xác định khái niệm minh hoạ và giá trị chân trị của một công thức trong một minh hoạ, có thể đưa ra các khái niệm công thức vững chắc (thoả được, không thoả được), mô hình của công thức giống như trong logic mệnh đề.

- Các công thức tương đương

Cũng như trong logic mệnh đề, ta nói hai công thức G và H tương đương (viết là G $\equiv$ H) nếu chúng cùng đúng hoặc cùng sai trong một minh hoạ. Ngoài các tương đương đã biết trong logic mệnh đề, trong logic vị từ cấp một còn có các tương đương khác liên quan tới các lượng tử. Giả sử G là một công thức, cách viết G(x) nói rằng công thức G có chứa các xuất hiện của biến x. Khi đó công thức G(y) là công thức nhận được từ G(x) bằng cách thay tất cả các xuất hiện của x bởi y. Ta nói G(y) là công thức nhận được từ G(x) bằng cách đặt tên lại (biến x được đổi tên lại là y).

Định nghĩa: một công thức F trong logic vị từ cấp một là dạng **prenex chuẩn** nếu và chỉ nếu F ở dạng thức sau: $(Q_1\ x_1)\ldots(Q_n\ x_n)(M)$

Trong đó $(Q_i\ x_i)$, $\forall$ i=1..n thì hoặc ($\forall\ x_i$) hoặc ($\exists\ x_i$)và M không chứa lượng tử, $(Q_1\ x_1)\ldots(Q_n\ x_n)$ gọi là tiền tố và M là ma trận của công thức F

Chúng ta có các tương đương sau đây:

1. $\forall\ x\ G(x) \equiv \forall\ y\ G(y)$

$\exists\ x\ G(x) \equiv \exists\ y\ G(y)$

Đặt tên lại biến đi sau lượng tử phổ dụng (tồn tại), ta nhận được công thức tương đương.

Ví dụ : $\forall\ x$ Love(x, Husband(x)) $\equiv \forall\ y$ Love(y, Husband(y))

2. $\neg(\forall\ x\ G(x)) \equiv \exists\ x\ (\neg\ G(x))$

$\neg(\exists\ x\ G(x)) \equiv \forall\ x\ (\neg\ G(x))$

3. $\forall\ x\ (G(x) \wedge H(x)) \equiv \forall\ x\ G(x) \wedge \forall\ x\ H(x)$

$\exists\ x\ (G(x) \vee H(x)) \equiv \exists\ x\ G(x) \vee \exists\ x\ H(x)$

4. $(Qx)F(x) \vee G \equiv (Qx)(F(x) \vee G)$

5. $(Qx)F(x) \wedge G \equiv (Qx)(F(x) \wedge G)$

III. Bài tập

1. Đặt P(x): x là số hửu tỷ, Q(x): x là số thực. Mô tả các câu sau dưới dạng logic vị từ cấp một
    a. Mỗi số hữu tỷ là số thực
    b. Vài số thực là số hửu tỷ
    c. Không phải tất cả số thực là số hửu tỷ
2. Đặt C(x): x là người bán xe hơi cũ và H(x): x thành thật. Hãy phát biểu các công thức sau đây:
    a. $\exists\ x\ C(x)$
    b. $\exists\ x\ H(x)$
    c. $\forall\ x\ (C(x) \rightarrow \neg H(x))$
    d. $\exists\ x\ (C(x) \wedge H(x))$
    e. $\exists\ x\ (H(x) \rightarrow C(x))$
3. Đặt P(x): x là một điểm, L(x): x là một đường, R(x,y,z): z đi qua x và y, E(x,y): x=y. Mô tả các câu sau dưới dạng logic vị từ cấp một:

Với mỗi hai điểm khác nhau, có một và chỉ một đường qua hai điểm

4. Một nhóm Abelian là một tập hợp A với một toán tử nhị phân +. Đặt P(x,y,z) và E(x,y) biểu diễn x+y=z và x=y tương ứng. Mô tả các tiên đề cho nhóm Abelian dưới dạng logic vị từ cấp một
    a. Với mỗi x, y trong A, Có tồn tại một z trong trong A sao cho x+y=z (đóng)
    b. Nếu x+y=z và x+y=w thì z=w (duy nhất)
    c. (x+y)+z = x+(y+z) (liên kết)
    d. x+y = y+x (đối xứng)

e. Cho mỗi x và y trong A, tồn tại z sao cho x+z=y (giải pháp đúng)

5. Cho minh hoạ sau (D={a,b})

P(a,a)	P(a,b)	P(b,a)	P(b,b)
T	F	F	T

xác định giá trị đúng của các công thức sau:

$\forall x \exists y P(x,y)$

$\forall x \forall y P(x,y)$

$\exists x \forall y P(x,y)$

$\exists y \neg P(a,y)$

$\forall x \forall y (P(x,y) \rightarrow P(y,x))$

$\forall x P(x,x)$

6. Xét công thức sau:

   A: $\exists x P(x) \rightarrow \forall x P(x)$

   a. Chứng minh rằng công thức này luôn luôn đúng nếu miền giá trị D chỉ có một phần tử
   b. Đặt D={a,b}. Tìm một minh hoạ trên D mà A là F

7. Xem minh hoạ sau:
   Miền giá trị D={1,2}
   Phép gán hằng a và b

a	b
1	2

   Phép gán hàm f

f(1)	f(2)
2	1

   Phép gán cho vị từ P

P(1,1)	P(1,2)	P(2,1)	P(2,2)
T	T	F	F

   Lượng giá giá trị đúng của các công thức sau theo các phép gán trên
   $P(a,f(a)) \wedge P(b, f(b))$
   $\forall x \exists y P(y,x)$
   $\forall x \forall y (P(x,y) \rightarrow P(f(x),f(y)))$

8. Đặt

F_1: $\forall x\,(P(x) \rightarrow Q(x))$

F_2: $\neg Q(a)$

Chứng minh rằng $\neg P(a)$ là hệ quả logic của F_1 và F_2

9. Chuyển các công thức sau thành dạng prenex chuẩn:
   a. $\forall x\,(P(x) \rightarrow \exists y\,Q(x, y))$
   b. $\exists x \neg(\exists y\,P(x,y) \rightarrow (\exists z\,Q(z) \rightarrow R(x)))$
   c. $\forall x\ \forall y\,(\exists z\ P(x,y,z) \wedge (\exists u\,Q(x, u) \rightarrow \exists v\,Q(y, v)))$
10. Xem xét các câu sau:

F_1: Mỗi sinh viên thì thành thật

F_2: An không thành thật

Chứng minh An không là một sinh viên

Xem xét các tiền đề sau

(1) Tất cả vận động viên thì khoẻ

(2) Người nào vừa khoẻ và thông minh thì sẽ thành công trong sự nghiệp

(3) An là vận động viên

(4) An thông minh

Hãy kết luận rằng An sẽ thành công trong sự nghiệp

12. Giả sử rằng Sơn được yêu bởi bất cứ người nào mà có yêu ai đó. Giả sử thêm rằng không ai không yêu người nào. Chứng tỏ rằng Sơn được yêu bởi ai đó.

Phần II

GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Vấn đề tìm kiếm, một cách tổng quát, có thể hiểu là tìm một đối tượng thỏa mãn một số đòi hỏi nào đó, trong một tập hợp rộng lớn các đối tượng. Chúng ta có thể kể ra rất nhiều vấn đề mà việc giải quyết nó được quy về vấn đề tìm kiếm.

Các trò chơi, chẳng hạn cờ vua, cờ carô có thể xem như vấn đề tìm kiếm. Trong số rất nhiều nước đi được phép thực hiện, ta phải tìm ra các nước đi dẫn tới tình thế kết cuộc mà ta là người thắng.

Chứng minh định lý cũng có thể xem như vấn đề tìm kiếm. Cho một tập các tiên đề và các luật suy diễn, trong trường hợp này mục tiêu của ta là tìm ra một chứng minh (một dãy các luật suy diễn được áp dụng) để được đưa đến công thức mà ta cần chứng minh.

Trong các lĩnh vực nghiên cứu của Trí Tuệ Nhân Tạo, chúng ta thường xuyên phải đối đầu với vấn đề tìm kiếm. Đặc biệt trong lập kế hoạch và học máy, tìm kiếm đóng vai trò quan trọng.

Trong phần này chúng ta sẽ nghiên cứu các kỹ thuật tìm kiếm cơ bản được áp dụng để giải quyết các vấn đề và được áp dụng rộng rãi trong các lĩnh vực nghiên cứu khác của Trí Tuệ Nhân Tạo. Chúng ta lần lượt nghiên cứu các kỹ thuật sau:

- Các kỹ thuật tìm kiếm mù, trong đó chúng ta không có hiểu biết gì về các đối tượng để hướng dẫn tìm kiếm mà chỉ đơn thuần là xem xét theo một hệ thống nào đó tất cả các đối tượng để phát hiện ra đối tượng cần tìm.
- Các kỹ thuật tìm kiếm kinh nghiệm (tìm kiếm heuristic) trong đó chúng ta dựa vào kinh nghiệm và sự hiểu biết của chúng ta về vấn đề cần giải quyết để xây dựng nên hàm đánh giá hướng dẫn sự tìm kiếm.
- Các kỹ thuật tìm kiếm tối ưu.
- Các phương pháp tìm kiếm có đối thủ, tức là các chiến lược tìm kiếm nước đi trong các trò chơi hai người, chẳng hạn cờ vua, cờ tướng, cờ carô.

Chương III

CÁC CHIẾN LƯỢC TÌM KIẾM MÙ

Trong chương này, chúng ta sẽ nghiên cứu các chiến lược tìm kiếm mù (blind search): tìm kiếm theo bề rộng (breadth-first search) và tìm kiếm theo độ sâu (depth-first search). Hiệu quả của các phương pháp tìm kiếm này cũng sẽ được đánh giá.

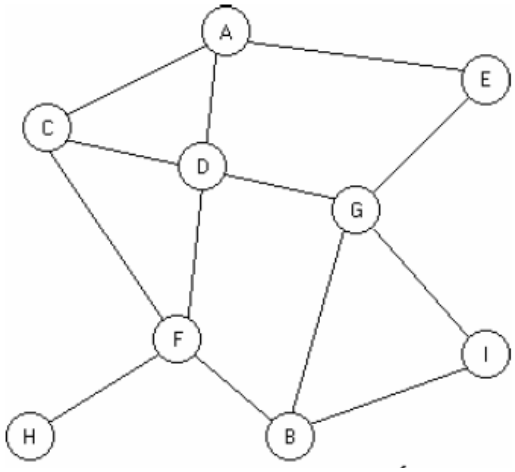
I. Biểu diễn vấn đề trong không gian trạng thái

Một khi chúng ta muốn giải quyết một vấn đề nào đó bằng tìm kiếm, đầu tiên ta phải xác định không gian tìm kiếm. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm tìm kiếm.

Nó có thể là không gian liên tục, chẳng hạn không gian các véctơ thực n chiều; nó cũng có thể là không gian các đối tượng rời rạc.

Trong mục này ta sẽ xét việc biểu diễn một vấn đề trong không gian trạng thái sao cho việc giải quyết vấn đề được quy về việc tìm kiếm trong không gian trạng thái.

Một phạm vi rộng lớn các vấn đề, đặc biệt các câu đố, các trò chơi, có thể mô tả bằng cách sử dụng khái niệm trạng thái và toán tử (phép biến đổi trạng thái). Chẳng hạn, một khách du lịch có trong tay bản đồ mạng lưới giao thông nối các thành phố trong một vùng lãnh thổ (hình 3.1), du khách đang ở thành phố A và anh ta muốn tìm đường đi tới thăm thành phố B. Trong bài toán này, các thành phố có trong các bản đồ là các trạng thái, thành phố A là trạng thái ban đầu, B là trạng thái kết thúc. Khi đang ở một thành phố, chẳng hạn ở thành phố D anh ta có thể đi theo các con đường để nối tới các thành phố C, F và G. Các con đường nối các thành phố sẽ được biểu diễn bởi các toán tử. Một toán tử biến đổi một trạng thái thành một trạng thái khác. Chẳng hạn, ở trạng thái D sẽ có ba toán tử dẫn trạng thái D tới các trạng thái C, F và G.



Hình 3.1 Tìm đường đi từ A tới B

Vấn đề của du khách bây giờ sẽ là tìm một dãy toán tử để đưa trạng thái ban đầu A tới trạng thái kết thúc B.

Một ví dụ khác, trong trò chơi cờ vua, mỗi cách bố trí các quân trên bàn cờ là một trạng thái. Trạng thái ban đầu là sự sắp xếp các quân lúc bắt đầu cuộc chơi. Mỗi nước đi hợp lệ là một toán tử, nó biến đổi một cảnh huống trên bàn cờ thành một cảnh huống khác.

Như vậy muốn biểu diễn một vấn đề trong không gian trạng thái, ta cần xác định các yếu tố sau:

- Trạng thái ban đầu.
- Một tập hợp các toán tử. Trong đó mỗi toán tử mô tả một hành động hoặc một phép biến đổi có thể đưa một trạng thái tới một trạng thái khác.

Tập hợp tất cả các trạng thái có thể đạt tới từ trạng thái ban đầu bằng cách áp dụng một dãy toán tử, lập thành không gian trạng thái của vấn đề.

Ta sẽ ký hiệu không gian trạng thái là U, trạng thái ban đầu là u_0 ($u_0 \in U$). Mỗi toán tử R có thể xem như một ánh xạ R: U → U. Nói chung R là một ánh xạ không xác định khắp nơi trên U.

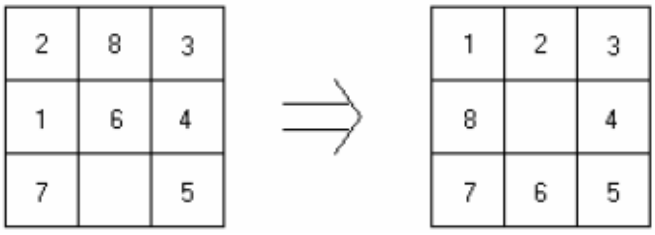
- Một tập hợp T các trạng thái kết thúc (trạng thái đích). T là tập con của không gian U. Trong vấn đề của du khách trên, chỉ có một trạng thái đích, đó là thành phố B. Nhưng trong nhiều vấn đề (chẳng hạn các loại cờ) có thể có nhiều trạng thái đích và ta không thể xác định trước được các trạng thái đích. Nói chung trong phần lớn các vấn đề ta chỉ có thể mô tả các trạng thái đích là các trạng thái thỏa mãn một số điều kiện nào đó.

Khi chúng ta biểu diễn một vấn đề thông qua các trạng thái và các toán tử, thì việc tìm nghiệm của bài toán được quy về việc tìm đường đi từ trạng thái ban đầu tới trạng thái đích. (Một đường đi trong không gian trạng thái là một dãy toán tử dẫn một trạng thái tới một trạng thái khác).

Chúng ta có thể biểu diễn không gian trạng thái bằng đồ thị định hướng, trong đó mỗi đỉnh của đồ thị tương ứng với một trạng thái. Nếu có toán tử R biến đổi trạng thái u thành trạng thái v, thì có cung gán nhãn R đi từ đỉnh u tới đỉnh v. Khi đó một đường đi trong không gian trạng thái sẽ là một đường đi trong đồ thị này.

Sau đây chúng ta sẽ xét một số ví dụ về các không gian trạng thái được xây dựng cho một số vấn đề.

Ví dụ 1: Bài toán 8 số. Chúng ta có bảng 3x3 ô và tám quân mang số hiệu từ 1 đến 8 được xếp vào tám ô, còn lại một ô trống, chẳng hạn như trong hình 2 bên trái. Trong trò chơi này, bạn có thể chuyển dịch các quân ở cạch ô trống tới ô trống đó. Vấn đề của bạn là tìm ra một dãy các chuyển dịch để biến đổi cảnh huống ban đầu (hình 3.2 bên trái) thành một cảnh huống xác định nào đó, chẳng hạn cảnh huống trong hình 3.2 bên phải.



Hình 3.2 Trạng thái ban đầu và trạng thái kết thúc của bài toán số.

Trong bài toán này, trạng thái ban đầu là cảnh huống ở bên trái hình 3.2, còn trạng thái kết thúc ở bên phải hình 3.2. Tương ứng với các quy tắc chuyển dịch các quân, ta có bốn toán tử: up (đẩy quân lên trên), down (đẩy quân xuống dưới), left (đẩy quân sang trái), right (đẩy quân sang phải). Rõ ràng là, các toán tử này chỉ là các toán tử bộ phận; chẳng hạn, từ trạng thái ban đầu (hình 3.2), ta chỉ có thể áp dụng các toán tử down, left, right.

Trong các ví dụ trên việc tìm ra một biểu diễn thích hợp để mô tả các trạng thái của vấn đề là khá dễ dàng và tự nhiên. Song trong nhiều vấn đề việc tìm hiểu được biểu diễn thích hợp cho các trạng thái của vấn đề là hoàn toàn không đơn giản. Việc tìm ra dạng biểu diễn tốt cho các trạng thái đóng vai trò hết sức quan trọng trong quá trình giải quyết một vấn đề. Có thể nói rằng, nếu ta tìm được dạng biểu diễn tốt cho các trạng thái của vấn đề, thì vấn đề hầu như đã được giải quyết.

Ví dụ 2: Vấn đề triệu phú và kẻ cướp. Có ba nhà triệu phú và ba tên cướp ở bên bờ tả ngạn một con sông, cùng một chiếc thuyền chở được một hoặc hai người. Hãy tìm cách đưa mọi người qua sông sao cho không để lại ở bên bờ sông kẻ cướp nhiều hơn triệu phú. Đương nhiên trong bài toán này, các toán tử tương ứng với các hành động chở 1 hoặc 2 người qua sông. Nhưng ở đây ta cần lưu ý rằng, khi hành động xẩy ra (lúc thuyền đang bơi qua sông) thì ở bên bờ sông thuyền vừa dời chỗ, số kẻ cướp không được nhiều hơn số triệu phú. Tiếp theo ta cần quyết định cái gì là trạng thái của vấn đề. ở đây ta không cần phân biệt các nhà triệu phú và các tên cướp, mà chỉ số lượng của họ ở bên bờ sông là quan trọng. Để biểu diễn các trạng thái, ta sử dụng bộ ba (a, b, k), trong đó a là số triệu phú, b là số kẻ cướp ở bên bờ tả ngạn vào các thời điểm mà thuyền ở bờ này hoặc bờ kia, k = 1 nếu thuyền ở bờ tả ngạn và k = 0 nếu thuyền ở bờ hữu ngạn. Như vậy, không gian trạng thái cho bài toán triệu phú và kẻ cướp được xác định như sau:

Trạng thái ban đầu là (3, 3, 1).

Các toán tử. Có năm toán tử tương ứng với hành động thuyền chở qua sông 1 triệu phú, hoặc 1 kẻ cướp, hoặc 2 triệu phú, hoặc 2 kẻ cướp, hoặc 1 triệu phú và 1 kẻ cướp.

Trạng thái kết thúc là (3, 3, 0).

II. Các chiến lược tìm kiếm

Như ta đã thấy trong mục 1.1, để giải quyết một vấn đề bằng tìm kiếm trong không gian trạng thái, đầu tiên ta cần tìm dạng thích hợp mô tả các trạng thái của vấn đề. Sau đó cần xác định:

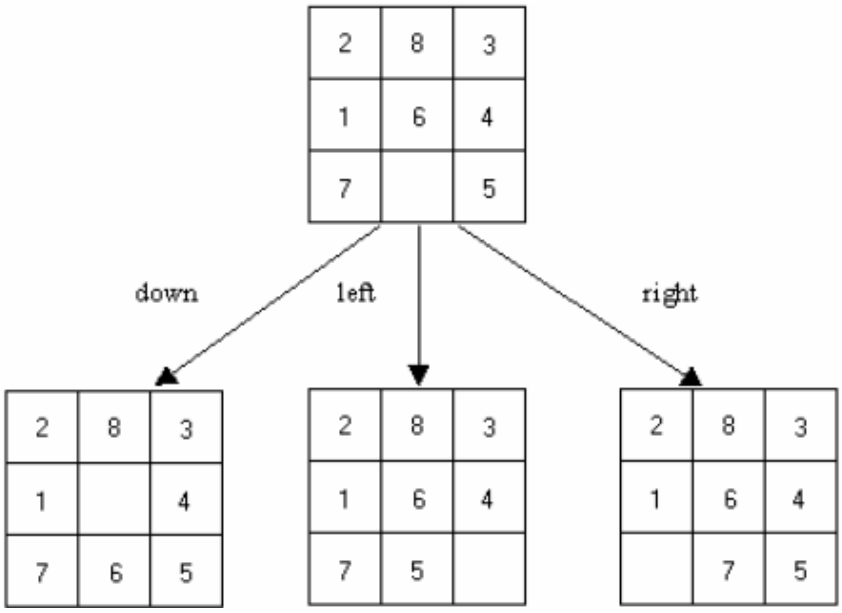
- Trạng thái ban đầu.
- Tập các toán tử.
- Tập T các trạng thái kết thúc. (T có thể không được xác định cụ thể gồm các trạng thái nào mà chỉ được chỉ định bởi một số điều kiện nào đó).

Giả sử u là một trạng thái nào đó và R là một toán tử biến đổi u thành v. Ta sẽ gọi v là trạng thái kề u, hoặc v được sinh ra từ trạng thái u bởi toán tử R. Quá trình áp dụng các toán tử để sinh ra các trạng thái kề u được gọi là phát triển trạng thái u. Chẳng hạn, trong bài toán toán số, phát triển trạng thái ban đầu (hình 2 bên trái), ta nhận được ba trạng thái kề (hình 3.3).

Khi chúng ta biểu diễn một vấn đề cần giải quyết thông qua các trạng thái và các toán tử thì việc tìm lời giải của vấn đề được quy về việc tìm đường đi từ trạng thái ban đầu tới một trạng thái kết thúc nào đó.

Có thể phân các chiến lược tìm kiếm thành hai loại:

Các chiến lược tìm kiếm mù. Trong các chiến lược tìm kiếm này, không có một sự hướng dẫn nào cho sự tìm kiếm, mà ta chỉ phát triển các trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có hai kỹ thuật tìm kiếm mù, đó là tìm kiếm theo bề rộng và tìm kiếm theo độ sâu.



Hình 3.3 Phát triển một trạng thái

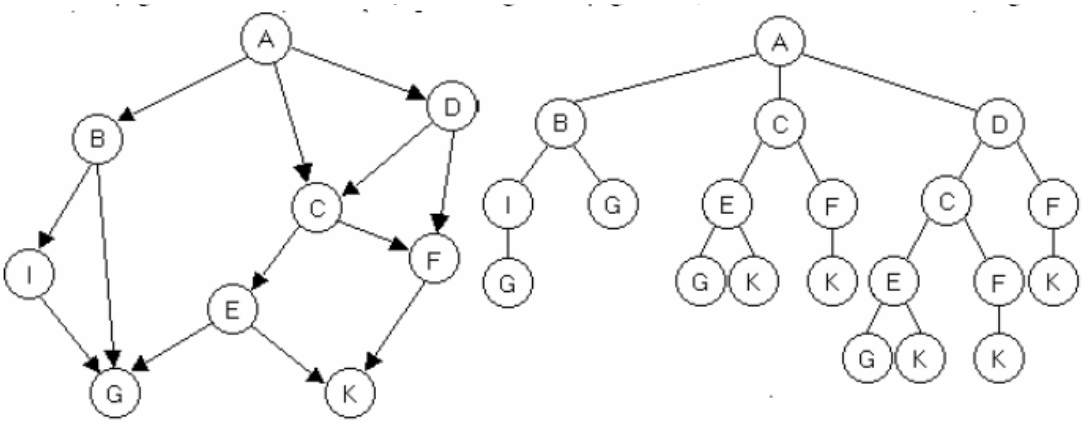
Tư tưởng của tìm kiếm theo bề rộng là các trạng thái được phát triển theo thứ tự mà chúng được sinh ra, tức là trạng thái nào được sinh ra trước sẽ được phát triển trước.

Trong nhiều vấn đề, dù chúng ta phát triển các trạng thái theo hệ thống nào (theo bề rộng hoặc theo độ sâu) thì số lượng các trạng thái được sinh ra trước khi ta gặp trạng thái đích thường là cực kỳ lớn. Do đó các thuật toán tìm kiếm mù kém hiệu quả, đòi hỏi rất nhiều không gian và thời gian. Trong thực tế, nhiều vấn đề không thể giải quyết được bằng tìm kiếm mù.

• Tìm kiếm kinh nghiệm (tìm kiếm heuristic). Trong rất nhiều vấn đề, chúng ta có thể dựa vào sự hiểu biết của chúng ta về vấn đề, dựa vào kinh nghiệm, trực giác, để đánh giá các trạng thái. Sử dụng sự đánh giá các trạng thái để hướng dẫn sự tìm kiếm: trong quá trình phát triển các trạng thái, ta sẽ chọn trong số các trạng thái chờ phát triển, trạng thái được đánh giá là tốt nhất để phát triển. Do đó tốc độ tìm kiếm sẽ nhanh hơn. Các phương pháp tìm kiếm dựa vào sự đánh giá các trạng thái để hướng dẫn sự tìm kiếm gọi chung là các phương pháp tìm kiếm kinh nghiệm.

Như vậy chiến lược tìm kiếm được xác định bởi chiến lược chọn trạng thái để phát triển ở mỗi bước. Trong tìm kiếm mù, ta chọn trạng thái để phát triển theo thứ tự mà chúng được sinh ra; còn trong tìm kiếm kinh nghiệm ta chọn trạng thái dựa vào sự đánh giá các trạng thái.

Chúng ta có thể nghĩ đến quá trình tìm kiếm như quá trình xây dựng cây tìm kiếm. Cây tìm kiếm là cây mà các đỉnh được gắn bởi các trạng thái của không gian trạng thái. Gốc của cây tìm kiếm tương ứng với trạng thái ban đầu. Nếu một đỉnh ứng với trạng thái u, thì các đỉnh con của nó ứng với các trạng thái v kề u. Hình 3.4a là đồ thị biểu diễn một không gian trạng thái với trạng thái ban đầu là A, hình 3.4b là cây tìm kiếm tương ứng.



Hình 3.4

Mỗi chiến lược tìm kiếm trong không gian trạng thái tương ứng với một phương pháp xây dựng cây tìm kiếm. Quá trình xây dựng cây bắt đầu từ cây chỉ có một đỉnh là trạng thái ban đầu. Giả sử tới một bước nào đó trong chiến lược tìm kiếm, ta đã xây dựng được một cây nào đó, các lá của cây tương ứng với các trạng thái chưa được phát triển. Bước tiếp theo phụ thuộc vào chiến lược tìm kiếm mà một đỉnh nào đó trong các lá được chọn để phát triển. Khi phát triển đỉnh đó, cây tìm kiếm được mở rộng bằng cách thêm vào các đỉnh con của đỉnh đó. Kỹ thuật tìm kiếm theo bề rộng (theo độ sâu) tương ứng với phương pháp xây dựng cây tìm kiếm theo bề rộng (theo độ sâu).

III. Các chiến lược tìm kiếm mù

Trong mục này chúng ta sẽ trình bày hai chiến lược tìm kiếm mù: tìm kiếm theo bề rộng và tìm kiếm theo độ sâu. Trong tìm kiếm theo bề rộng, tại mỗi bước ta sẽ chọn trạng thái để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác. Còn trong tìm kiếm theo độ sâu, trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển.

Chúng ta sử dụng danh sách L để lưu các trạng thái đã được sinh ra và chờ được phát triển. Mục tiêu của tìm kiếm trong không gian trạng thái là tìm đường đi từ trạng thái ban đầu tới trạng thái đích, do đó ta cần lưu lại vết của đường đi. Ta có thể sử dụng hàm father để lưu lại cha của mỗi đỉnh trên đường đi, father(v) = u nếu cha của đỉnh v là u.

III.1 Tìm kiếm theo bề rộng

Thuật toán tìm kiếm theo bề rộng được mô tả bởi thủ tục sau:

procedure Breadth_First_Search;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** L rỗng **then**

{thông báo tìm kiếm thất bại; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo tìm kiếm thành công; **stop**};

2.4 **for** mỗi trạng thái v kề u **do** {

Đặt v vào cuối danh sách L;

father(v) ← u}

end;

Chúng ta có một số nhận xét sau đây về thuật toán tìm kiếm theo bề rộng:

• Trong tìm kiếm theo bề rộng, trạng thái nào được sinh ra trước sẽ được phát triển trước, do đó danh sách L được xử lý như hàng đợi. Trong bước 2.3, ta cần kiểm tra xem u có là trạng thái kết thúc hay không. Nói chung các trạng thái kết thúc được xác định bởi một số điều kiện nào đó, khi đó ta cần kiểm tra xem u có thỏa mãn các điều kiện đó hay không.

• Nếu bài toán có nghiệm (tồn tại đường đi từ trạng thái ban đầu tới trạng thái đích), thì thuật toán tìm kiếm theo bề rộng sẽ tìm ra nghiệm, đồng thời đường đi tìm được sẽ là ngắn nhất. Trong trường hợp bài toán vô nghiệm và không gian trạng thái hữu hạn, thuật toán sẽ dừng và cho thông báo vô nghiệm.

Đánh giá tìm kiếm theo bề rộng

Bây giờ ta đánh giá thời gian và bộ nhớ mà tìm kiếm theo bề rộng đòi hỏi. Giả sử rằng, mỗi trạng thái khi được phát triển sẽ sinh ra b trạng thái kề. Ta sẽ gọi b là nhân tố nhánh. Giả sử rằng, nghiệm của bài toán là đường đi có độ dài d. Bởi nhiều nghiệm có thể được tìm ra tại một đỉnh bất kỳ ở mức d của cây tìm kiếm, do đó số đỉnh cần xem xét để tìm ra nghiệm là:

$1 + b + b^2 + ... + b^{d-1} + k$

Trong đó k có thể là 1, 2, ..., b^d. Do đó số lớn nhất các đỉnh cần xem xét là:

$1 + b + b^2 + ... + b^d$

Như vậy, độ phức tạp thời gian của thuật toán tìm kiếm theo bề rộng là $O(b^d)$. Độ phức tạp không gian cũng là $O(b^d)$, bởi vì ta cần lưu vào danh sách L tất cả các đỉnh của cây tìm kiếm ở mức d, số các đỉnh này là b^d.

Để thấy rõ tìm kiếm theo bề rộng đòi hỏi thời gian và không gian lớn tới mức nào, ta xét trường hợp nhân tố nhánh b = 10 và độ sâu d thay đổi. Giả sử để phát hiện và kiểm tra 1000 trạng thái cần 1 giây, và lưu giữ 1 trạng thái cần 100 bytes. Khi đó thời gian và không gian mà thuật toán đòi hỏi được cho trong bảng sau:

Độ sâu d	Thời gian	Không gian
4	11 giây	1 MB
6	18 phút	111 MB
8	31 giờ	11 GB
10	128 ngày	1TB
12	35 năm	111 TB
14	3500 năm	11 PB

III.2 Tìm kiếm theo độ sâu

Như ta đã biết, tư tưởng của chiến lược tìm kiếm theo độ sâu là, tại mỗi bước trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển. Do đó thuật toán tìm kiếm theo độ sâu là hoàn toàn tương tự như thuật toán tìm kiếm theo bề rộng, chỉ có một điều khác là, ta xử lý danh sách L các trạng thái chờ phát triển không phải như hàng đợi mà như ngăn xếp. Cụ thể là trong bước 2.4 của thuật toán tìm kiếm theo bề rộng, ta cần sửa lại là "Đặt v vào đầu danh sách L".

Sau đây chúng ta sẽ đưa ra các nhận xét so sánh hai chiến lược tìm kiếm mù:

• Thuật toán tìm kiếm theo bề rộng luôn luôn tìm ra nghiệm nếu bài toán có nghiệm. Song không phải với bất kỳ bài toán có nghiệm nào thuật toán tìm kiếm theo độ sâu cũng tìm ra nghiệm! Nếu bài toán có nghiệm và không gian trạng thái hữu hạn, thì thuật toán tìm kiếm theo độ sâu sẽ tìm ra nghiệm. Tuy nhiên, trong trường hợp không gian trạng thái vô hạn, thì có thể nó không tìm ra nghiệm, lý do là ta luôn luôn đi xuống theo độ sâu, nếu ta đi theo một nhánh vô hạn mà nghiệm không nằm trên nhánh đó thì thuật toán sẽ không dừng. Do đó người ta khuyên rằng, không nên áp dụng tìm kiếm theo dộ sâu cho các bài toán có cây tìm kiếm chứa các nhánh vô hạn.

• Độ phức tạp của thuật toán tìm kiếm theo độ sâu.

Giả sử rằng, nghiệm của bài toán là đường đi có độ dài d, cây tìm kiếm có nhân tố nhánh là b và có chiều cao là d. Có thể xẩy ra, nghiệm là đỉnh ngoài cùng bên phải trên mức d của cây tìm kiếm, do đó độ phức tạp thời gian của tìm kiếm theo độ sâu trong trường hợp xấu nhất là $O(b^d)$, tức là cũng như tìm kiếm theo bề rộng. Tuy nhiên, trên thực tế đối với nhiều bài toán, tìm kiếm theo độ sâu thực sự nhanh hơn tìm kiếm theo bề rộng. Lý do là tìm kiếm theo bề rộng phải xem xét toàn bộ cây tìm kiếm tới mức d-1, rồi mới xem xét các đỉnh ở mức d. Còn trong tìm kiếm theo độ sâu, có thể ta chỉ cần xem xét một bộ phận nhỏ của cây tìm kiếm thì đã tìm ra nghiệm.

Để đánh giá độ phức tạp không gian của tìm kiếm theo độ sâu ta có nhận xét rằng, khi ta phát triển một đỉnh u trên cây tìm kiếm theo độ sâu, ta chỉ cần lưu các đỉnh chưa được phát triển mà chúng là các đỉnh con của các đỉnh nằm trên đường đi từ gốc tới đỉnh u. Như vậy đối với cây tìm kiếm có nhân tố nhánh b và độ sâu lớn nhất là d, ta chỉ

cần lưu ít hơn db đỉnh. Do đó độ phức tạp không gian của tìm kiếm theo độ sâu là O(db), trong khi đó tìm kiếm theo bề rộng đòi hỏi không gian nhớ $O(b^d)$!

III.3 Các trạng thái lặp

Như ta thấy trong mục 1.2, cây tìm kiếm có thể chứa nhiều đỉnh ứng với cùng một trạng thái, các trạng thái này được gọi là trạng thái lặp. Chẳng hạn, trong cây tìm kiếm hình 4b, các trạng thái C, E, F là các trạng thái lặp. Trong đồ thị biểu diễn không gian trạng thái, các trạng thái lặp ứng với các đỉnh có nhiều đường đi dẫn tới nó từ trạng thái ban đầu. Nếu đồ thị có chu trình thì cây tìm kiếm sẽ chứa các nhánh với một số đỉnh lập lại vô hạn lần. Trong các thuật toán tìm kiếm sẽ lãng phí rất nhiều thời gian để phát triển lại các trạng thái mà ta đã gặp và đã phát triển. Vì vậy trong quá trình tìm kiếm ta cần tránh phát sinh ra các trạng thái mà ta đã phát triển. Chúng ta có thể áp dụng một trong các giải pháp sau đây:

1. Khi phát triển đỉnh u, không sinh ra các đỉnh trùng với cha của u.

2. Khi phát triển đỉnh u, không sinh ra các đỉnh trùng với một đỉnh nào đó nằm trên đường đi dẫn tới u.

3. Không sinh ra các đỉnh mà nó đã được sinh ra, tức là chỉ sinh ra các đỉnh mới.

Hai giải pháp đầu dễ cài đặt và không tốn nhiều không gian nhớ, tuy nhiên các giải pháp này không tránh được hết các trạng thái lặp.

Để thực hiện giải pháp thứ 3 ta cần lưu các trạng thái đã phát triển vào tập Q, lưu các trạng thái chờ phát triển vào danh sách L. Đương nhiên, trạng thái v lần đầu được sinh ra nếu nó không có trong Q và L.

Việc lưu các trạng thái đã phát triển và kiểm tra xem một trạng thái có phải lần đầu được sinh ra không đòi hỏi rất nhiều không gian và thời gian. Chúng ta có thể cài đặt tập Q bởi bảng băm.

III.4 Tìm kiếm sâu lặp

Như chúng ta đã nhận xét, nếu cây tìm kiếm chứa nhánh vô hạn, khi sử dụng tìm kiếm theo độ sâu, ta có thể mắc kẹt ở nhánh đó và không tìm ra nghiệm. Để khắc phục hoàn cảnh đó, ta tìm kiếm theo độ sâu chỉ tới mức d nào đó; nếu không tìm ra nghiệm, ta tăng độ sâu lên d+1 và lại tìm kiếm theo độ sâu tới mức d+1. Quá trình trên được lặp lại với d lần lượt là 1, 2, ... đến một độ sâu max nào đó. Như vậy, thuật toán tìm kiếm sâu lặp (iterative deepening search) sẽ sử dụng thủ tục tìm kiếm sâu hạn chế (depth_limited search) như thủ tục con. Đó là thủ tục tìm kiếm theo độ sâu, nhưng chỉ đi tới độ sâu d nào đó rồi quay lên.

Trong thủ tục tìm kiếm sâu hạn chế, d là tham số độ sâu, hàm depth ghi lại độ sâu của mỗi đỉnh

procedure Depth_Limited_Search(d);

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu u_0;

$depth(u_0) \leftarrow 0$;

2. **loop do**

    2.1 **if** L rỗng **then**

{thông báo thất bại; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo thành công; **stop**};

2.4 **if** depth(u) ≤ d **then**

for mỗi trạng thái v kề u **do**

{Đặt v vào đầu danh sách L;

depth(v) ← depth(u) + 1};

end;

procedure Depth_Deepening_Search;

begin

for d ← 0 to max **do**

{Depth_Limited_Search(d);

if thành công **then** exit}

end;

Kỹ thuật tìm kiếm sâu lặp kết hợp được các ưu điểm của tìm kiếm theo bề rộng và tìm kiếm theo độ sâu. Chúng ta có một số nhận xét sau:

- Cũng như tìm kiếm theo bề rộng, tìm kiếm sâu lặp luôn luôn tìm ra nghiệm (nếu bài toán có nghiệm), miễn là ta chọn độ sâu đủ lớn.
- Tìm kiếm sâu lặp chỉ cần không gian nhớ như tìm kiếm theo độ sâu.
- Trong tìm kiếm sâu lặp, ta phải phát triển lặp lại nhiều lần cùng một trạng thái. Điều đó làm cho ta có cảm giác rằng, tìm kiếm sâu lặp lãng phí nhiều thời gian. Thực ra thời gian tiêu tốn cho phát triển lặp lại các trạng thái là không đáng kể so với thời gian tìm kiếm theo bề rộng. Thật vậy, mỗi lần gọi thủ tục tìm kiếm sâu hạn chế tới mức d, nếu cây tìm kiếm có nhân tố nhánh là b, thì số đỉnh cần phát triển là:

$1 + b + b^2 + ... + b^d$

Nếu nghiệm ở độ sâu d, thì trong tìm kiếm sâu lặp, ta phải gọi thủ tục tìm kiếm sâu hạn chế với độ sâu lần lượt là 0, 1, 2, ..., d. Do đó các đỉnh ở mức 1 phải phát triển lặp d lần, các đỉnh ở mức 2 lặp d-1 lần, ..., các đỉnh ở mức d lặp 1 lần. Như vậy tổng số đỉnh cần phát triển trong tìm kiếm sâu lặp là:

$(d+1)1 + db + (d-1)b^2 + ... + 2b^{d-1} + 1b^d$

Do đó thời gian tìm kiếm sâu lặp là $O(b^d)$.

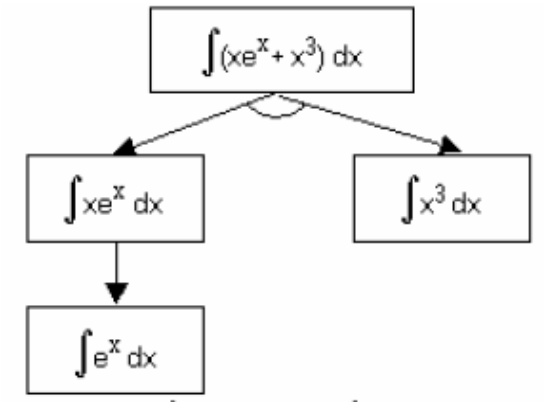
Tóm lại, tìm kiếm sâu lặp có độ phức tạp thời gian là $O(b^d)$ (như tìm kiếm theo bề rộng), và có độ phức tạp không gian là O(biểu diễn) (như tìm kiếm theo độ sâu). Nói chung, chúng ta nên áp dụng tìm kiếm sâu lặp cho các vấn đề có không gian trạng thái lớn và độ sâu của nghiệm không biết trước.

IV. Quy vấn đề về các vấn đề con

Chúng ta đã nghiên cứu việc biểu diễn vấn đề thông qua các trạng thái và các toán tử. Khi đó việc tìm nghiệm của vấn đề được quy về việc tìm đường trong không gian trạng thái. Trong mục này chúng ta sẽ nghiên cứu một phương pháp luận khác để giải quyết vấn đề, dựa trên việc quy vấn đề về các vấn đề con. Quy vấn đề về các vấn đề con (còn gọi là rút gọn vấn đề) là một phương pháp được sử dụng rộng rãi nhất để giải quyết các vấn đề. Trong đời sống hàng ngày, cũng như trong khoa học kỹ thuật, mỗi khi gặp một vấn đề cần giải quyết, ta vẫn thường cố gắng tìm cách đưa nó về các vấn đề đơn giản hơn. Quá trình rút gọn vấn đề sẽ được tiếp tục cho tới khi ta dẫn tới các vấn đề con có thể giải quyết được dễ dàng. Sau đây chúng ta xét một số vấn đề.

- Vấn đề tính tích phân bất định

Giả sử ta cần tính một tích phân bất định, chẳng hạn $\int(xe^x + x^3)\,dx$. Quá trình chúng ta vẫn thường làm để tính tích phân bất định là như sau. Sử dụng các quy tắc tính tích phân (quy tắc tính tích phân của một tổng, quy tắc tính tích phân từng phần...), sử dụng các phép biến đổi biến số, các phép biến đổi các hàm (chẳng hạn, các phép biến đổi lượng giác),... để đưa tích phân cần tính về tích phân của các hàm số sơ cấp mà chúng ta đã biết cách tính. Chẳng hạn, đối với tích phân $\int(xe^x + x^3)\,dx$, áp dụng quy tắc tích phân của tổng ta đưa về hai tích phân $\int xe^x dx$ và $\int x^3 dx$. Áp dụng quy tắc tích phân từng phần ta đưa tích phân $\int xe^x dx$ về tích phân $\int e^x dx$. Quá trình trên có thể biểu diễn bởi đồ thị trong hình 3.5.



Hình 3.5 Quy một số tích phân về tích phân cơ bản

Các tích phân $\int e^x dx$ và $\int x^3 dx$ là các tích phân cơ bản đã có trong bảng tích phân. Kết hợp các kết quả của các tích phân cơ bản, ta nhận được kết quả của tích phân đã cho.

Chúng ta có thể biểu diễn việc quy một vấn đề về các vấn đề con cơ bởi các trạng thái và các toán tử. ở đây, bài toán cần giải là trạng thái ban đầu. Mỗi cách quy bài toán về các bài toán con được biểu diễn bởi một toán tử, toán tử A→B, C biểu diễn việc quy bài toán A về hai bài toán B và C. Chẳng hạn, đối với bài toán tính tích phân bất định, ta có thể xác định các toán tử dạng:

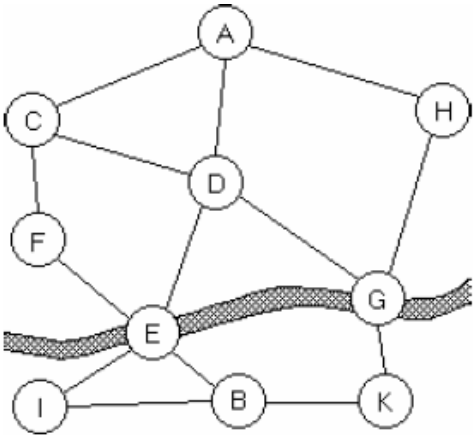
$\int(f_1 + f_2)\,dx \rightarrow \int f_1\,dx$, $\int f_2\,dx$ và $\int u\,dv \rightarrow \int v\,du$

Các trạng thái kết thúc là các bài toán sơ cấp (các bài toán đã biết cách giải). Chẳng hạn, trong bài toán tính tích phân, các tích phân cơ bản là các trạng thái kết thúc. Một

điều cần lưu ý là, trong không gian trạng thái biểu diễn việc quy vấn đề về các vấn đề con, các toán tử có thể là đa trị, nó biến đổi một trạng thái thành nhiều trạng thái khác.

- Vấn đề tìm đường đi trên bản đồ giao thông

Bài toán này đã được phát triển như bài toán tìm đường đi trong không gian trạng thái (xem 3.1), trong đó mỗi trạng thái ứng với một thành phố, mỗi toán tử ứng với một con đường nối, nối thành phố này với thành phố khác. Bây giờ ta đưa ra một cách biểu diễn khác dựa trên việc quy vấn đề về các vấn đề con.



Hình 3.6

Giả sử ta có bản đồ giao thông trong một vùng lãnh thổ (xem hình 3.6). Giả sử ta cần tìm đường đi từ thành phố A tới thành phố B. Có con sông chảy qua hai thành phố E và G và có cầu qua sông ở mỗi thành phố đó. Mọi đường đi từ A đến B chỉ có thể qua E hoặc G. Như vậy bài toán tìm đường đi từ A đến B được quy về:

1) Bài toán tìm đường đi từ A đến B qua E (hoặc)

2) Bài toán tìm đường đi từ A đến b qua G.

Mỗi một trong hai bài toán trên lại có thể phân nhỏ như sau

1) Bài toán tìm đường đi từ A đến B qua E được quy về:

1.1 Tìm đường đi từ A đến E (và)

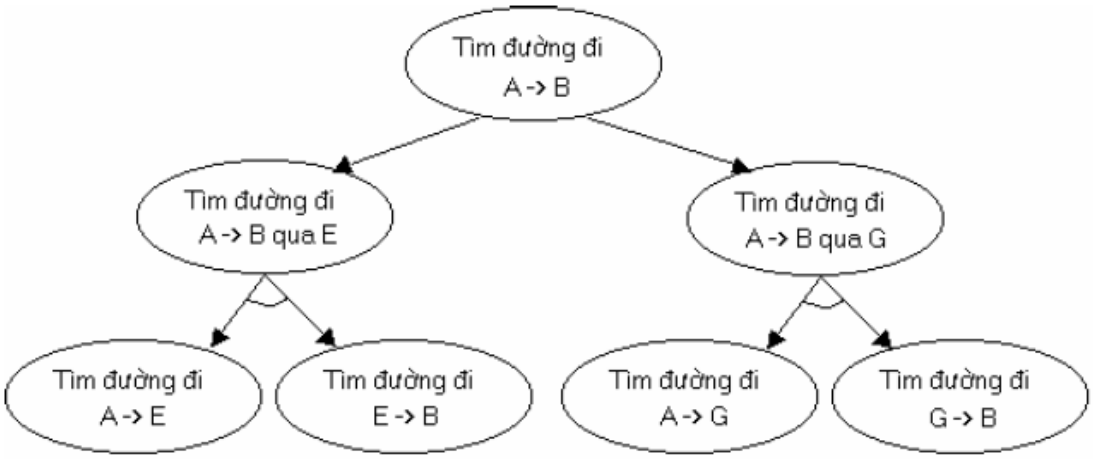
1.2 Tìm đường đi từ E đến B.

2) Bài toán tìm đường đi từ A đến B qua G được quy về:

2.1 Tìm đường đi từ A đến G (và)

2.2 Tìm đường đi từ G đến B.

Quá trình rút gọn vấn đề như trên có thể biểu diễn dưới dạng đồ thị (đồ thị và/hoặc) trong hình 3.7. ở đây mỗi bài toán tìm đường đi từ một thành phố tới một thành phố khác ứng với một trạng thái. Các trạng thái kết thúc là các trạng thái ứng với các bài toán tìm đường đi, chẳng hạn từ A đến C, hoặc từ D đến E, bởi vì đã có đường nối A với C, nối D với E.

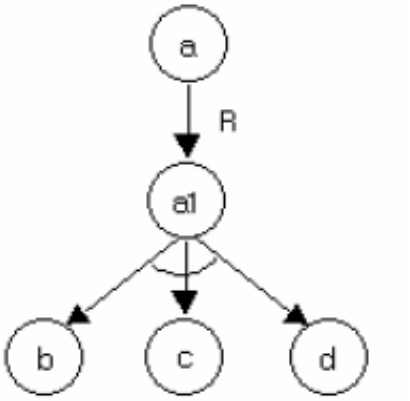


Hình 3.7 Đồ thị và/hoặc của vấn đề tìm đường đi

V. Đồ thị

Không gian trạng thái mô tả việc quy vấn đề về các vấn đề con có thể biểu diễn dưới dạng đồ thị định hướng đặc biệt được gọi là đồ thị và/hoặc. Đồ thị này được xây dựng như sau:

Mỗi bài toán ứng với một đỉnh của đồ thị. Nếu có một toán tử quy một bài toán về một bài toán khác, chẳng hạn R : a → b, thì trong đồ thị sẽ có cung gán nhãn đi từ đỉnh a tới đỉnh b. Đối với mỗi toán tử quy một bài toán về một số bài toán con, chẳng hạn R : a → b, c, d ta đưa vào một đỉnh mới a_1, đỉnh này biểu diễn tập các bài toán con {b, c, d} và toán tử R : a → b, c, d được biểu diễn bởi đồ thị hình 3.8.



Hình 3.8 Đồ thị biểu diễn toán tử R : a → b, c, d

Ví dụ: Giả sử chúng ta có không gian trạng thái sau:

- Trạng thái ban đầu (bài toán cần giải) là a.
- Tập các toán tử gồm:

R_1 : a → d, e, f

R_2 : a → d, k

R_3 : a → g, h

R_4 : d → b, c

R_5 : f → i

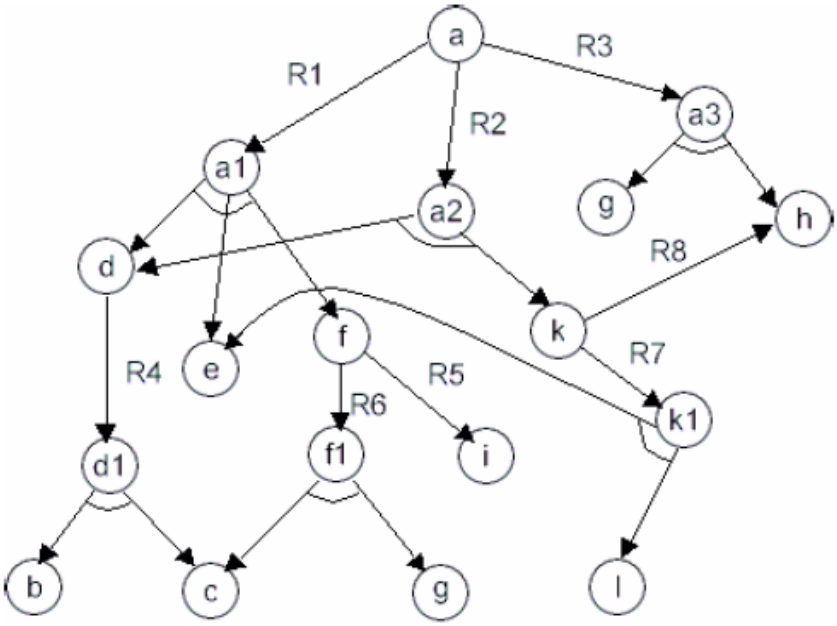
R_6 : f → c, g

R_7 : k → e, l

R_8 : k → h

• Tập các trạng thái kết thúc (các bài toán sơ cấp) là T = {b, c, e, g}.

Không gian trạng thái trên có thể biểu diễn bởi đồ thị và/hoặc trong hình 3.9. Trong đồ thị đó, các đỉnh chẳng hạn a, f, k hoặc a_1, a_2, a_3 được gọi là đỉnh. Lý do là, đỉnh a_1 biểu diễn tập các bài toán {d, e, f} và a_1 được giải quyết nếu d và e và f được giải quyết. Còn tại đỉnh a, ta có các toán tử R_1, R_2, R_3 quy bài toán a về các bài toán con khác nhau, do đó a được giải quyết nếu hoặc a_1 = {d, e, f}, hoặc a_2 = {d, k}, hoặc a_3 = {g, h} được giải quyết.



Hình 3.9 Một đồ thị và/hoặc

Người ta thường sử dụng đồ thị và/hoặc ở dạng rút gọn. Chẳng hạn, đồ thị và/hoặc trong hình 3.9 có thể rút gọn thành đồ thị trong hình 3.10. Trong đồ thị rút gọn này, ta sẽ nói chẳng hạn d, e, f là các đỉnh kề đỉnh a theo toán tử R_1, còn d, k là các đỉnh kề a theo toán tử R_2.

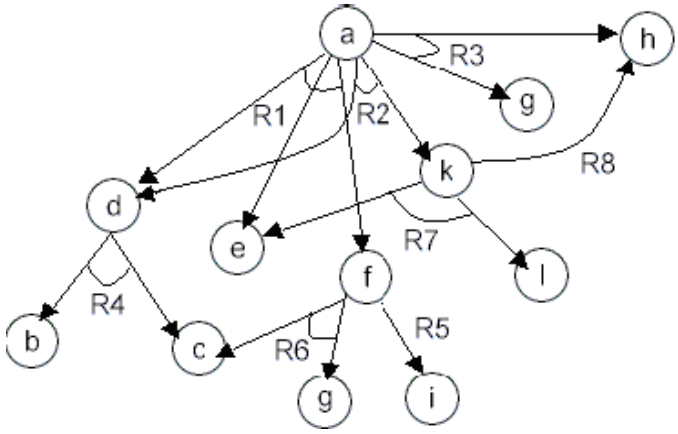
Khi đã có các toán tử rút gọn vấn đề, thì bằng cách áp dụng liên tiếp các toán tử, ta có thể đưa bài toán cần giải về một tập các bài toán con. Chẳng hạn, trong ví dụ trên nếu ta áp dụng các toán tử R_1, R_4, R_6, ta sẽ quy bài toán a về tập các bài toán con {b, c, e, g}, tất cả các bài toán con này đều là sơ cấp. Từ các toán tử R_1, R_4 và R_6 ta xây dựng được một cây trong hình 1.11a, cây này được gọi là cây nghiệm. Cây nghiệm được định nghĩa như sau:

Cây nghiệm là một cây, trong đó:

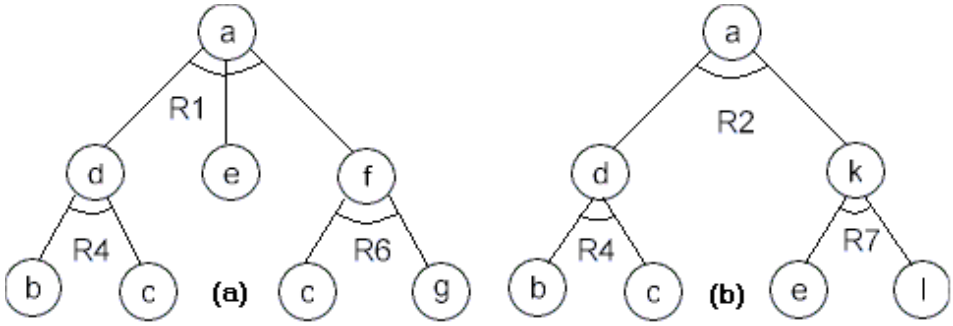
- Gốc của cây ứng với bài toán cần giải.
- Tất cả các lá của cây là các đỉnh kết thúc (đỉnh ứng với các bài toán sơ cấp).

- Nếu u là đỉnh trong của cây, thì các đỉnh con của u là các đỉnh kề u theo một toán tử nào đó.

Các đỉnh của đồ thị và/hoặc sẽ được gắn nhãn giải được hoặc không giải được.



Hình 3.10 Đồ thị rút gọn của đồ thị trong hình 3.9



Hình 3.11 Các cây nghiệm

Các đỉnh **giải được** được xác định đệ quy như sau:

- Các đỉnh kết thúc là các đỉnh giải được.
- Nếu u không phải là đỉnh kết thúc, nhưng có một toán tử R sao cho tất cả các đỉnh kề u theo R đều giải được thì u giải được.

Các đỉnh **không giải được** được xác định đệ quy như sau:

- Các đỉnh không phải là đỉnh kết thúc và không có đỉnh kề, là các đỉnh không giải được.
- Nếu u không phải là đỉnh kết thúc và với mọi toán tử R áp dụng được tại u đều có một đỉnh v kề u theo R không giải được, thì u không giải được.

Ta có nhận xét rằng, nếu bài toán a giải được thì sẽ có một cây nghiệm gốc a, và ngược lại nếu có một cây nghiệm gốc a thì a giải được. Hiển nhiên là, một bài toán giải được có thể có nhiều cây nghiệm, mỗi cây nghiệm biểu diễn một cách giải bài toán đó. Chẳng hạn trong ví dụ đã nêu, bài toán a có hai cây nghiệm trong hình 3.11.

Thứ tự giải các bài toán con trong một cây nghiệm là như sau. Bài toán ứng với đỉnh u chỉ được giải sau khi tất cả các bài toán ứng với các đỉnh con của u đã được giải.

Chẳng hạn, với cây nghiệm trong hình 3.11a, thứ tự giải các bài toán có thể là b, c, d, g, f, e, a. ta có thể sử dụng thủ tục sắp xếp topo để sắp xếp thứ tự các bài toán trong một cây nghiệm. Đương nhiên ta cũng có thể giải quyết đồng thời các bài toán con ở cùng một mức trong cây nghiệm.

Vấn đề của chúng ta bây giờ là, tìm kiếm trên đồ thị và/hoặc để xác định được đỉnh ứng với bài toán ban đầu là giải được hay không giải được, và nếu nó giải được thì xây dựng một cây nghiệm cho nó.

- Tìm kiếm trên đồ thị và/hoặc

Ta sẽ sử dụng kỹ thuật tìm kiếm theo độ sâu trên đồ thị và/hoặc để đánh dấu các đỉnh. Các đỉnh sẽ được đánh dấu giải được hoặc không giải được theo định nghĩa đệ quy về đỉnh giải được và không giải được. Xuất phát từ đỉnh ứng với bài toán ban đầu, đi xuống theo độ sâu, nếu gặp đỉnh u là đỉnh kết thúc thì nó được đánh dấu giải được. Nếu gặp đỉnh u không phải là đỉnh kết thúc và từ u không đi tiếp được, thì u được đánh dấu không giải được. Khi đi tới đỉnh u, thì từ u ta lần lượt đi xuống các đỉnh v kề u theo một toán tử R nào đó. Nếu đánh dấu được một đỉnh v không giải được thì không cần đi tiếp xuống các đỉnh v còn lại. Tiếp tục đi xuống các đỉnh kề u theo một toán tử khác. Nếu tất cả các đỉnh kề u theo một toán tử nào đó được đánh dấu giải được thì u sẽ được đánh dấu giải được và quay lên cha của u. Còn nếu từ u đi xuống các đỉnh kề nó theo mọi toán tử đều gặp các đỉnh kề được đánh dấu không giải được, thì u được đánh dấu không giải được và quay lên cha của u.

Ta sẽ biểu diễn thủ tục tìm kiếm theo độ sâu và đánh dấu các đỉnh đã trình bày trên bởi hàm đệ quy Solvable(u). Hàm này nhận giá trị **true** nếu u giải được và nhận giá trị **false** nếu u không giải được. Trong hàm Solvable(u), ta sẽ sử dụng:

- Biến Ok. Với mỗi toán tử R áp dụng được tại u, biến Ok nhận giá trị true nếu tất cả các đỉnh v kề u theo R đều giải được, và Ok nhận giá trị false nếu có một đỉnh v kề u theo R không giải được.
- Hàm Operator(u) ghi lại toán tử áp dụng thành công tại u, tức là Operator(u) = R nếu mọi đỉnh v kề u theo R đều giải được.

function Solvable(u);

begin

1. **if** u là đỉnh kết thúc **then**

    {Solvable(u) ←**true**; **stop**};

2. **if** u không là đỉnh kết thúc và không có đỉnh kề **then**

    {Solvable(u) ← **false**; **stop**};

3. **for** mỗi toán tử R áp dụng được tại u **do** (1)

    {Ok ← **true**;

    for mỗi v kề u theo R **do** (2)

        if Solvable(v) = **false** then {Ok ← **false**; **exit**}; // thoát for (2)

    if Ok **then**

{Solvable(u) ← **true**; Operator(u) ← R; **stop**}

}

4. Solvable(u) ← **false**;

end;

Nhận xét

Hoàn toàn tương tự như thuật toán tìm kiếm theo độ sâu trong không gian trạng thái (mục III.2), thuật toán tìm kiếm theo độ sâu trên đồ thị và/hoặc sẽ xác định được bài toán ban đầu là giải được hay không giải được, nếu cây tìm kiếm không có nhánh vô hạn. Nếu cây tìm kiếm có nhánh vô hạn thì chưa chắc thuật toán đã dừng, vì có thể nó bị xa lầy khi đi xuống nhánh vô hạn. Trong trường hợp này ta nên sử dụng thuật toán tìm kiếm sâu lặp (mục III.4). Nếu bài toán ban đầu giải được, thì bằng cách sử dụng hàm Operator ta sẽ xây dựng được cây nghiệm.

VI. Bài tập

1. Tính số trạng thái tối đa phải lưu khi duyệt một cây theo bề rộng có độ sâu là 5 và hệ số nhánh là 4
2. Viết chương trình minh họa tìm kiếm theo bề rộng và độ sâu của bài toán 8 số với các yêu cầu sau
   a) Trạng thái ban đầu của bài toán có các số ở vị trí ngẫu nhiên
   b) Trình bày tất cả trạng thái trong quá trình tìm trạng thái đích
3. Viết chương trình minh họa bài toán nhà triệu phú và kẻ cướp với số nhà triệu phú và kẻ cướp tuỳ ý
4. Viết chương trình giải bài toán tích phân $\int(xe^x + x^3)\,dx$

Chương IV

CÁC CHIẾN LƯỢC TÌM KIẾM KINH NGHIỆM

Trong chương III, chúng ta đã nghiên cứu việc biểu diễn vấn đề trong không gian trạng thái và các kỹ thuật tìm kiếm mù. Các kỹ thuật tìm kiếm mù rất kém hiệu quả và trong nhiều trường hợp không thể áp dụng được. Trong chương này, chúng ta sẽ nghiên cứu các phương pháp tìm kiếm kinh nghiệm (tìm kiếm heuristic), đó là các phương pháp sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm.

I. Hàm đánh giá và tìm kiếm kinh nghiệm

Trong nhiều vấn đề, ta có thể sử dụng kinh nghiệm, tri thức của chúng ta về vấn đề để đánh giá các trạng thái của vấn đề. Với mỗi trạng thái u, chúng ta sẽ xác định một giá trị số h(u), số này đánh giá "sự gần đích" của trạng thái u. Hàm h(u) được gọi là hàm đánh giá. Chúng ta sẽ sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm. Trong quá trình tìm kiếm, tại mỗi bước ta sẽ chọn trạng thái để phát triển là trạng thái có giá trị hàm đánh giá nhỏ nhất, trạng thái này được xem là trạng thái có nhiều hứa hẹn nhất hướng tới đích.

Các kỹ thuật tìm kiếm sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm được gọi chung là các kỹ thuật tìm kiếm kinh nghiệm (heuristic search). Các giai đoạn cơ bản để giải quyết vấn đề bằng tìm kiếm kinh nghiệm như sau:

1. Tìm biểu diễn thích hợp mô tả các trạng thái và các toán tử của vấn đề.

2. Xây dựng hàm đánh giá.

3. Thiết kế chiến lược chọn trạng thái để phát triển ở mỗi bước.

- Hàm đánh giá

Trong tìm kiếm kinh nghiệm, hàm đánh giá đóng vai trò cực kỳ quan trọng. Chúng ta có xây dựng được hàm đánh giá cho ta sự đánh giá đúng các trạng thái thì tìm kiếm mới hiệu quả. Nếu hàm đánh giá không chính xác, nó có thể dẫn ta đi chệch hướng và do đó tìm kiếm kém hiệu quả.

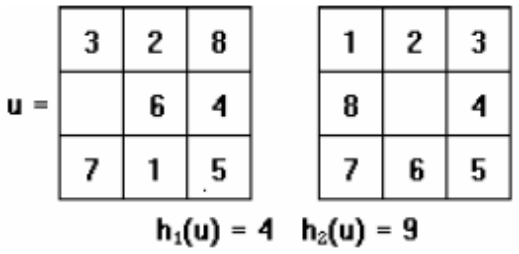
Hàm đánh giá được xây dựng tùy thuộc vào vấn đề. Sau đây là một số ví dụ về hàm đánh giá:

• Trong bài toán tìm kiếm đường đi trên bản đồ giao thông, ta có thể lấy độ dài của đường chim bay từ một thành phố tới một thành phố đích làm giá trị của hàm đánh giá.

• Bài toán 8 số. Chúng ta có thể đưa ra hai cách xây dựng hàm đánh giá.

Hàm h_1: Với mỗi trạng thái u thì $h_1(u)$ là số quân không nằm đúng vị trí của nó trong trạng thái đích.

Chẳng hạn trạng thái đích ở bên phải hình 4.1, và u là trạng thái ở bên trái hình 4.1, thì $h_1(u) = 4$, vì các quân không đúng vị trí là 3, 8, 6 và 1.



Hình 4.1 Đánh giá trạng thái u

Hàm h_2: $h_2(u)$ là tổng khoảng cách giữa vị trí của các quân trong trạng thái u và vị trí của nó trong trạng thái đích. ở đây khoảng cách được hiểu là số ít nhất các dịch chuyển theo hàng hoặc cột để đưa một quân tới vị trí của nó trong trạng thái đích. Chẳng hạn với trạng thái u và trạng thái đích như trong hình 2.1, ta có:

$h_2(u) = 2 + 3 + 1 + 3 = 9$

Vì quân 3 cần ít nhất 2 dịch chuyển, quân 8 cần ít nhất 3 dịch chuyển, quân 6 cần ít nhất 1 dịch chuyển và quân 1 cần ít nhất 3 dịch chuyển.

Hai chiến lược tìm kiếm kinh nghiệm quan trọng nhất là tìm kiếm tốt nhất - đầu tiên (best-first search) và tìm kiếm leo đồi (hill-climbing search). Có thể xác định các chiến lược này như sau:

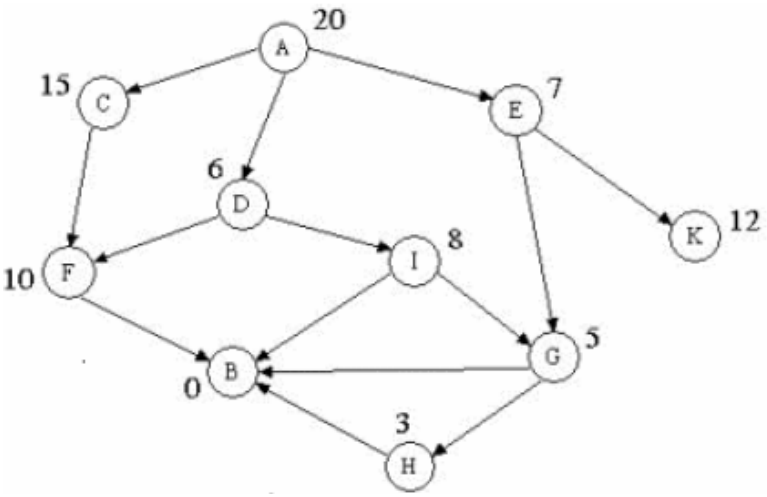
Tìm kiếm tốt nhất đầu tiên = Tìm kiếm theo bề rộng + Hàm đánh giá

Tìm kiếm leo đồi = Tìm kiếm theo độ sâu + Hàm đánh giá

Chúng ta sẽ lần lượt nghiên cứu các kỹ thuật tìm kiếm này trong các mục sau.

II. Tìm kiếm tốt nhất - đầu tiên

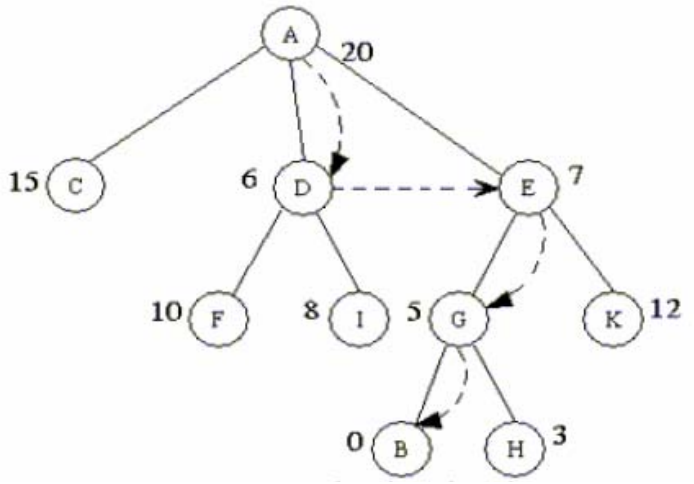
Tìm kiếm tốt nhất - đầu tiên (best-first search) là tìm kiếm theo bề rộng được hướng dẫn bởi hàm đánh giá. Nhưng nó khác với tìm kiếm theo bề rộng ở chỗ, trong tìm kiếm theo bề rộng ta lần lượt phát triển tất cả các đỉnh ở mức hiện tại để sinh ra các đỉnh ở mức tiếp theo, còn trong tìm kiếm tốt nhất - đầu tiên ta chọn đỉnh để phát triển là đỉnh tốt nhất được xác định bởi hàm đánh giá (tức là đỉnh có giá trị hàm đánh giá là nhỏ nhất), đỉnh này có thể ở mức hiện tại hoặc ở các mức trên.



Hình 4.2 Đồ thị không gian trạng thái

Ví dụ: Xét không gian trạng thái được biểu diễn bởi đồ thị trong hình 4.2, trong đó trạng thái ban đầu là A, trạng thái kết thúc là B. Giá trị của hàm đánh giá là các số ghi cạnh mỗi đỉnh. Quá trình tìm kiếm tốt nhất - đầu tiên diễn ra như sau: Đầu tiên phát triển đỉnh A sinh ra các đỉnh kề là C, D và E. Trong ba đỉnh này, đỉnh D có giá trị hàm đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra F, I. Trong số các đỉnh chưa được phát triển C, E, F, I thì đỉnh E có giá trị đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra các đỉnh G, K. Trong số các đỉnh chưa được phát triển thì G tốt nhất, phát triển G sinh ra B, H.

Đến đây ta đã đạt tới trạng thái kết thúc. Cây tìm kiếm tốt nhất - đầu tiên được biểu diễn trong hình 4.3.



Hình 4.3 Cây tìm kiếm tốt nhất - đầu tiên

Sau đây là thủ tục tìm kiếm tốt nhất - đầu tiên. Trong thủ tục này, chúng ta sử dụng danh sách L để lưu các trạng thái chờ phát triển, danh sách được sắp theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái có giá trị hàm đánh giá nhỏ nhất ở đầu danh sách.

procedure Best_First_Search;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

    2.1 **if** L rỗng **then**

        {thông báo thất bại; **stop**};

    2.2 Loại trạng thái u ở đầu danh sách L;

    2.3 **if** u là trạng thái kết thúc **then**

        {thông báo thành công; **stop**}

    2.4 **for** mỗi trạng thái v kề u **do**

        Xen v vào danh sách L sao cho L được sắp theo thứ tự tăng dần của hàm đánh giá;

end;

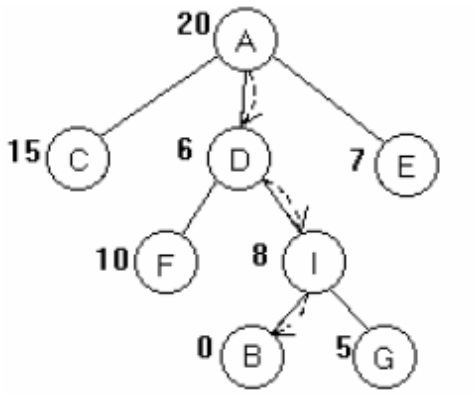
III. Tìm kiếm leo đồi

Tìm kiếm leo đồi (hill-climbing search) là tìm kiếm theo độ sâu được hướng dẫn bởi hàm đánh giá.

Song khác với tìm kiếm theo độ sâu, khi ta phát triển một đỉnh u thì bước tiếp theo, ta chọn trong số các đỉnh con của u, đỉnh có nhiều hứa hẹn nhất để phát triển, đỉnh này được xác định bởi hàm đánh giá.

Ví dụ: Ta lại xét đồ thị không gian trạng thái trong hình 2.2. Quá trình tìm kiếm leo đồi được tiến hành như sau. Đầu tiên phát triển đỉnh A sinh ra các đỉnh con C, D, E. Trong các đỉnh này chọn D để phát triển, và nó sinh ra các đỉnh con B, G. Quá trình tìm kiếm kết thúc. Cây tìm kiếm leo đồi được cho trong hình 4.4.

Trong thủ tục tìm kiếm leo đồi được trình bày dưới đây, ngoài danh sách L lưu các trạng thái chờ được phát triển, chúng ta sử dụng danh sách L_1 để lưu giữ tạm thời các trạng thái kề trạng thái u, khi ta phát triển u. Danh sách L_1 được sắp xếp theo thứ tự tăng dần của hàm đánh giá, rồi được chuyển vào danh sách L sao trạng thái tốt nhất kề u đứng ở danh sách L.



Hình 4.4 Cây tìm kiếm leo đồi

procedure Hill_Climbing_Search;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

    2.1 **if** L rỗng **then**

        {thông báo thất bại; **stop**};

    2.2 Loại trạng thái u ở đầu danh sách L;

    2.3 **if** u là trạng thái kết thúc **then**

        {thông báo thành công; **stop**};

    2.4 **for** mỗi trạng thái v kề u do đặt v vào L_1;

    2.5 Sắp xếp L_1 theo thứ tự tăng dần của hàm đánh giá;

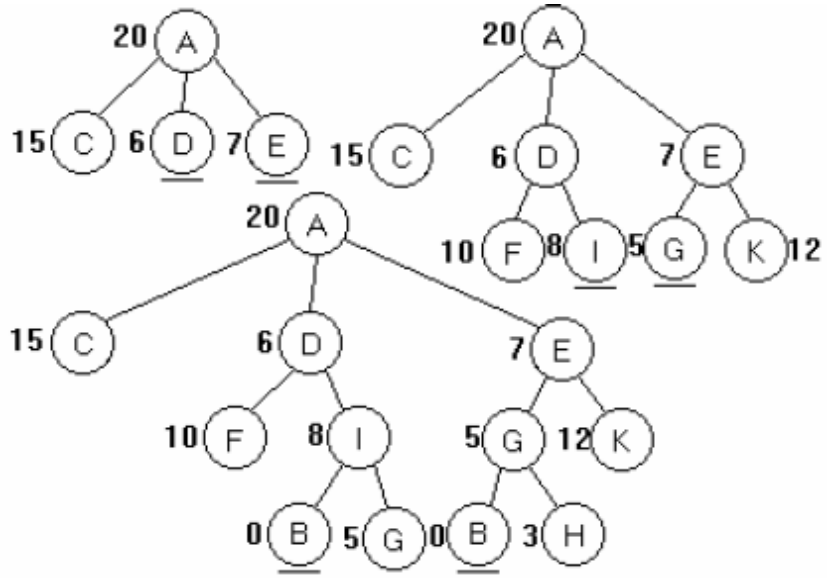
    2.6 Chuyển danh sách L_1 vào đầu danh sách L;

end;

IV. Tìm kiếm beam

Tìm kiếm beam (beam search) giống như tìm kiếm theo bề rộng, nó phát triển các đỉnh ở một mức rồi phát triển các đỉnh ở mức tiếp theo. Tuy nhiên, trong tìm kiếm theo bề rộng, ta phát triển tất cả các đỉnh ở một mức, còn trong tìm kiếm beam, ta hạn chế chỉ phát triển k đỉnh tốt nhất (các đỉnh này được xác định bởi hàm đánh giá). Do đó trong tìm kiếm beam, ở bất kỳ mức nào cũng chỉ có nhiều nhất k đỉnh được phát triển, trong khi tìm kiếm theo bề rộng, số đỉnh cần phát triển ở mức d là bd (b là nhân tố nhánh).

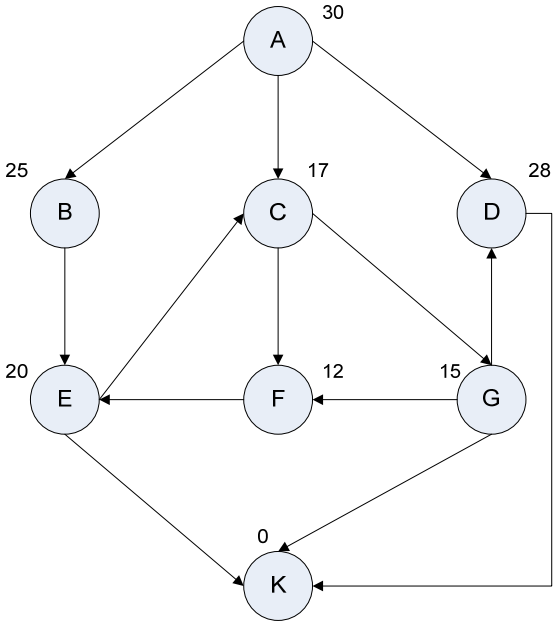
Ví dụ: Chúng ta lại xét đồ thị không gian trạng thái trong hình 4.2. Chọn k = 2. Khi đó cây tìm kiếm beam được cho như hình 4.5. Các đỉnh được gạch dưới là các đỉnh được chọn để phát triển ở mỗi mức.



Hình 4.5 Cây tìm kiếm beam

V. Bài tập

1. Trong tìm kiếm tốt nhất - đầu tiên nếu giá trị của hàm đánh giá của các trạng thái thay đổi sau mỗi bước chọn thì thủ tục tìm kiếm tốt nhất - đầu tiên cần thay đổi như thế nào ?
2. Viết chương trình minh họa tìm kiếm tốt nhất - đầu tiên và leo đồi và beam của bài toán 8 số với các yêu cầu sau
   a) Trạng thái ban đầu của bài toán có các số ở vị trí ngẫu nhiên
   b) Trình bày tất cả trạng thái trong quá trình tìm trạng thái đích
3. Viết giải thuật tìm kiếm beam (dùng mã giả)
4. Cho đồ thị không gian trạng thái:



Để tìm đường đi từ A tới K, hãy vẽ cây tìm kiếm của tìm kiếm tốt nhất-đầu tiên, tìm kiếm leo đồi và tìm kiếm beam có k = 2

Chương V

CÁC CHIẾN LƯỢC TÌM KIẾM TỐI ƯU

Vấn đề tìm kiếm tối ưu, một cách tổng quát, có thể phát biểu như sau. Mỗi đối tượng x trong không gian tìm kiếm được gắn với một số đo giá trị của đối tượng đó f(x), mục tiêu của ta là tìm đối tượng có giá trị f(x) lớn nhất (hoặc nhỏ nhất) trong không gian tìm kiếm. Hàm f(x) được gọi là hàm mục tiêu. Trong chương này chúng ta sẽ nghiên cứu các thuật toán tìm kiếm sau:

- Các kỹ thuật tìm đường đi ngắn nhất trong không gian trạng thái: Thuật toán A*, thuật toán nhánh_và_cận.
- Các kỹ thuật tìm kiếm đối tượng tốt nhất: Tìm kiếm leo đồi, tìm kiếm gradient, tìm kiếm mô phỏng luyện kim.
- Tìm kiếm bắt chước sự tiến hóa: thuật toán di truyền.

I. Tìm đường đi ngắn nhất

Trong các chương trước chúng ta đã nghiên cứu vấn đề tìm kiếm đường đi từ trạng thái ban đầu tới trạng thái kết thúc trong không gian trạng thái. Trong mục này, ta giả sử rằng, giá phải trả để đưa trạng thái a tới trạng thái b (bởi một toán tử nào đó) là một số $k(a,b) \geq 0$, ta sẽ gọi số này là độ dài cung (a,b) hoặc giá trị của cung (a,b) trong đồ thị không gian trạng thái. Độ dài của các cung được xác định tùy thuộc vào vấn đề. Chẳng hạn, trong bài toán tìm đường đi trong bản đồ giao thông, giá của cung (a,b) chính là độ dài của đường nối thành phố a với thành phố b. Độ dài đường đi được xác định là tổng độ dài của các cung trên đường đi. Vấn đề của chúng ta trong mục này, tìm đường đi ngắn nhất từ trạng thái ban đầu tới trạng thái đích. Không gian tìm kiếm ở đây bao gồm tất cả các đường đi từ trạng thái ban đầu tới trạng thái kết thúc, hàm mục tiêu được xác định ở đây là độ dài của đường đi.

Chúng ta có thể giải quyết vấn đề đặt ra bằng cách tìm tất cả các đường đi có thể có từ trạng thái ban đầu tới trạng thái đích (chẳng hạn, sử sụng các kỹ thuật tìm kiếm mù), sau đó so sánh độ dài của chúng, ta sẽ tìm ra đường đi ngắn nhất. Thủ tục tìm kiếm này thường được gọi là thủ tục bảo tàng Anh Quốc (British Museum Procedure). Trong thực tế, kỹ thuật này không thể áp dụng được, vì cây tìm kiếm thường rất lớn, việc tìm ra tất cả các đường đi có thể đòi hỏi rất nhiều thời gian. Do đó chỉ có một cách tăng hiệu quả tìm kiếm là sử dụng các hàm đánh giá để hướng dẫn tìm kiếm. Các phương pháp tìm kiếm đường đi ngắn nhất mà chúng ta sẽ trình bày đều là các phương pháp tìm kiếm heuristic.

Giả sử u là một trạng thái đạt tới (có dường đi từ trạng thái ban đầu u_0 tới u). Ta xác định hai hàm đánh giá sau:

- g(u) là đánh giá độ dài đường đi ngắn nhất từ u_0 tới u (Đường đi từ u_0 tới trạng thái u không phải là trạng thái đích được gọi là đường đi một phần, để phân biệt với đường đi đầy đủ, là đường đi từ u_0 tới trạng thái đích).

- h(u) là đánh giá độ dài đường đi ngắn nhất từ u tới trạng thái đích.

Hàm h(u) được gọi là chấp nhận được (hoặc đánh giá thấp) nếu với mọi trạng thái u, h(u) ≤ độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đích. Chẳng hạn trong bài toán tìm đường đi ngắn nhất trên bản đồ giao thông, ta có thể xác định h(u) là độ dài đường chim bay từ u tới đích.

Ta có thể sử dụng kỹ thuật tìm kiếm leo đồi với hàm đánh giá h(u). Tất nhiên phương pháp này chỉ cho phép ta tìm được đường đi tương đối tốt, chưa chắc đã là đường đi tối ưu.

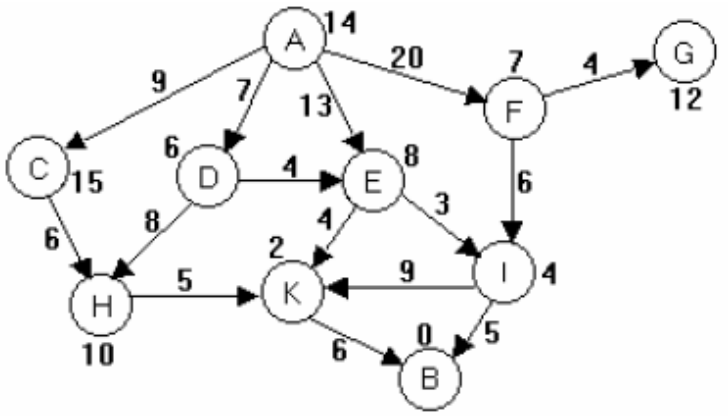
Ta cũng có thể sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u). Phương pháp này sẽ tìm ra đường đi ngắn nhất, tuy nhiên nó có thể kém hiệu quả.

Để tăng hiệu quả tìm kiếm, ta sử dụng hàm đánh giá mới :

f(u) = g(u) + h(u)

Tức là, f(u) là đánh giá độ dài đường đi ngắn nhất qua u từ trạng thái ban đầu tới trạng thái kết thúc.

I.1 Thuật toán A*



Hình 5.1 Đồ thị không gian trạng thái với hàm đánh giá

Thuật toán A* là thuật toán sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá f(u).

Để thấy được thuật toán A* làm việc như thế nào, ta xét đồ thị không gian trạng thái trong hình 5.1. Trong đó, trạng thái ban đầu là trạng thái A, trạng thái đích là B, các số ghi cạnh các cung là độ dài đường đi, các số cạnh các đỉnh là giá trị của hàm h. Đầu tiên, phát triển đỉnh A sinh ra các đỉnh con C, D, E và F.

Tính giá trị của hàm f tại các đỉnh này ta có:

g(C) = 9, f(C) = 9 + 15 = 24, g(D) = 7, f(D) = 7 + 6 = 13,

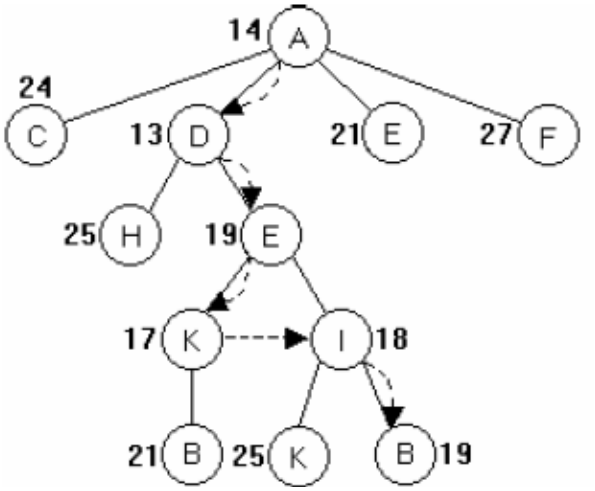
g(E) = 13, f(E) = 13 + 8 = 21, g(F) = 20, f(F) = 20 +7 = 27

Như vậy đỉnh tốt nhất là D (vì f(D) = 13 là nhỏ nhất). Phát triển D, ta nhận được các đỉnh con H và E. Ta đánh giá H và E (mới):

g(H) = g(D) + Độ dài cung(D, H) = 7 + 8 = 15, f(H) = 15 + 10 = 25.

Đường đi tới E qua D có độ dài: g(E) = g(D) + Độ dài cung(D, E) = 7 + 4 = 11.

Vậy đỉnh E mới có đánh giá là f(E) = g(E) + h(E) = 11 + 8 = 19. Trong số các đỉnh cho phát triển, thì đỉnh E với đánh giá f(E) = 19 là đỉnh tốt nhất. Phát triển đỉnh này, ta nhận được các đỉnh con của nó là K và I. Chúng ta tiếp tục quá trình trên cho tới khi đỉnh được chọn để phát triển là đỉnh kết thúc B, độ dài đường đi ngắn nhất tới B là g(B) = 19. Quá trình tìm kiếm trên được mô tả bởi cây tìm kiếm trong hình 5.2, trong đó các số cạnh các đỉnh là các giá trị của hàm đánh giá f(u).



Hình 5.2 Cây tìm kiếm theo thuật toán A*

procedure A*;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

    2.1 **if** L rỗng **then**

        {thông báo thất bại; **stop**};

    2.2 Loại trạng thái u ở đầu danh sách L;

    2.3 **if** u là trạng thái đích **then**

        {thông báo thành công; **stop**}

    2.4 **for** mỗi trạng thái v kề u **do**

        {g(v) ← g(u) + k(u,v);

        f(v) ← g(v) + h(v);

        Đặt v vào danh sách L;}

    2.5 Sắp xếp L theo thứ tự tăng dần của hàm f sao cho trạng thái có giá trị của hàm f nhỏ nhất ở đầu danh sách;

end;

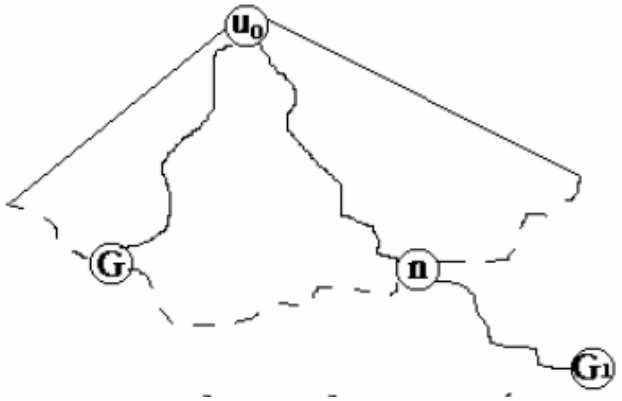
Chúng ta đưa ra một số nhận xét về thuật toán A*.

• Người ta chứng minh được rằng, nếu hàm đánh giá h(u) là đánh giá thấp nhất (trường hợp đặc biệt, h(u) = 0 với mọi trạng thái u) thì thuật toán A* là thuật toán tối ưu, tức là

nghiệm mà nó tìm ra là nghiệm tối ưu. Ngoài ra, nếu độ dài của các cung không nhỏ hơn một số dương δ nào đó thì thuật toán A* là thuật toán đầy đủ theo nghĩa rằng, nó luôn dừng và tìm ra nghiệm.

Chúng ta chứng minh tính tối ưu của thuật toán A*.

Giả sử thuật toán dừng lại ở đỉnh kết thúc G với độ dài đường đi từ trạng thái ban đầu u_0 tới G là g(G). Vì G là đỉnh kết thúc, ta có h(G) = 0 và f(G) = g(G) + h(G) = g(G). Giả sử nghiệm tối ưu là đường đi từ u_0 tới đỉnh kết thúc G_1 với độ dài l. Giả sử đường đi này "thoát ra" khỏi cây tìm kiếm tại đỉnh lá n (Xem hình 5.3). Có thể xẩy ra hai khả năng: n trùng với G_1 hoặc không. Nếu n là G_1 thì vì G được chọn để phát triển trước G_1, nên $f(G) \leq f(G_1)$, do đó $g(G) \leq g(G_1) = l$. Nếu $n \neq G_1$ thì do h(u) là hàm đánh giá thấp, nên $f(n) = g(n) + h(n) \leq l$. Mặt khác, cũng do G được chọn để phát triển trước n, nên $f(G) \leq f(n)$, do đó, $g(G) \leq l$.



Hình 5.3 Đỉnh lá n của cây tìm kiếm nằm trên đường đi tối ưu

Như vậy, ta đã chứng minh được rằng độ dài của đường đi mà thuật toán tìm ra g(G) không dài hơn độ dài l của đường đi tối ưu. Vậy nó là độ dài đường đi tối ưu.

• Trong trường hợp hàm đánh giá g(u) = 0 với mọi u, thuật toán A* chính là thuật toán tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u) mà ta đã nói đến.

• Thuật toán A* đã được chứng tỏ là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho vấn đề tìm kiếm đường đi ngắn nhất.

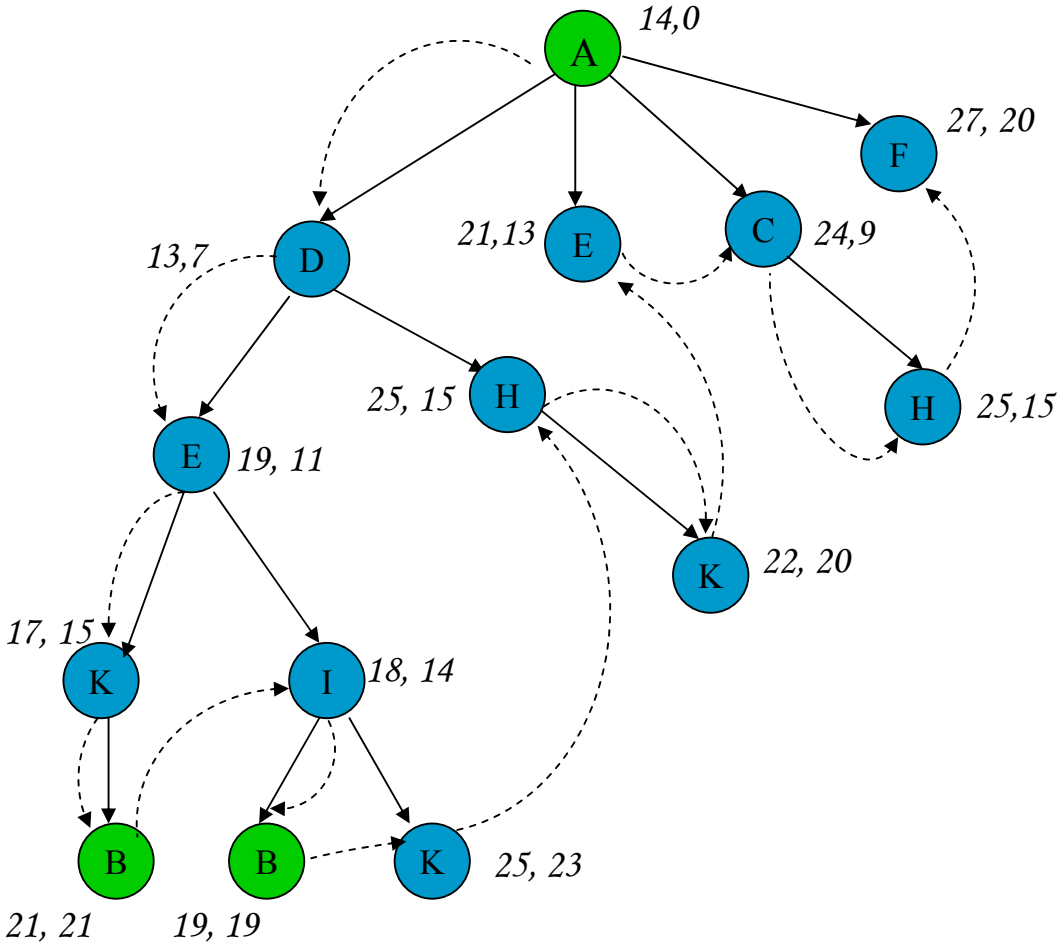
I.2 Thuật toán tìm kiếm nhánh-và-cận

Thuật toán nhánh_và_cận là thuật toán sử dụng tìm kiếm leo đồi với hàm đánh giá f(u).

Trong thuật toán này, tại mỗi bước khi phát triển trạng thái u, thì ta sẽ chọn trạng thái tốt nhất v (f(v) nhỏ nhất) trong số các trạng thái kề u để phát triển ở bước sau. Đi xuống cho tới khi gặp trạng thái v là đích, hoặc gặp trạng thái v không có đỉnh kề, hoặc gặp trạng thái v mà g(v) lớn hơn độ dài đường đi tối ưu tạm thời, tức là đường đi đầy đủ ngắn nhất trong số các đường đi đầy đủ mà ta đã tìm ra, hoặc v đã phát triển nhưng g(v) hiện tại lớn hơn g(v) được lưu, tức đường đi mới qua v không tốt hơn đường đi cũ qua v. Trong các trường hợp này, ta không phát triển đỉnh v nữa, hay nói cách khác, ta quay lên cha của v để tiếp tục đi xuống trạng thái tốt nhất trong các trạng thái còn lại chờ phát triển.

Ví dụ: Chúng ta lại xét không gian trạng thái trong hình 3.1. Phát triển đỉnh A, ta nhận được các đỉnh con C, D, E và F, f(C) = 24, f(D) = 13, f(E) = 21, f(F) = 27. Trong số này D là tốt nhất, phát triển D, sinh ra các đỉnh con H và E, f(H) = 25, f(E) = 19. Đi xuống phát triển E, sinh ra các đỉnh con là K và I, f(K) = 17, f(I) = 18. Đi xuống phát triển K sinh ra đỉnh B với f(B) = g(B) = 21. Đi xuống B, vì B là đỉnh đích, vậy ta tìm được đường đi tối ưu tạm thời với độ dài 21. Từ B quay lên K, rồi từ K quay lên cha nó là E. Từ E đi xuống I, f(I) = 18 nhỏ hơn độ dài đường đi tạm thời (là 21). Phát triển I sinh ra các con K và B, f(K) = 25, f(B) = g(B) = 19. Đi xuống đỉnh B, vì đỉnh B là đích ta tìm được đường đi đầy đủ mới với độ dài là 19 nhỏ hơn độ dài đường đi tối ưu tạm thời cũ (21). Vậy độ dài đường đi tối ưu tạm thời bây giờ là 19.

Bây giờ từ B ta lại quay lên các đỉnh còn lại chưa được phát triển. Song các đỉnh này đều có giá trị g lớn hơn 19, do đó không có đỉnh nào được phát triển nữa. Như vậy, ta tìm được đường đi tối ưu với độ dài 19. Cây tìm kiếm được biểu diễn trong hình 5.4.



Hình 5.4 Cây tìm kiếm nhánh_và_cận

Thuật toán nhánh_và_cận sẽ được biểu diễn bởi thủ tục Branch_and_Bound. Trong thủ tục này, biến cost được dùng để lưu độ dài đường đi ngắn nhất. Giá trị ban đầu của cost là số đủ lớn, hoặc độ dài của một đường đi đầy đủ mà ta đã biết.

procedure Branch_and_Bound;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu; //các trạng thái chưa phát triển và // chờ duyệt;
2. Khởi tạo danh sách M rỗng; //các trạng thái đã phát triển;
3. Gán giá trị ban đầu cho cost;
4. **loop do**

    4.1 **if** L rỗng **then** stop;

    4.2 Loại trạng thái u ở đầu danh sách L;

    4.3 **if** u là trạng thái kết thúc **then**

        {**if** g(u) ≤ cost **then** cost ← g(u);

        Quay lại 4.1};

    4.4 **if** g(u) > cost **then** Quay lại 4.1;

    4.5 **if** u trong M **then** //đã xét đường qua u;

        if g(u) ≥ giá trị g(u) được lưu trong M **then** Quay lại 4.1;

        else cập nhật giá trị g(u) trong M

    4.6 Khởi tạo danh sách L_1 rỗng;

    4.7 **for** mỗi trạng thái v kề u **do**

        {g(v) ← g(u) + k(u,v);

        f(v) ← g(v) + h(v);

        Đặt v vào danh sách L_1};

    4.8 **if** u không có trong M **then** đặt u vào M và lưu giá trị g(u);

    4.9 Sắp xếp L_1 theo thứ tự tăng của hàm f;

    4.10 Chuyển L_1 vào đầu danh sách L;

end;

Người ta chứng minh được rằng, thuật toán nhánh_và_cận cũng là thuật toán đầy đủ và tối ưu nếu hàm đánh giá h(u) là đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó.

II. Tìm đối tượng tốt nhất

Trong mục này chúng ta sẽ xét vấn đề tìm kiếm sau. Trên không gian tìm kiếm U được xác định hàm giá (hàm mục tiêu) cost, ứng với mỗi đối tượng x ∈ U với một giá trị số cost(x), số này được gọi là giá trị của x. Chúng ta cần tìm một đối tượng mà tại đó hàm giá trị lớn nhất, ta gọi đối tượng đó là đối tượng tốt nhất. Giả sử không gian tìm kiếm có cấu trúc cho phép ta xác định được khái niệm lân cận của mỗi đối tượng. Chẳng hạn, U là không gian trạng thái thì lân cận của trạng thái u gồm tất cả các trạng thái v

kề u; nếu U là không gian các vectơ thực n-chiều thì lân cận của vectơ x = (x_1, x_2, ... x_n) gồm tất cả các vectơ ở gần x theo khoảng cách Euclit thông thường.

Trong mục này, ta sẽ xét kỹ thuật tìm kiếm leo đồi để tìm đối tượng tốt nhất. Sau đó ta sẽ xét kỹ thuật tìm kiếm gradient (gradient search). Đó là kỹ thuật leo đồi áp dụng cho không gian tìm kiếm là không gian các vectơ thực n-chiều và hàm giá là là hàm khả vi liên tục. Cuối cùng ta sẽ nghiên cứu kỹ thuật tìm kiếm mô phỏng luyện kim(simulated annealing).

II.1 Tìm kiếm leo đồi

Kỹ thuật tìm kiếm leo đồi để tìm kiếm đối tượng tốt nhất hoàn toàn giống như kỹ thuật tìm kiếm leo đồi để tìm trạng thái kết thúc đã xét trong mục 4.3. Nghĩa là từ một đỉnh u ta chỉ leo lên đỉnh tốt nhất v (được xác định bởi hàm giá cost) trong lân cận u nếu đỉnh này "cao hơn" đỉnh u, tức là cost(v) > cost(u). Còn ở đây, từ một trạng thái ta "leo lên" trạng thái kề tốt nhất (được xác định bởi hàm giá), tiếp tục cho tới khi đạt tới trạng thái đích; nếu chưa đạt tới trạng thái đích mà không leo lên được nữa, thì ta tiếp tục "tụt xuống" trạng thái trước nó, rồi lại leo lên trạng thái tốt nhất còn lại. Quá trình tìm kiếm sẽ dừng lại ngay khi ta không leo lên đỉnh cao hơn được nữa.

Trong thủ tục leo đồi sau, biến u lưu đỉnh hiện thời, biến v lưu đỉnh tốt nhất (cost(v) nhỏ nhất) trong các đỉnh ở lân cận u. Khi thuật toán dừng, biến u sẽ lưu đối tượng tìm được.

procedure Hill_Climbing;

begin

1. u ← một đối tượng ban đầu nào đó;
2. **loop do**

2.1 v là trạng thái kề của u có giá trị cost lớn nhất trong các trạng thái kề

2.2 **if** cost(v) > cost(u) **then** u ← v

else return u;

end;

- Tối ưu địa phương và tối ưu toàn cục

Rõ ràng là, khi thuật toán leo đồi dừng lại tại đối tương u*, thì giá của nó cost(u*) lớn hơn giá của tất cả các đối tượng nằm trong lân cận của tất cả các đối tượng trên đường đi từ đối tượng ban đầu tới trạng thái u*. Do đó nghiệm u* mà thuật toán leo đồi tìm được là tối ưu địa phương. Cần nhấn mạnh rằng không có gì đảm bảo nghiệm đó là tối ưu toàn cục theo nghĩa là cost(u*) là lớn nhất trên toàn bộ không gian tìm kiếm.

Để nhận được nghiệm tốt hơn bằng thuật toán leo đồi, ta có thể áp dụng lặp lại nhiều lần thủ tục leo đồi xuất phát từ một dãy các đối tượng ban đầu được chọn ngẫu nhiên và lưu lại nghiệm tốt nhất qua mỗi lần lặp. Nếu số lần lặp đủ lớn thì ta có thể tìm được nghiệm tối ưu.

Kết quả của thuật toán leo đồi phụ thuộc rất nhiều vào hình dáng của “mặt cong” của hàm đánh giá. Nếu mặt cong chỉ có một số ít cực đại địa phương, thì kỹ thuật leo đồi sẽ tìm ra rất nhanh cực đại toàn cục.

Song có những vấn đề mà mặt cong của hàm giá tựa như lông nhím vậy, khi đó sử dụng kỹ thuật leo đồi đòi hỏi rất nhiều thời gian.

II.2 Tìm kiếm gradient

Tìm kiếm gradient là kỹ thuật tìm kiếm leo đồi để tìm giá trị lớn nhất (hoặc nhỏ nhất) của hàm khả vi liên tục f(x) trong không gian các vectơ thực n-chiều. Như ta đã biết, trong lân cận đủ nhỏ của điểm $x = (x_1,...,x_n)$, thì hàm f tăng nhanh nhất theo hướng của vectơ gradient:

Do đó tư tưởng của tìm kiếm gradient là từ một điểm ta đi tới điểm ở lân cận nó theo hướng của vectơ gradient.

$$\nabla f = \left(\frac{\partial f}{\partial x1}, \frac{\partial f}{\partial x2}, ..., \frac{\partial f}{\partial xn} \right)$$

procedure Gradient_Search;

begin

x ← điểm xuất phát nào đó;

repeat

x ← x + α∇f(x);

until |∇f| < ε;

end;

Trong thủ tục trên, α là hằng số dương nhỏ nhất xác định tỉ lệ của các bước, còn ε là hằng số dương nhỏ xác định tiêu chuẩn dừng. Bằng cách lấy các bước đủ nhỏ theo hướng của vectơ gradient chúng ta sẽ tìm được điểm cực đại địa phương, đó là điểm mà tại đó ∇f = 0, hoặc tìm được điểm rất gần với cực đại địa phương.

II.3 Tìm kiếm mô phỏng luyện kim

Như đã nhấn mạnh ở trên, tìm kiếm leo đồi không đảm bảo cho ta tìm được nghiệm tối ưu toàn cục. Để cho nghiệm tìm được gần với tối ưu toàn cục, ta áp dụng kỹ thuật leo đồi lặp xuất phát từ các điểm được lựa chọn ngẫu nhiên. Bây giờ thay cho việc luôn luôn "leo lên đồi" xuất phát từ các điểm khác nhau, ta thực hiện một số bước "tụt xuống" nhằm thoát ra khỏi các điểm cực đại địa phương. Đó chính là tư tưởng của kỹ thuật tìm kiếm mô phỏng luyện kim.

Trong tìm kiếm leo đồi, khi ở một trạng thái u ta luôn luôn đi tới trạng thái tốt nhất trong lân cận nó. Còn bây giờ, trong tìm kiếm mô phỏng luyện kim, ta chọn ngẫu nhiên một trạng thái v trong lân cận u. Nếu trạng thái v được chọn tốt hơn u (cost(v) > cost(u)) thì ta đi tới v, còn nếu không ta chỉ đi tới v với một xác suất nào đó. Xác suất này giảm theo hàm mũ của "độ xấu" của trạng thái v. Xác suất này còn phụ thuộc vào tham số nhiệt độ T. Nhiệt độ T càng cao thì bước đi tới trạng thái xấu càng có khả năng được thực hiện. Trong quá trình tìm kiếm, tham số nhiệt độ T giảm dần tới không. Khi T gần không, thuật toán hoạt động gần giống như leo đồi, hầu như nó không thực hiện bước tụt xuống. Cụ thể ta xác định xác suất đi tới trạng thái xấu v từ u là $e^{\Delta/T}$, ở đây Δ = cost(v) - cost(u).

Sau đây là thủ tục mô phỏng luyện kim.

```
procedure Simulated_Anneaning;
begin
    t ← 0;
    u ← trạng thái ban đầu nào đó;
    T ← nhiệt độ ban đầu;
    repeat
        v ← trạng thái được chọn nhẫu nhiên trong lân cận u;
        if cost(v) > cost(u) then u ← v
        else u ← v với xác suất e^(Δ/T);
        T ← g(T, t);
        t ← t + 1;
    until T đủ nhỏ
end;
```

Trong thủ tục trên, hàm g(T, t) thỏa mãn điều kiện g(T, t) < T với mọi t, nó xác định tốc độ giảm của nhiệt độ T. Người ta chứng minh được rằng, nếu nhiệt độ T giảm đủ chậm, thì thuật toán sẽ tìm được nghiệm tối ưu toàn cục. Thuật toán mô phỏng luyện kim đã được áp dụng thành công cho các bài toán tối ưu cỡ lớn.

III. Tìm kiếm mô phỏng sự tiến hóa. Thuật toán di truyền

Thuật toán di truyền (TTDT) là thuật toán bắt chước sự chọn lọc tự nhiên và di truyền. Trong tự nhiên, các cá thể khỏe, có khả năng thích nghi tốt với môi trường sẽ được tái sinh và nhân bản ở các thế hệ sau. Mỗi cá thể có cấu trúc gien đặc trưng cho phẩm chất của cá thể đó. Trong quá trình sinh sản, các cá thể con có thể thừa hưởng các phẩm chất của cả cha và mẹ, cấu trúc gien của nó mang một phần cấu trúc gien của cha và mẹ. Ngoài ra, trong quá trình tiến hóa, có thể xảy ra hiện tượng đột biến, cấu trúc gien của cá thể con có thể chứa các gien mà cả cha và mẹ đều không có.

Trong TTDT, mỗi cá thể được mã hóa bởi một cấu trúc dữ liệu mô tả cấu trúc gien của cá thể đó, ta sẽ gọi nó là nhiễm sắc thể (chromosome). Mỗi nhiễm sắc thể được tạo thành từ các đơn vị được gọi là gien. Chẳng hạn, trong các TTDT cổ điển, các nhiễm sắc thể là các chuỗi nhị phân, tức là mỗi cá thể được biểu diễn bởi một chuỗi nhị phân.

TTDT sẽ làm việc trên các quần thể gồm nhiều cá thể. Một quần thể ứng với một giai đoạn phát triển sẽ được gọi là một thế hệ. Từ thế hệ ban đầu được tạo ra, TTDT bắt chước chọn lọc tự nhiên và di truyền để biến đổi các thế hệ. TTDT sử dụng các toán tử cơ bản sau đây để biến đổi các thế hệ.

- Toán tử tái sinh (reproduction) (còn được gọi là toán tử chọn lọc (selection)). Các cá thể tốt được chọn lọc để đưa vào thế hệ sau. Sự lựa chọn này được thực hiện dựa vào độ thích nghi với môi trường của mỗi cá thể. Ta sẽ gọi hàm ứng mỗi cá thể với độ thích nghi của nó là hàm thích nghi (fitness function).
- Toán tử lai ghép (crossover). Hai cá thể cha và mẹ trao đổi các gien để tạo ra hai cá thể con.

- Toán tử đột biến (mutation). Một cá thể thay đổi một số gien để tạo thành cá thể mới.

Tất cả các toán tử trên khi thực hiện đều mang tính ngẫu nhiên. Cấu trúc cơ bản của TTDT là như sau:

procedure Genetic_Algorithm;

begin

t ← 0;

Khởi tạo thế hệ ban đầu P(t);

Đánh giá P(t) (theo hàm thích nghi);

repeat

t ← t + 1;

Sinh ra thế hệ mới P(t) từ P(t-1) bởi

- Chọn lọc
- Lai ghép
- Đột biến;

Đánh giá P(t);

until điều kiện kết thúc được thỏa mãn;

end;

Trong thủ tục trên, điều kiện kết thúc vòng lặp có thể là một số thế hệ đủ lớn nào đó, hoặc độ thích nghi của các cá thể tốt nhất trong các thế hệ kế tiếp nhau khác nhau không đáng kể. Khi thuật toán dừng, cá thể tốt nhất trong thế hệ cuối cùng được chọn làm nghiệm cần tìm.

Bây giờ ta sẽ xét chi tiết hơn toán tử chọn lọc và các toán tử di truyền (lai ghép, đột biến) trong các TTDT cổ điển.

1. Chọn lọc: Việc chọn lọc các cá thể từ một quần thể dựa trên độ thích nghi của mỗi cá thể. Các cá thể có độ thích nghi cao có nhiều khả năng được chọn. Cần nhấn mạnh rằng, hàm thích nghi chỉ cần là một hàm thực dương, nó có thể không tuyến tính, không liên tục, không khả vi. Quá trình chọn lọc được thực hiện theo kỹ thuật quay bánh xe.

Giả sử thế hệ hiện thời P(t) gồm có n cá thể $\{x_1,..,x_n\}$. Số n được gọi là cỡ của quần thể. Với mỗi cá thể x_i, ta tính độ thích nghi của nó $f(x_i)$. Tính tổng các độ thích nghi của tất cả các cá thể trong quần thể:

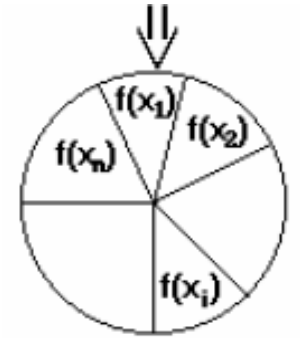
$$F = \sum_{i=1}^{n} f(xi)$$

Mỗi lần chọn lọc, ta thực hiện hai bước sau:

- Sinh ra một số thực ngẫu nhiên q trong khoảng (0, F);
- x_k là cá thể được chọn, nếu k là số nhỏ nhất sao cho

$$\sum_{i=1}^{k} f(xi) \geq 4$$

Việc chọn lọc theo hai bước trên có thể minh họa như sau: Ta có một bánh xe được chia thành n phần, mỗi phần ứng với độ thích nghi của một cá thể (hình 5.5). Một mũi tên chỉ vào bánh xe. Quay bánh xe, khi bánh xe dừng, mũi tên chỉ vào phần nào, cá thể ứng với phần đó được chọn.



Hình 5.5 Kỹ thuật quay bánh xe

Rõ ràng là với cách chọn này, các cá thể có thể có độ thích nghi càng cao càng có khả năng được chọn. Các cá thể có độ

thích nghi cao có thể có một hay nhiều bản sao, các cá thể có độ thích nghi thấp có thể không có mặt ở thế hệ sau (nó bị chết đi).

2. Lai ghép: Trên cá thể được chọn lọc, ta tiến hành toán tử lai ghép. Đầu tiên ta cần đưa ra xác suất lai ghép p_c. xác suất này cho ta hy vọng có p_c x n cá thể được lai ghép (n là cỡ của quần thể).

Với mỗi cá thể ta thực hiện hai bước sau:

• Sinh ra số thực ngẫu nhiên r trong đoạn [0, 1];

• Nếu $r < p_c$ thì cá thể đó được chọn để lai ghép

Từ các cá thể được chọn để lai ghép, người ta cặp đôi chúng một cách ngẫu nhiên. Trong trường hợp các nhiễm sắc thể là các chuỗi nhị phân có độ dài cố định m, ta có thể thực hiện lai ghép như sau:

Với mỗi cặp, sinh ra một số nguyên ngẫu nhiên p trên đoạn [1, m], p là vị trí điểm ghép. Cặp gồm hai nhiễm sắc thể

$a = (a_1 , ... , a_p , a_{p+1} , ... , a_m)$

$b = (b_1 , ... , b_p , b_{p+1} , ... , b_m)$

được thay bởi hai con là:

$a' = (a_1 , ... , a_p , b_{p+1} , ... , b_m)$

$b' = (b_1 , ... , b_p , a_{p+1} , ... , a_m)$

3. Đột biến: Ta thực hiện toán tử đột biến trên các cá thể có được sau quá trình lai ghép. Đột biến là thay đổi trạng thái một số gien nào đó trong nhiễm sắc thể. Mỗi gien chịu đột biến với xác suất p_m. Xác suất đột biến p_m do ta xác định và là xác suất thấp. Sau đây là toán tử đột biến trên các nhiễm sắc thể chuỗi nhị phân.

Với mỗi vị trí i trong nhiễm sắc thể:

$a = (a_1 , ... , a_i , ... , a_m)$

Ta sinh ra một số thực nghiệm ngẫu nhiên p_i trong [0,1]. Qua đột biến a được biến thành a' như sau:

$a' = (a'_1 , ... , a'_i , ... , a'_m)$

Trong đó :

$$a'_i = \begin{cases} a_i \text{ nếu } p_i \geq p_m \\ 1 - a_i \text{ nếu } p_i < p_m \end{cases}$$

Sau quá trình chọn lọc, lai ghép, đột biến, một thế hệ mới được sinh ra. Công việc còn lại của thuật toán di truyền bây giờ chỉ là lặp lại các bước trên.

Ví dụ: Xét bài toán tìm max của hàm $f(x) = x^2$ với x là số nguyên trên đoạn [0,31]. Để sử dụng TTDT, ta mã hoá mỗi số nguyên x trong đoạn [0,31] bởi một số nhị phân độ dài 5, chẳng hạn, chuỗi 11000 là mã của số nguyên 24. Hàm thích nghi được xác định là chính hàm $f(x) = x^2$. Quần thể ban đầu gồm 4 cá thể (cỡ của quần thể là n = 4). Thực hiện quá trình chọn lọc, ta nhận được kết quả trong bảng sau. Trong bảng này, ta thấy cá thể 2 có độ thích nghi cao nhất (576) nên nó được chọn 2 lần, cá thể 3 có độ thích nghi thấp nhất (64) không được chọn lần nào. Mỗi cá thể 1 và 4 được chọn 1 lần.

Bảng kết quả chọn lọc

Số liệu cá thể	Quần thể ban đầu	x	Độ thích nghi $f(x) = x^2$	Số lần được chọn
1	0 1 1 0 1	13	169	1
2	1 1 0 0 0	24	576	2
3	0 1 0 0 0	8	64	0
4	1 0 0 1 1	19	361	1

Thực hiện quá trình lai ghép với xác suất lai ghép $p_c = 1$, nên cá thể 1, 2, và 4 sau chọn lọc đều được lai ghép. Kết quả lai ghép được cho trong bảng sau. Trong bảng này, chuỗi thứ nhất được lai ghép với chuỗi thứ hai với điểm ghép là 4, chuỗi thứ hai và 4 được lai ghép với nhau với điểm ghép là 2.

Bảng kết quả lai ghép

Quần thể sau chọn lọc	Điểm ghép	Quần thể sau lai ghép	x	Độ thích nghi $f(x) = x^2$
0 1 1 0 | 1	4	0 1 1 0 0	12	144
1 1 0 0 | 0	4	1 1 0 0 1	25	625
1 1 | 0 0 0	2	1 1 0 1 1	27	729
1 0 | 0 1 1	2	1 0 0 0 0	16	256

Để thực hiện quá trình đột biến, ta chọn xác suất đột biến p_m= 0,001, tức là ta hy vọng có 5x4x0,001 = 0,02 bit được đột biến. Thực tế sẽ không có bit nào được đột biến. Như vậy thế hệ mới là quần thể sau lai ghép. Trong thế hệ ban đầu, độ thích nghi cao nhất là 576, độ thích nghi trung bình 292. Trong thế hệ sau, độ thích nghi cao nhất là 729, trung bình là 438. Chỉ qua một thế hệ, các cá thể đã "tốt lên" rất nhiều.

Thuật toán di truyền khác với các thuật toán tối ưu khác ở các điểm sau:

- TTDT chỉ sử dụng hàm thích nghi để hướng dẫn sự tìm kiếm, hàm thích nghi chỉ cần là hàm thực dương. Ngoài ra, nó không đòi hỏi không gian tìm kiếm phải có cấu trúc nào cả.
- TTDT làm việc trên các nhiễm sắc thể là mã của các cá thể cần tìm.
- TTDT tìm kiếm từ một quần thể gồm nhiều cá thể.
- Các toán tử trong TTDT đều mang tính ngẫu nhiên.

Để giải quyết một vấn đề bằng TTDT, chúng ta cần thực hiện các bước sau đây:

- Trước hết ta cần mã hóa các đối tượng cần tìm bởi một cấu trúc dữ liệu nào đó. Chẳng hạn, trong các TTDT cổ điển, như trong ví dụ trên, ta sử dụng mã nhị phân.
- Thiết kế hàm thích nghi. Trong các bài toán tối ưu, hàm thích nghi được xác định dựa vào hàm mục tiêu.
- Trên cơ sở cấu trúc của nhiễm sắc thể, thiết kế các toán tử di truyền (lai ghép, đột biến) cho phù hợp với các vấn đề cần giải quyết.
- Xác định cỡ của quần thể và khởi tạo quần thể ban đầu.
- Xác định xác suất lai ghép p_c và xác suất đột biến. Xác suất đột biến cần là xác suất thấp. Người ta (Goldberg, 1989) khuyên rằng nên chọn xác suất lai ghép là 0,6 và xác suất đột biến là 0,03. Tuy nhiên cần qua thử nghiệm để tìm ra các xác suất thích hợp cho vấn đề cần giải quyết.

Nói chung thuật ngữ TTDT là để chỉ TTDT cổ điển, khi mà cấu trúc của các nhiễm sắc thể là các chuỗi nhị phân với các toán tử di truyền đã được mô tả ở trên. Song trong nhiều vấn đề thực tế, thuận tiện hơn, ta có thể biểu diễn nhiễm sắc thể bởi các cấu trúc khác, chẳng hạn vectơ thực, mảng hai chiều, cây,...

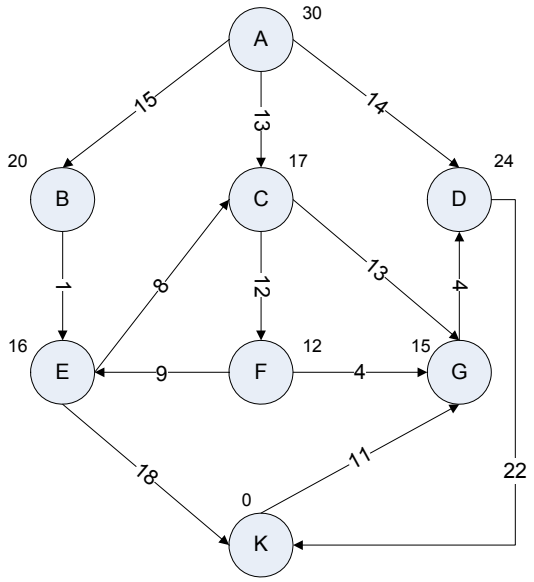
Tương ứng với cấu trúc của nhiễm sắc thể, có thể có nhiều cách xác định các toán tử di truyền. Quá trình sinh ra thế hệ mới P(t) từ thế hệ cũ P(t - 1) cũng có nhiều cách chọn lựa. Người ta gọi chung các thuật toán này là thuật toán tiến hóa (evolutionary algorithms) hoặc chương trình tiến hóa (evolution program).

IV. Bài tập

1. Chứng minh rằng nếu độ dài của các cung không nhỏ hơn một số dương δ nào đó thì thuật toán A* là thuật toán đầy đủ.
2. Viết chương trình minh họa tìm kiếm theo thuật toán A*, nhánh và cận, đối tượng tốt nhất, leo đồi của bài toán 8 số.
3. Viết chương trình giải bài toán 8 con hậu.
4. Ứng dụng giải thuật di truyền để tìm giá trị của các biến nguyên x, y, z sao cho hàm $f(x,y,z) = y\sin(z\cos(x)) - x\cos(z\sin(y))$ đạt giá trị lớn nhất.

Biết rằng *0 < x < 10, 0< y < 10, 0 <z < 10.*

5. Cho đồ thị không gian trạng thái:



Để tìm đường đi từ A tới K, hãy vẽ cây tìm kiếm của tìm kiếm A*, nhánh và cận và leo đồi.

Chương VI

TÌM KIẾM CÓ ĐỐI THỦ

Nghiên cứu máy tính chơi cờ đã xuất hiện rất sớm. Không lâu sau khi máy tính lập trình được ra đời vào năm 1950, Claude Shannon đã viết chương trình chơi cờ đầu tiên. các nhà nghiên cứu Trí Tuệ Nhân Tạo đã nghiên cứu việc chơi cờ, vì rằng máy tính chơi cờ là một bằng chứng rõ ràng về khả năng máy tính có thể làm được các công việc đòi hỏi trí thông minh của con người. Trong chương này chúng ta sẽ xét các vấn đề sau đây:

- Chơi cờ có thể xem như vấn đề tìm kiếm trong không gian trạng thái.
- Chiến lược tìm kiếm nước đi Minimax.
- Phương pháp cắt cụt α-β, một kỹ thuật để tăng hiệu quả của tìm kiếm Minimax.

I. Cây trò chơi và tìm kiếm trên cây trò chơi

Trong chương này chúng ta chỉ quan tâm nghiên cứu các trò chơi có hai người tham gia, chẳng hạn các loại cờ (cờ vua, cờ tướng, cờ ca rô...). Một người chơi được gọi là Trắng, đối thủ của anh ta được gọi là Đen. Mục tiêu của chúng ta là nghiên cứu chiến lược chọn nước đi cho Trắng (Máy tính cầm quân Trắng).

Chúng ta sẽ xét các trò chơi hai người với các đặc điểm sau. Hai người chơi thay phiên nhau đưa ra các nước đi tuân theo các luật đi nào đó, các luật này là như nhau cho cả hai người. Điển hình là cờ vua, trong cờ vua hai người chơi có thể áp dụng các luật đi con tốt, con xe, ... để đưa ra nước đi. Luật đi con tốt Trắng xe Trắng, ... cũng như luật đi con tốt Đen, xe Đen, ... Một đặc điểm nữa là hai người chơi đều được biết thông tin đầy đủ về các tình thế trong trò chơi (không như trong chơi bài, người chơi không thể biết các người chơi khác còn những con bài gì). Vấn đề chơi cờ có thể xem như vấn đề tìm kiếm nước đi, tại mỗi lần đến lượt mình, người chơi phải tìm trong số rất nhiều nước đi hợp lệ (tuân theo đúng luật đi), một nước đi tốt nhất sao cho qua một dãy nước đi đã thực hiện, anh ta giành phần thắng. Tuy nhiên vấn đề tìm kiếm ở đây sẽ phức tạp hơn vấn đề tìm kiếm mà chúng ta đã xét trong các chương trước, bởi vì ở đây có đối thủ, người chơi không biết được đối thủ của mình sẽ đi nước nào trong tương lai. Sau đây chúng ta sẽ phát biểu chính xác hơn vấn đề tìm kiếm này.

Vấn đề chơi cờ có thể xem như vấn đề tìm kiếm trong không gian trạng thái. Mỗi trạng thái là một tình thế (sự bố trí các quân của hai bên trên bàn cờ).

- Trạng thái ban đầu là sự sắp xếp các quân cờ của hai bên lúc bắt đầu cuộc chơi.
- Các toán tử là các nước đi hợp lệ.
- Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường được xác định bởi một số điều kiện dừng nào đó.
- Một hàm kết cuộc (payoff function) ứng mỗi trạng thái kết thúc với một giá trị nào đó. Chẳng hạn như cờ vua, mỗi trạng thái kết thúc chỉ có thể là thắng, hoặc thua

(đối với Trắng) hoặc hòa. Do đó, ta có thể xác định hàm kết cuộc là hàm nhận giá trị 1 tại các trạng thái kết thúc là thắng (đối với Trắng), -1 tại các trạng thái kết thúc là thua (đối với Trắng) và 0 tại các trạng thái kết thúc hòa. Trong một số trò chơi khác, chẳng hạn trò chơi tính điểm, hàm kết cuộc có thể nhận giá trị nguyên trong khoảng [-k, k] với k là một số nguyên dương nào đó.

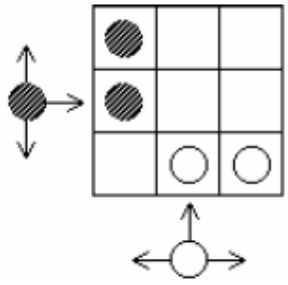
Như vậy vấn đề của Trắng là, tìm một dãy nước đi sao cho xen kẽ với các nước đi của Đen tạo thành một đường đi từ trạng thái ban đầu tới trạng thái kết thúc là thắng cho Trắng.

Để thuận lợi cho việc nghiên cứu các chiến lược chọn nước đi, ta biểu diễn không gian trạng thái trên dưới dạng cây trò chơi.

- Cây trò chơi

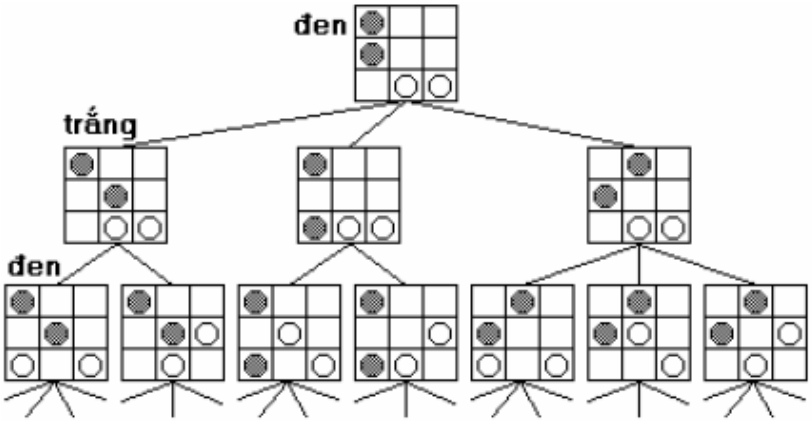
Cây trò chơi được xây dựng như sau. Gốc của cây ứng với trạng thái ban đầu. Ta sẽ gọi đỉnh ứng với trạng thái mà Trắng (Đen) đưa ra nước đi là đỉnh Trắng (Đen). Nếu một đỉnh là Trắng (Đen) ứng với trạng thái u, thì các đỉnh con của nó là tất cả các đỉnh biểu diễn trạng thái v, v nhận được từ u do Trắng (Đen) thực hiện nước đi hợp lệ nào đó. Do đó, trên cùng một mức của cây các đỉnh đều là Trắng hặc đều là Đen, các lá của cây ứng với các trạng thái kết thúc.

Ví dụ: Xét trò chơi Dodgen (được tạo ra bởi Colin Vout). Có hai quân Trắng và hai quân Đen, ban đầu được xếp vào bàn cờ 3*3 (hình 6.1). Quân Đen có thể đi tới ô trống ở bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi tới trống ở bên trái, bên phải, ở trên. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa hai quân của mình ra khỏi bàn cờ trước sẽ thắng, hoặc tạo ra tình thế bắt đối phương không đi được cũng sẽ thắng.



Hình 6.1 Trò chơi Dodgem

Giả sử Đen đi trước, ta có cây trò chơi được biểu diễn như trong hình 6.2.



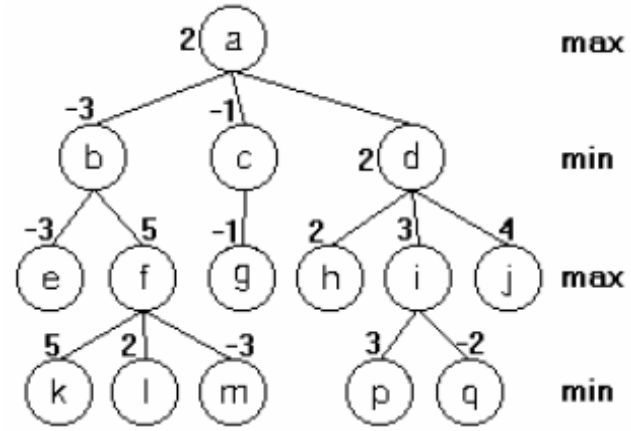
Hình 6.2 Cây trò chơi Dodgem với Đen đi trước

Quá trình chơi cờ là quá trình Trắng và Đen thay phiên nhau đưa ra quyết định, thực hiện một trong số các nước đi hợp lệ. Trên cây trò chơi, quá trình đó sẽ tạo ra đường đi từ gốc tới lá. Giả sử tới một thời điểm nào đó, đường đi đã dẫn tới đỉnh u. Nếu u là đỉnh Trắng (Đen) thì Trắng (Đen) cần chọn đi tới một trong các đỉnh Đen (Trắng) v là con của u. Tại đỉnh Đen (Trắng) v mà Trắng (Đen) vừa chọn, Đen (Trắng) sẽ phải chọn đi tới một trong các đỉnh Trắng (Đen) w là con của v. Quá trình trên sẽ dừng lại khi đạt tới một đỉnh là lá của cây.

Giả sử Trắng cần tìm nước đi tại đỉnh u. Nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con của v là đỉnh tốt nhất (cho Trắng) trong số các đỉnh con của u. Ta cần giả thiết rằng, đến lượt đối thủ chọn nước đi từ v, Đen cũng sẽ chọn nước đi tốt nhất cho anh ta. Như vậy, để chọn nước đi tối ưu cho Trắng tại đỉnh u, ta cần phải xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá (ứng với các trạng thái kết thúc) là giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất lên gốc u.

Giả sử u là đỉnh trong của cây và giá trị các đỉnh con của nó đã được xác định. Khi đó nếu u là đỉnh Trắng thì giá trị của nó được xác định là giá trị lớn nhất trong các giá trị của các đỉnh con. Còn nếu u là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các giá trị của các đỉnh con.

Ví dụ: Xét cây trò chơi trong hình 6.3, gốc a là đỉnh Trắng. Giá trị của các đỉnh là số ghi cạnh mỗi đỉnh. Đỉnh i là Trắng, nên giá trị của nó là max(3,-2) = 3, đỉnh d là đỉnh Đen, nên giá trị của nó là min(2, 3, 4) = 2.



Hình 6.3 Gán giá trị cho các đỉnh của cây trò chơi

Việc gán giá trị cho các đỉnh được thực hiện bởi các hàm đệ qui MaxVal và MinVal. Hàm MaxVal xác định giá trị cho các đỉnh Trắng, hàm MinVal xác định giá trị cho các đỉnh Đen.

function MaxVal(u);

begin

if u là đỉnh kết thúc **then** return f(u)

else return **max**{MinVal(v) | v là đỉnh con của u}

end;

function MinVal(u);

begin

if u là đỉnh kết thúc **then** return f(u)

else return min{MaxVal(v) | v là đỉnh con của u}

end;

Trong các hàm đệ quy trên, f(u) là giá trị của hàm kết cuộc tại đỉnh kết thúc u. Sau đây là thủ tục chọn nước đi cho Trắng tại đỉnh u. Trong thủ tục Minimax(u,v), v là biến lưu lại trạng thái mà Trắng đã chọn đi tới từ u.

procedure Minimax(u, v);

begin

val ← -∞;

for mỗi w là đỉnh con của u **do**

if val <= MinVal(w) **then**

{ val ← MinVal(w);

v ← w }

end;

Thủ tục chọn nước đi như trên gọi là chiến lược Minimax, bởi vì Trắng đã chọn được nước đi dẫn tới đỉnh con có giá trị là max của các giá trị các đỉnh con, và Đen đáp lại bằng nước đi tới đỉnh có giá trị là min của các giá trị các đỉnh con.

Thuật toán Minimax là thuật toán tìm kiếm theo độ sâu, ở đây ta đã cài đặt thuật toán Minimax bởi các hàm đệ quy. Bạn đọc hãy viết thủ tục không đệ quy thực hiện thuật toán này.

Về mặt lí thuyết, chiến lược Minimax cho phép ta tìm được nước đi tối ưu cho Trắng. Song nó không thực tế, chúng ta sẽ không có đủ thời gian để tính được nước đi tối ưu. Bởi vì thuật toán Minimax đòi hỏi ta phải xem xét toàn bộ các đỉnh của cây trò chơi. Trong các trò chơi hay, cây trò chơi là cực kỳ lớn. Chẳng hạn, đối với cờ vua, chỉ tính đến độ sâu 40, thì cây trò chơi đã có khoảng 10^{120} đỉnh! Nếu cây có độ cao m, và tại mỗi đỉnh có b nước đi thì độ phức tạp về thời gian của thuật toán Minimax là $O(b^m)$.

Để có thể tìm ra nhanh nước đi tốt (không phải là tối ưu) thay cho việc sử dụng hàm kết cuộc và xem xét tất cả các khả năng dẫn tới các trạng thái kết thúc, chúng ta sẽ sử dụng hàm đánh giá và chỉ xem xét một bộ phận của cây trò chơi.

- Hàm đánh giá

Hàm đánh giá eval ứng với mỗi trạng thái u của trò chơi với một giá trị số eval(u), giá trị này là sự đánh giá "độ lợi thế" của trạng thái u. Trạng thái u càng thuận lợi cho Trắng thì eval(u) là số dương càng lớn; u càng thuận lợi cho Đen thì eval(u) là số âm càng nhỏ; eval(u) ≈ 0 đối với trạng thái không lợi thế cho ai cả.

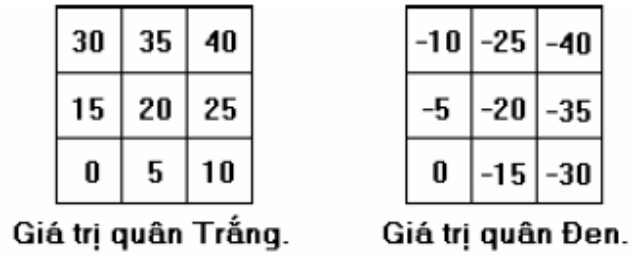
Chất lượng của chương trình chơi cờ phụ thuộc rất nhiều vào hàm đánh giá. Nếu hàm đánh giá cho ta sự đánh giá không chính xác về các trạng thái, nó có thể hướng dẫn ta đi tới trạng thái được xem là tốt, nhưng thực tế lại rất bất lợi cho ta. Thiết kế một hàm đánh giá tốt là một việc khó, đòi hỏi ta phải quan tâm đến nhiều nhân tố: các quân còn lại của hai bên, sự bố trí của các quân đó, ... ở đây có sự mâu thuẫn giữa độ chính xác của hàm đánh giá và thời gian tính của nó. Hàm đánh giá chính xác sẽ đòi hỏi rất nhiều thời gian tính toán, mà người chơi lại bị giới hạn bởi thời gian phải đưa ra nước đi.

Ví dụ 1: Sau đây ta đưa ra một cách xây dựng hàm đánh giá đơn giản cho cờ vua. Mỗi loại quân được gán một giá trị số phù hợp với "sức mạnh" của nó. Chẳng hạn, mỗi tốt Trắng (Đen) được cho 1 (-1), mã hoặc tượng Trắng (Đen) được cho 3 (-3), xe Trắng (Đen) được cho 5 (-5) và hoàng hậu Trắng (Đen) được cho 9 (-9). Lấy tổng giá trị của tất cả các quân trong một trạng thái, ta sẽ được giá trị đánh giá của trạng thái đó. Hàm đánh giá như thế được gọi là hàm tuyến tính có trọng số, vì nó có thể biểu diễn dưới dạng: $s_1w_1 + s_2w_2 + \ldots + s_nw_n$

Trong đó, w_i là giá trị mỗi loại quân, còn s_i là số quân loại đó. Trong cách đánh giá này, ta đã không tính đến sự bố trí của các quân, các mối tương quan giữa chúng.

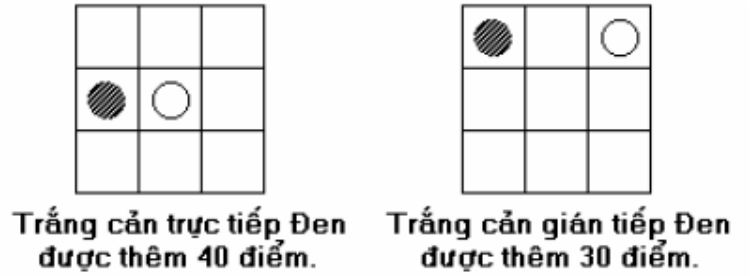
Ví dụ 2: Bây giờ ta đưa ra một cách đánh giá các trạng thái trong trò chơi Dodgem. Mỗi quân Trắng ở một vị trí trên bàn cờ được cho một giá trị tương ứng trong bảng bên trái hình 4.4. Còn mỗi quân Đen ở một vị trí sẽ được cho một giá trị tương ứng trong bảng bên phải hình 4.4:

Ngoài ra, nếu quân Trắng cản trực tiếp một quân Đen, nó được thêm 40 điểm, nếu cản gián tiếp nó được thêm 30 điểm (Xem hình 6.4). Tương tự, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, còn cản gián tiếp nó được thêm -30 điểm.

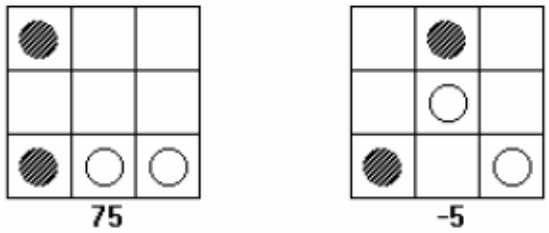


Hình 6.4 Đánh giá các quân trong trò chơi Dodgem

áp dụng các qui tắc trên, ta tính được giá trị của trạng thái ở bên trái hình 6.6 là 75, giá trị của trạng thái bên phải hình vẽ là -5.



Hình 6.5 Đánh giá sự tương quan giữa quân trắng và đen



Hình 6.6 Giá trị của một số trạng thái trong trò chơi Dodgem

Trong cánh đánh giá trên, ta đã xét đến vị trí của các quân và mối tương quan giữa các quân.

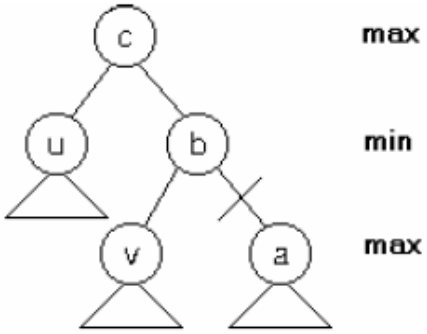
Một cách đơn giản để hạn chế không gian tìm kiếm là, khi cần xác định nước đi cho Trắng tại u, ta chỉ xem xét cây trò chơi gốc u tới độ cao h nào đó. áp dụng thủ tục Minimax cho cây trò chơi gốc u, độ cao h và sử dụng giá trị của hàm đánh giá cho các lá của cây đó, chúng ta sẽ tìm được nước đi tốt cho Trắng tại u.

II. Phương pháp cắt cụt alpha - beta

Trong chiến lược tìm kiếm Minimax, để tìm kiếm nước đi tốt cho Trắng tại trạng thái u, cho dù ta hạn chế không gian tìm kiếm trong phạm vi cây trò chơi gốc u với độ cao h, thì số đỉnh của cây trò chơi này cũng còn rất lớn với h ≥ 3. Chẳng hạn, trong cờ vua, nhân tố nhánh trong cây trò chơi trung bình khoảng 35, thời gian đòi hỏi phải đưa ra nước đi là 150 giây, với thời gian này trên máy tính thông thường chương trình của bạn chỉ có thể xem xét các đỉnh trong độ sâu 3 hoặc 4. Một người chơi cờ trình độ trung bình cũng có thể tính trước được 5, 6 nước hoặc hơn nữa, và do đó chương trình của bạn mới đạt trình độ người mới tập chơi!

Khi đánh giá đỉnh u tới độ sâu h, một thuật toán Minimax đòi hỏi ta phải đánh giá tất cả các đỉnh của cây gốc u tới độ sâu h. Song ta có thể giảm bớt số đỉnh cần phải đánh giá mà vẫn không ảnh hưởng gì đến sự đánh giá đỉnh u. Phương pháp cắt cụt alpha-beta cho phép ta cắt bỏ các nhánh không cần thiết cho sự đánh giá đỉnh u.

Tư tưởng của kỹ thuật cắt cụt alpha-beta là như sau: Nhớ lại rằng, chiến lược tìm kiếm Minimax là chiến lược tìm kiếm theo độ sâu. Giả sử trong quá trình tìm kiếm ta đi xuống đỉnh a là đỉnh Trắng, đỉnh a có người anh em v đã được đánh giá. Giả sử cha của đỉnh a là b và b có người anh em u dã được đánh giá, và giả sử cha của b là c (Xem hình 6.7). Khi đó ta có giá trị đỉnh c (đỉnh Trắng) ít nhất là giá trị của u, giá trị của đỉnh b (đỉnh Đen) nhiều nhất là giá trị v. Do đó, nếu eval(u) ≥ eval(v), ta không cần đi xuống để đánh giá đỉnh a nữa mà vẫn không ảnh hưởng gì dến đánh giá đỉnh c. Hay nói cách khác ta có thể cắt bỏ cây con gốc a. Lập luận tương tự cho trường hợp a là đỉnh Đen, trong trường hợp này nếu eval(u) ≤ eval(v) ta cũng có thể cắt bỏ cây con gốc a.



Hình 6.7 Cắt bỏ cây con gốc a, nếu eval(u) ≥ eval(v)

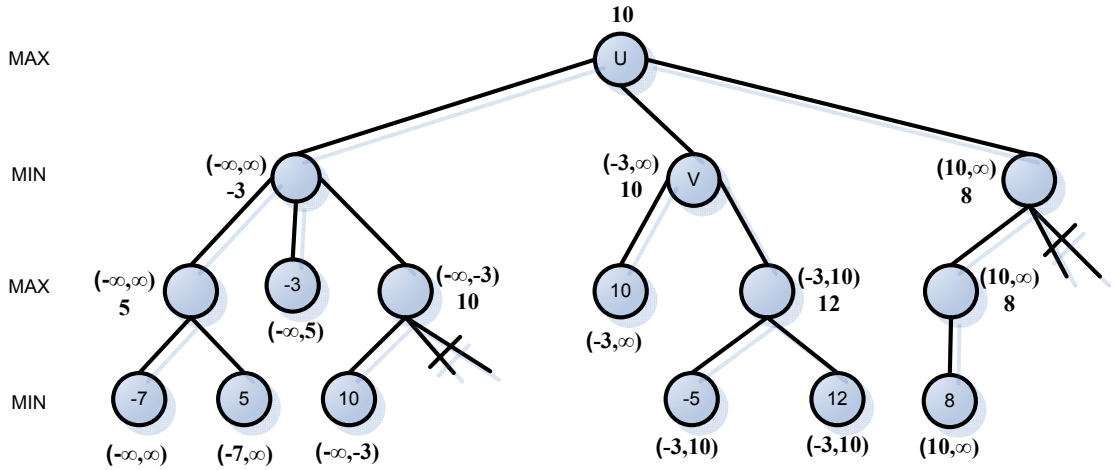
Để cài đặt kỹ thuật cắt cụt alpha-beta, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Trắng, còn tham số β ghi lại giá trị nhỏ nhất trong các đỉnh con đã đánh giá của một đỉnh Đen. Giá trị của α và β sẽ được cập nhật trong quá trình tìm kiếm. α và β được sử dụng như các biến địa phương trong các hàm MaxVal(u, α, β) (hàm xác định giá trị của đỉnh Trắng u) và Minval(u, α, β) (hàm xác định giá trị của đỉnh Đen u).

Thuật toán tìm nước đi cho Trắng sử dụng kỹ thuật cắt cụt alpha-beta, được cài đặt bởi thủ tục Alpha_beta(u,v), trong đó v là tham biến ghi lại đỉnh mà Trắng cần đi tới từ u.

```
function MaxVal(u α, β);
begin
    if u là lá của cây hạn chế hoặc u là đỉnh kết thúc then return eval(u)
    else {
        for mỗi đỉnh v là con của u do
            {α ← max[α, MinVal(v, α, β)];
            // Cắt bỏ các cây con từ các đỉnh v còn lại
            if α ≥ β then return β; }
        return α;
        }
end;
function MinVal(u, α, β);
begin
    if u là lá của cây hạn chế hoặc u là đỉnh kết thúc then return eval(u)
    else {
        for mỗi đỉnh v là con của u do
            {β ← min[β, MaxVal(v, α, β)];
            // Cắt bỏ các cây con từ các đỉnh v còn lại
            if α ≥ β then return α; }
        return β;
        }
end;
procedure Alpha_beta(u, v);
α ← -∞;
β ← ∞;
begin
    for mỗi đỉnh w là con của u do
        if α ≤ MinVal(w, α, β) then
            { α ← MinVal(w, α, β);
            v ← w;}
end;
```

Ví dụ. Xét cây trò chơi gốc u (đỉnh Trắng) giới hạn bởi độ cao h = 3 (hình 4.8). Số ghi cạnh các lá là giá trị của hàm đánh giá. Áp dụng chiến lược Minimax và kỹ thuật cắt

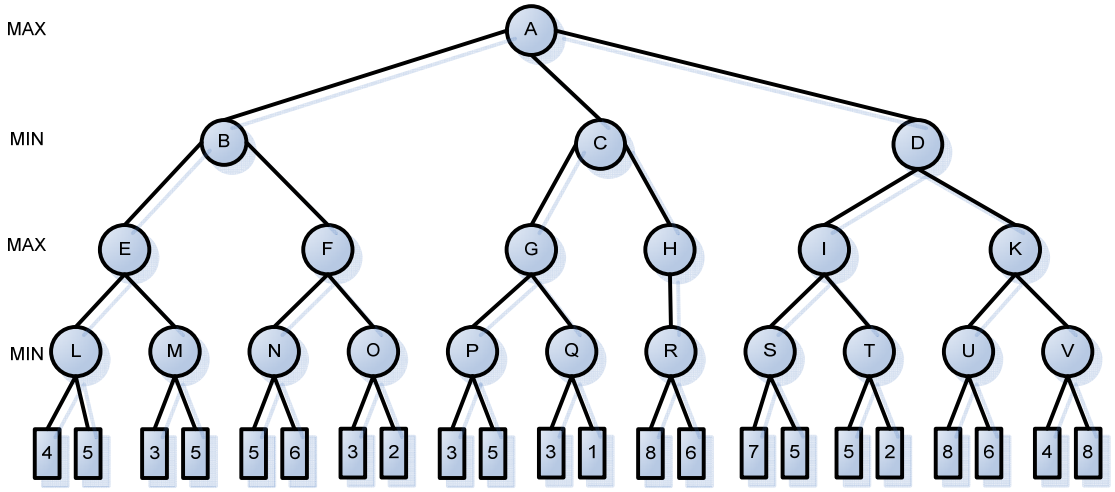
cụt, ta xác định được nước đi tốt nhất cho Trắng tại u, đó là nước đi dẫn tới đỉnh v có giá trị 10. Cạnh mỗi đỉnh ta cũng cho giá trị của cặp tham số (α, β). Khi gọi các hàm MaxVal và MinVal để xác định giá trị của đỉnh đó. Các nhánh bị cắt bỏ được chỉ ra trong hình 6.8.



Hình 6.8 Xác định giá trị các đỉnh bằng kỹ thuật cắt cụt

III. Bài tập

1. Viết chương trình trò chơi Dodgem có áp dụng phương pháp cắt cụt alpha-beta.
2. Dùng kỹ thuật cắt cụt alpha-beta để định trị cho nút gốc của cây trò chơi (các nút lá đã được gán trị):



3. Xét một trò chơi có 6 viên bi, hai người thay phiên nhau nhặt từ 1 đến 3 viên. Người phải nhặt viên bi cuối cùng thì bị thua.
   a. Vẽ toàn bộ cây trò chơi
   b. Sử dụng kỹ thuật cắt tỉa alpha-beta định trị cho nút gốc
   c. Ai sẽ thắng trong trò chơi này nếu hai người đều đi những nước tốt nhất. Hãy cho một trường hợp tổng quát khi ban đầu có n viên bi và mỗi lần có thể nhặt từ 1 đến m viên.

4. Xét một trò chơi có 7 cái đĩa. Người chơi 1 chia thành 2 chồng có số đĩa không bằng nhau. Người chơi 2 chọn một chồng trong số các chồng con để chia và tiếp tục chia thành 2 chồng không bằng nhau. Hai người luân phiên nhau chia đĩa như vậy cho đến khi không thể chia được nữa thì thua
   a. Vẽ toàn bộ cây trò chơi
   b. Sử dụng kỹ thuật cắt tỉa alpha-beta định trị cho nút gốc
   c. Ai sẽ thắng trong trò chơi này nếu hai người đều đi những nước tốt nhất.

Phần III

HỌC MÁY (MACHINE LEARNING)

Nội dung chính: Trong phần này, chúng ta sẽ tìm hiểu về một nhánh nghiên cứu hiện đại của Trí Tuệ Nhân Tạo, đó là học máy bao gồm giải thuật ID3, mạng neuron, và giải thuật di truyền.

Mục tiêu cần đạt : Sau chương này, sinh viên có thể :

- Hiểu được mục tiêu của lĩnh vực 'máy hoc'.
- Biết 3 tiếp cận học của lĩnh vực học máy.
- Vận dụng giải thuật ID3 vào các bài toán thực tế
- Hiểu khái niệm mạng neuron và các vấn đề có liên quan
- Hiểu giải thuật di truyền và ứng dụng của nó vào các bài toán thực tế.

Kiến thức tiên quyết: Biểu diễn tri thức ở dạng luật, tìm kiếm trong không gian trạng thái, khái niệm Entropy trong Lý thuyết thông tin.

GIỚI THIỆU

Khi được hỏi về những kỹ năng thông minh nào là cơ bản nhất đồng thời khó tự động hóa nhất của con người ngoài các hoạt động sáng tạo nghệ thuật, hành động ra quyết định mang tính đạo đức, trách nhiệm xã hội thì người ta thường đề cập đến vấn đề ngôn ngữ và học. Trãi qua nhiều năm, hai lĩnh vực này vẫn là mục tiêu, thách thức của khoa học TTNT.

Tầm quan trọng của việc học thì không cần phải tranh cãi, vì khả năng học chính là một trong những thành tố quan trọng của hành vi thông minh. Mặc dù tiếp cận hệ chuyên gia đã phát triển được nhiều năm, song số lượng các hệ chuyên vẫn còn hạn chế. Một trong những nguyên nhân chủ yếu là do quá trình tích lũy tri thức phức tạp, chi phí phát triển các hệ chuyên gia rất cao, nhưng chúng không có khả năng học, khả năng tự thích nghi khi môi trường thay đổi. Các chiến lược giải quyết vấn đề của chúng cứng nhắc và khi có nhu cầu thay đổi, thì việc sửa đổi một lượng lớn mã chương trình là rất khó khăn. Một giải pháp hiển nhiên là các chương trình tự học lấy cách giải quyết vấn đề từ kinh nghiệm, từ sự giống nhau, từ các ví dụ hay từ những 'chỉ dẫn', 'lời khuyên',...

Mặc dù học vẫn còn là một vấn đề khó, nhưng sự thành công của một số chương trình học máy thuyết phục rằng có thể tồn tại một tập hợp các nguyên tắc học tổng quát cho phép xây dựng nên các chương trình có khả năng học trong nhiều lĩnh vực thực tế.

Chương này sẽ giới thiệu sơ lược về lĩnh vực nghiên cứu này, đồng thời đi vào chi tiết một số giải thuật học quan trọng.

Định nghĩa 'học'

Theo Herbert Simon: 'Học được định nghĩa như là bất cứ sự thay đổi nào trong một hệ thống cho phép nó tiến hành tốt hơn trong lần thứ hai khi lặp lại cùng một nhiệm vụ hoặc với một nhiệm vụ khác rút ra từ cùng một quần thể các nhiệm vụ đó'

Định nghĩa này mặc dù ngắn nhưng đưa ra nhiều vấn đề liên quan đến việc phát triển một chương trình có khả năng học. Học liên quan đến việc khái quát hóa từ kinh nghiệm: hiệu quả thực hiện của chương trình không chỉ cải thiện với 'việc lặp lại cùng một nhiệm vụ' mà còn với các nhiệm vụ tương tự. Vì những lĩnh vực đáng chú ý thường có khuynh hướng là to lớn, nên các chương trình học – CTH (learner) chỉ có thể khảo sát một phần nhỏ trong toàn bộ các ví dụ có thể; từ kinh nghiệm hạn chế này, CTH vẫn phải khái quát hóa được một cách đúng đắn những ví dụ chưa từng gặp trong lĩnh vực đó. Đây chính là bài toán quy nạp (induction), và nó chính là trung tâm của việc học. Trong hầu hết các bài toán học, dữ liệu luyện tập sẵn có thường không đủ để đảm bảo đưa ra được một khái quát hóa tối ưu, cho dù CTH sử dụng giải thuật nào. Vì vậy, các giải thuật học phải khái quát hóa theo phương pháp heuristic, nghĩa là chúng sẽ chọn một số khía cạnh nào đó mà theo kinh nghiệm là cho hiệu quả trong tương lai để khái quát. Các tiêu chuẩn lựa chọn này gọi là thiên lệch quy nạp (inductive bias).

Có nhiều nhiệm vụ học (learning task) khác nhau. Ở đây chỉ trình bày nhiệm vụ học quy nạp (inductive learning), đây là một trong những nhiệm vụ học cơ bản. Nhiệm vụ của CTH là học một khái quát (generalization) từ một tập hợp các ví dụ. Học khái niệm (concept learning) là một bài toán học quy nạp tiêu biểu: cho trước một số ví dụ của khái niệm, chúng ta phải suy ra một định nghĩa cho phép người dùng nhận biết một cách đúng đắn những thể hiện của khái niệm đó trong tương lai.

Các tiếp cận học

Có ba tiếp cận học: tiếp cận ký hiệu (symbol-based learning), tiếp cận mạng neuron hay kết nối (neural or connectionist networks) và tiếp cận nổi trội (emergent) hay di truyền và tiến hóa (genetic and evolutionary learning).

Các CTH thuộc tiếp cận dựa trên ký hiệu biểu diễn vấn đề dưới dạng các ký hiệu (symbol), các giải thuật học sẽ tìm cách suy ra các khái quát mới, hợp lệ, hữu dụng và được biểu diễn bằng các ký hiệu này. Có nhiều giải thuật được đưa ra theo tiếp cận học này, tuy nhiên chương VII chỉ trình bày một giải thuật được sử dụng rộng rãi trong số đó, đó là giải thuật quy nạp cây quyết định ID3.

Ngược lại với tiếp cận ký hiệu, tiếp cận kết nối không học bằng cách tích lũy các câu trong một ngôn ngữ ký hiệu. Giống như bộ não động vật chứa một số lượng lớn các tế bào thần kinh liên hệ với nhau, mạng neuron là những hệ thống gồm các neuron nhân tạo liên hệ với nhau. Tri thức của chương trình là ngầm định trong tổ chức và tương tác của các neuron này. chương VIII sẽ đi vào chi tiết của tiếp cận này.

Tiếp cận thứ ba là tiếp cận nổi trội mô phỏng cách thức các hệ sinh học tiến hóa trong tự nhiên, nên còn được gọi là tiếp cận di truyền và tiến hóa. chương IX sẽ đi vào chi tiết của tiếp cận này.

Chương VII

TIẾP CẬN KÝ HIỆU:
GIẢI THUẬT QUY NẠP CÂY QUYẾT ĐỊNH ID3

I. Giới thiệu

Giải thuật quy nạp cây ID3 (gọi tắt là ID3) là một giải thuật học đơn giản nhưng tỏ ra thành công trong nhiều lĩnh vực. ID3 là một giải thuật hay vì cách biểu diễn tri thức học được của nó, tiếp cận của nó trong việc quản lý tính phức tạp, heuristic của nó dùng cho việc chọn lựa các khái niệm ứng viên, và tiềm năng của nó đối với việc xử lý dữ liệu nhiễu.

ID3 biểu diễn các khái niệm (concept) ở dạng các cây quyết định (decision tree). Biểu diễn này cho phép chúng ta xác định phân loại của một đối tượng bằng cách kiểm tra các giá trị của nó trên một số thuộc tính nào đó.

Như vậy, nhiệm vụ của giải thuật ID3 là học cây quyết định từ một tập các ví dụ rèn luyện (training example) hay còn gọi là dữ liệu rèn luyện (training data). Hay nói khác hơn, giải thuật có:

- **Đầu vào:** Một tập hợp các ví dụ. Mỗi ví dụ bao gồm các thuộc tính mô tả một tình huống, hay một đối tượng nào đó, và một giá trị phân loại của nó.
- **Đầu ra:** Cây quyết định có khả năng phân loại đúng đắn các ví dụ trong tập dữ liệu rèn luyện, và hy vọng là phân loại đúng cho cả các ví dụ chưa gặp trong tương lai.

Ví dụ, chúng ta hãy xét bài toán phân loại xem ta 'có đi chơi tennis' ứng với thời tiết nào đó không. Giải thuật ID3 sẽ học cây quyết định từ tập hợp các ví dụ sau:

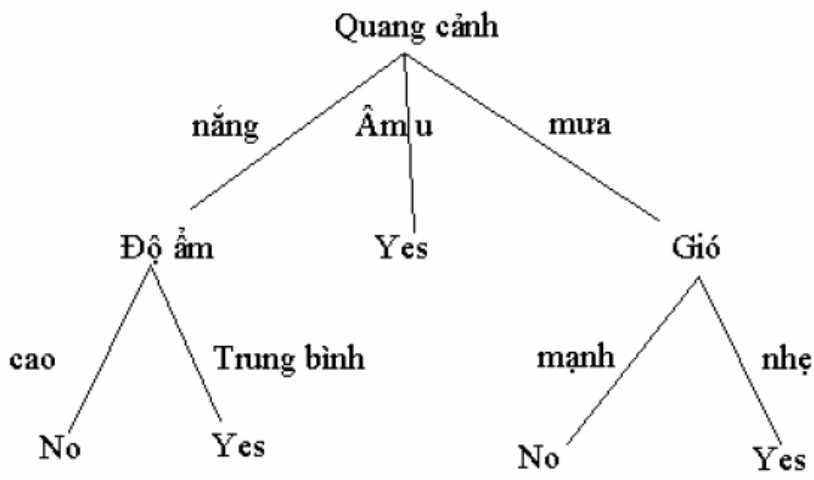
Ngày	Quang cảnh	Nhiệt độ	Độ ẩm	Gió	Chơi Tennis
D1	Nắng	Nóng	Cao	nhẹ	Không
D2	Nắng	Nóng	Cao	Mạnh	Không
D3	Âm u	Nóng	Cao	Nhẹ	Có
D4	Mưa	ấm áp	Cao	nhẹ	Có
D5	Mưa	Mát	TB	nhẹ	Có
D6	Mưa	Mát	TB	Mạnh	Không
D7	Âm u	Mát	TB	Mạnh	Có
D8	Nắng	ấm áp	Cao	nhẹ	Không
D9	Nắng	Mát	TB	nhẹ	Có
D10	Mưa	ấm áp	TB	nhẹ	Có
D11	Nắng	ấm áp	TB	Mạnh	Có
D12	Âm u	ấm áp	Cao	Mạnh	Có
D13	Âm u	Nóng	TB	nhẹ	Có
D14	Mưa	ấm áp	Cao	Mạnh	không

Bảng 7.1 - Tập dữ liệu rèn luyện cho khái niệm 'Có đi chơi tennis không?'

Tập dữ liệu này bao gồm 14 ví dụ. Mỗi ví dụ biểu diễn cho tình trạng thời tiết gồm các thuộc tính quang cảnh, nhiệt độ, độ ẩm và gió; và đều có một thuộc tính phân loại 'chơi Tennis' (có, không). 'Không' nghĩa là không đi chơi tennis ứng với thời tiết đó, 'Có' nghĩa là ngược lại. Giá trị phân loại ở đây chỉ có hai loại (có, không), hay còn ta nói phân loại của tập ví dụ của khái niệm này thành hai lớp (classes). Thuộc tính 'Chơi tennis' còn được gọi là thuộc tính đích (target attribute).

Mỗi thuộc tính đều có một tập các giá trị hữu hạn. Thuộc tính quang cảnh có ba giá trị (âm u, mưa, nắng), nhiệt độ có ba giá trị (nóng, mát, ấm áp), độ ẩm có hai giá trị (cao, TB) và gió có hai giá trị (mạnh, nhẹ). Các giá trị này chính là ký hiệu (symbol) dùng để biểu diễn bài toán.

Từ tập dữ liệu rèn luyện này, giải thuật ID3 sẽ học một cây quyết định có khả năng phân loại đúng đắn các ví dụ trong tập này, đồng thời hy vọng trong tương lai, nó cũng sẽ phân loại đúng các ví dụ không nằm trong tập này. Một cây quyết định ví dụ mà giải thuật ID3 có thể quy nạp được là:



Hình 7.1 - Cây quyết định cho khái niệm 'Có đi chơi tennis không?'

Các nút trong cây quyết định biểu diễn cho một sự kiểm tra trên một thuộc tính nào đó, mỗi giá trị có thể có của thuộc tính đó tương ứng với một nhánh của cây. Các nút lá thể hiện sự phân loại của các ví dụ thuộc nhánh đó, hay chính là giá trị của thuộc tính phân loại.

Sau khi giải thuật đã quy nạp được cây quyết định, thì cây này sẽ được sử dụng để phân loại tất cả các ví dụ hay thể hiện (instance) trong tương lai. Và cây quyết định sẽ không thay đổi cho đến khi ta cho thực hiện lại giải thuật ID3 trên một tập dữ liệu rèn luyện khác.

Ứng với một tập dữ liệu rèn luyện sẽ có nhiều cây quyết định có thể phân loại đúng tất cả các ví dụ trong tập dữ liệu rèn luyện. Kích cỡ của các cây quyết định khác nhau tùy thuộc vào thứ tự của các kiểm tra trên thuộc tính.

Vậy làm sao để học được cây quyết định có thể phân loại đúng tất cả các ví dụ trong tập rèn luyện? Một cách tiếp cận đơn giản là học thuộc lòng tất cả các ví dụ bằng cách xây dựng một cây mà có một lá cho mỗi ví dụ. Với cách tiếp cận này thì có thể cây quyết định sẽ không phân loại đúng cho các ví dụ chưa gặp trong tương lai. Vì phương pháp này cũng giống như hình thức 'học vẹt', mà cây không hề học được một khái quát nào của khái niệm cần học. Vậy, ta nên học một cây quyết định như thế nào là tốt?

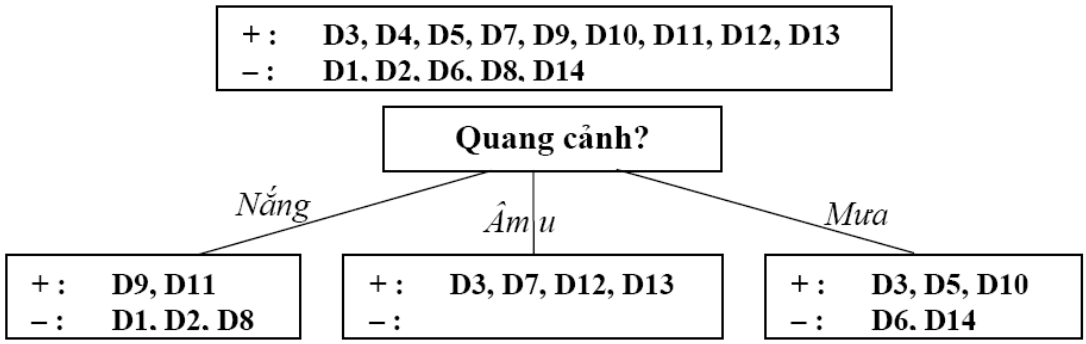
Occam's razor và một số lập luận khác đều cho rằng 'giả thuyết có khả năng nhất là giả thuyết đơn giản nhất thống nhất với tất cả các quan sát', ta nên luôn luôn chấp nhận những câu trả lời đơn giản nhất đáp ứng một cách đúng đắn dữ liệu của chúng ta. Trong trường hợp này là các giải thuật học cố gắng tạo ra cây quyết định nhỏ nhất phân loại một cách đúng đắn tất cả các ví dụ đã cho. Trong phần kế tiếp, chúng ta sẽ đi vào giải thuật ID3, là một giải thuật quy nạp cây quyết định đơn giản thỏa mãn các vấn đề vừa nêu.

II. Giải thuật ID3 xây dựng cây quyết định từ trên–xuống

ID3 xây dựng cây quyết định (cây QĐ) theo cách từ trên xuống. Lưu ý rằng đối với bất kỳ thuộc tính nào, chúng ta cũng có thể phân vùng tập hợp các ví dụ rèn luyện thành những tập con tách rời, mà ở đó mọi ví dụ trong một phân vùng (partition) có một giá trị chung cho thuộc tính đó. ID3 chọn một thuộc tính để kiểm tra tại nút hiện tại của cây và dùng trắc nghiệm này để phân vùng tập hợp các ví dụ; thuật toán khi đó xây dựng theo cách đệ quy một cây con cho từng phân vùng. Việc này tiếp tục cho đến khi mọi thành viên của phân vùng đều nằm trong cùng một lớp; lớp đó trở thành nút lá của cây.

Vì thứ tự của các trắc nghiệm là rất quan trọng đối với việc xây dựng một cây QĐ đơn giản, ID3 phụ thuộc rất nhiều vào tiêu chuẩn chọn lựa trắc nghiệm để làm gốc của cây. Để đơn giản, phần này chỉ mô tả giải thuật dùng để xây dựng cây QĐ, với việc giả định một hàm chọn trắc nghiệm thích hợp. Phần kế tiếp sẽ trình bày heuristic chọn lựa của ID3.

Ví dụ, hãy xem xét cách xây dựng cây QĐ của ID3 trong hình sau từ tập ví dụ rèn luyện trong bảng 7.1.



Hình 7.2 - Một phần cây quyết định xây dựng được

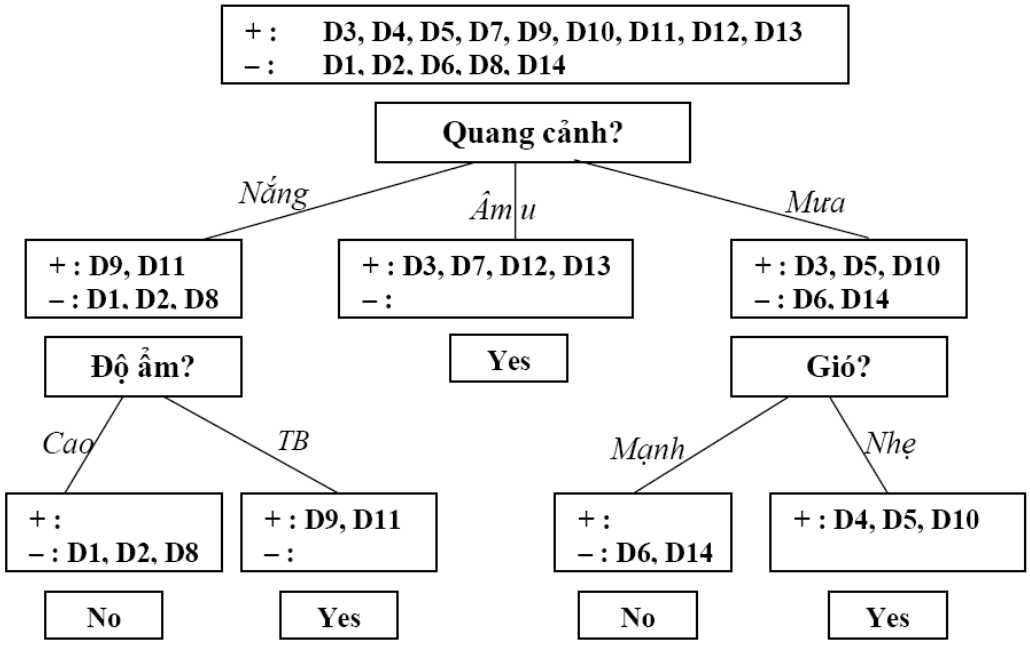
Bắt đầu với bảng đầy đủ gồm 14 ví dụ rèn luyện, ID3 chọn thuộc tính quang cảnh để làm thuộc tính gốc sử dụng hàm chọn lựa thuộc tính mô tả trong phần kế tiếp. Trắc nghiệm này phân chia tập ví dụ như cho thấy trong hình 7.2 với phần tử của mỗi phân vùng được liệt kê bởi số thứ tự của chúng trong bảng.

* ID3 xây dựng cây quyết định theo giải thuật sau:

Function induce_tree(tập_ví_dụ, tập_thuộc_tính)
begin
    if mọi ví dụ trong tập_ví_dụ đều nằm trong cùng một lớp **then**
        return một nút lá được gán nhãn bởi lớp đó
    else if tập_thuộc_tính là rỗng **then**
        return nút lá được gán nhãn bởi tuyển của tất cả các lớp trong tập_ví_dụ
    else
      begin
      chọn một thuộc tính P, lấy nó làm gốc cho cây hiện tại;

xóa P ra khỏi tập_thuộc_tính;
với mỗi giá trị V của P
begin
tạo một nhánh của cây gán nhãn V;
Đặt vào phân_vùng$_V$ các ví dụ trong tập_ví_dụ có giá trị V tại thuộc tính P;
Gọi induce_tree(phân_vùng$_V$, tập_thuộc_tính), gắn kết quả vào nhánh V
end
end
end

ID3 áp dụng hàm induce_tree một cách đệ quy cho từng phân vùng. Ví dụ, phân vùng của nhánh "Âm u" có các ví dụ toàn dương, hay thuộc lớp 'Có', nên ID3 tạo một nút lá với nhãn là lớp 'Có'. Còn phân vùng của hai nhánh còn lại vừa có ví dụ âm, vừa có ví dụ dương. Nên tiếp tục chọn thuộc tính "Độ ẩm" để làm trắc nghiệm cho nhánh Nắng, và thuộc tính Gió cho nhánh Mưa, vì các ví dụ trong các phân vùng con của các nhánh cây này đều thuộc cùng một lớp, nên giải thuật ID3 kết thúc và ta có được cây QĐ như hình 7.3.



Hình 7.3 - Cây quyết định đã xây dựng xong.

Lưu ý, để phân loại một ví dụ, có khi cây QĐ không cần sử dụng tất cả các thuộc tính đã cho, mặc dù nó vẫn phân loại đúng tất cả các ví dụ.

* Các khả năng có thể có của các phân vùng (partition):

Trong quá trình xây dựng cây QĐ, phân vùng của một nhánh mới có thể có các dạng sau:

1. Có các ví dụ thuộc các lớp khác nhau, chẳng hạn như có cả ví dụ âm và dương như phân vùng "Quang cảnh = Nắng" của ví dụ trên => giải thuật phải tiếp tục tách một lần nữa.

2. Tất cả các ví dụ đều thuộc cùng một lớp, chẳng hạn như toàn âm hoặc toàn dương như phân vùng "Quang cảnh = Âm u" của ví dụ trên => giải thuật trả về nút lá với nhãn là lớp đó.

3. Không còn ví dụ nào => giải thuật trả về mặc nhiên

4. Không còn thuộc tính nào => nghĩa là dữ liệu bị nhiễu, khi đó giải thuật phải sử dụng một luật nào đó để xử lý, chẳng hạn như luật đa số (lớp nào có nhiều ví dụ hơn sẽ được dùng để gán nhãn cho nút lá trả về).

Từ các nhận xét này, ta thấy rằng để có một cây QĐ đơn giản, hay một cây có chiều cao là thấp, ta nên chọn một thuộc tính sao cho tạo ra càng nhiều các phân vùng chỉ chứa các ví dụ thuộc cùng một lớp càng tốt. Một phân vùng chỉ có ví dụ thuộc cùng một lớp, ta nói phân vùng đó có tính thuần nhất. Vậy, để chọn thuộc tính kiểm tra có thể giảm thiểu chiều sâu của cây QĐ, ta cần một phép đo để đo tính thuần nhất của các phân vùng, và chọn thuộc tính kiểm tra tạo ra càng nhiều phân vùng thuần nhất càng tốt. ID3 sử dụng lý thuyết thông tin để thực hiện điều này.

III. Thuộc tính nào là thuộc tính dùng để phân loại tốt nhất?

Quinlan (1983) là người đầu tiên đề xuất việc sử dụng lý thuyết thông tin để tạo ra các cây quyết định và công trình của ông là cơ sở cho phần trình bày ở đây. Lý thuyết thông tin của Shannon (1948) cung cấp khái niệm entropy để đo tính thuần nhất (hay ngược lại là độ pha trộn) của một tập hợp. Một tập hợp là thuần nhất nếu như tất cả các phần tử của tập hợp đều thuộc cùng một loại, và khi đó ta nói tập hợp này có độ pha trộn là thấp nhất. Trong trường hợp của tập ví dụ, thì tập ví dụ là thuần nhất nếu như tất cả các ví dụ đều có cùng giá trị phân loại.

Khi tập ví dụ là thuần nhất thì có thể nói: ta biết chắc chắn về giá trị phân loại của một ví dụ thuộc tập này, hay ta có lượng thông tin về tập đó là cao nhất. Khi tập ví dụ có độ pha trộn cao nhất, nghĩa là số lượng các ví dụ có cùng giá trị phân loại cho mỗi loại là tương đương nhau, thì khi đó ta không thể đoán chính xác được một ví dụ có thể có giá trị phân loại gì, hay nói khác hơn, lượng thông tin ta có được về tập này là ít nhất. Vậy, điều ta mong muốn ở đây là làm sao chọn thuộc tính để hỏi sao cho có thể chia tập ví dụ ban đầu thành các tập ví dụ thuần nhất càng nhanh càng tốt. Vậy trước hết, ta cần có một phép đo để đo độ thuần nhất của một tập hợp, từ đó mới có thể so sánh tập ví dụ nào thì tốt hơn. Phần kế tiếp sẽ trình bày công thức tính entropy của một tập hợp.

III.1 Entropy đo tính thuần nhất của tập ví dụ

Khái niệm entropy của một tập S được định nghĩa trong Lý thuyết thông tin là số lượng mong đợi các bít cần thiết để mã hóa thông tin về lớp của một thành viên rút ra một cách ngẫu nhiên từ tập S. Trong trường hợp tối ưu, mã có độ dài ngắn nhất. Theo lý thuyết thông tin, mã có độ dài tối ưu là mã gán $-\log_2 p$ bits cho thông điệp có xác suất là p.

Trong trường hợp S là tập ví dụ, thì thành viên của S là một ví dụ, mỗi ví dụ thuộc một lớp hay có một giá trị phân loại.

Entropy có giá trị nằm trong khoảng [0..1],

Entropy(S) = 0 ⇔ tập ví dụ S chỉ toàn ví dụ thuộc cùng một loại, hay S là thuần nhất.

Entropy(S) = 1 ⇔ tập ví dụ S có các ví dụ thuộc các loại khác nhau với độ pha trộn là cao nhất.

0 < Entropy(S) < 1 ⇔ tập ví dụ S có số lượng ví dụ thuộc các loại khác nhau là không bằng nhau.

Để đơn giản ta xét trường hợp các ví dụ của S chỉ thuộc loại âm (-) hoặc dương (+).

Cho trước:

- Tập S là tập dữ liệu rèn luyện, trong đó thuộc tính phân loại có hai giá trị, giả sử là âm (-) và dương (+)
- p_+ là phần các ví dụ dương trong tập S.
- p_ là phần các ví dụ âm trong tập S.

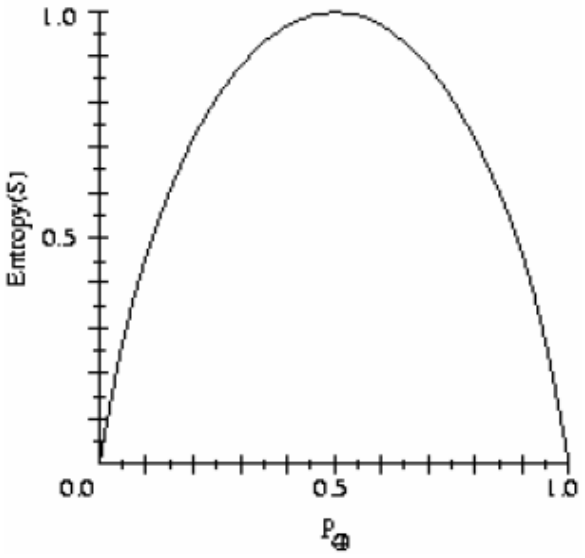
Khi đó, entropy đo độ pha trộn của tập S theo công thức sau:

Entropy(S) = $-p_+ \log_2 p_+ \; -p_- \log_2 p_-$

Một cách tổng quát hơn, nếu các ví dụ của tập S thuộc nhiều hơn hai loại, giả sử là có c giá trị phân loại thì công thức entropy tổng quát là:

$$\text{Entropy}(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Hình 7.4 minh họa sự phụ thuộc của giá trị entropy vào xác suất xuất hiện của ví dụ dương.



Hình 7.4 - Entropy(S)

III.2 Lượng thông tin thu được đo mức độ giảm entropy mong đợi

Entropy là một số đo đo độ pha trộn của một tập ví dụ, bây giờ chúng ta sẽ định nghĩa một phép đo hiệu suất phân loại các ví dụ của một thuộc tính. Phép đo này gọi là lượng thông tin
thu được, nó đơn giản là lượng giảm entropy mong đợi gây ra bởi việc phân chia các ví dụ theo thuộc tính này.
Một cách chính xác hơn, Gain(S,A) của thuộc tính A, trên tập S, được định nghĩa như sau:

$$\text{Gain(S,A)} = \text{Entropy(S)} - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

trong đó Values(A) là tập hợp có thể có các giá trị của thuộc tính A, và S_v là tập con của S chứa các ví dụ có thuộc tính A mang giá trị v.

Câu hỏi :

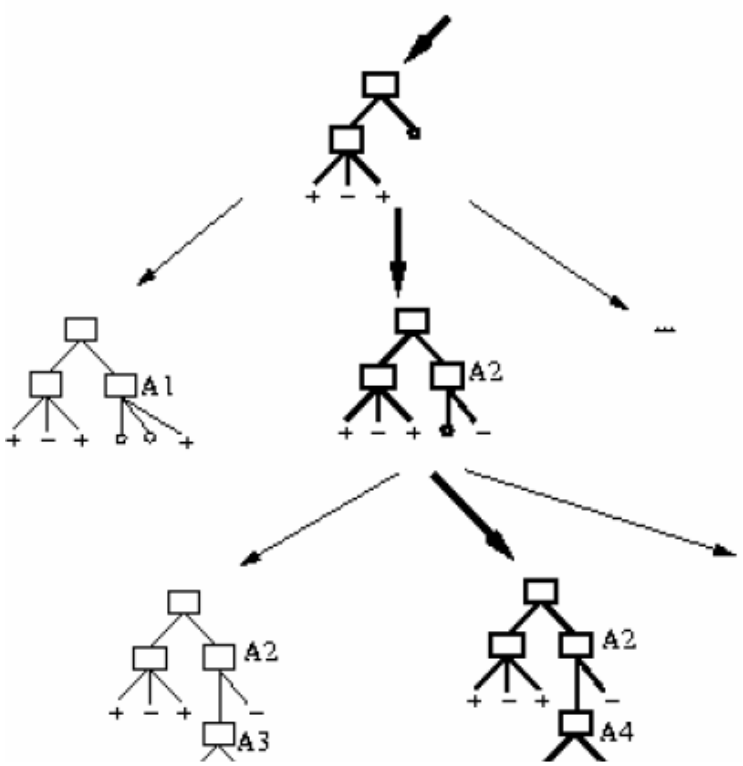
Áp dụng giải thuật ID3 kết hợp với công thức entropy và gain để tạo ra cây quyết định
đơn giản nhất cho bài toán phân loại xem ta 'có đi chơi tennis' từ tập dữ liệu rèn luyện
đã cho trong bảng 7.1.

IV. Tìm kiếm không gian giả thuyết trong ID3

Cũng như các phương pháp học quy nạp khác, ID3 cũng tìm kiếm trong một không gian các giả thuyết một giả thuyết phù hợp với tập dữ liệu rèn luyện. Không gian giả thuyết mà ID3 tìm kiếm là một tập hợp các cây quyết định có thể có. ID3 thực hiện một phép tìm kiếm từ đơn giản đến phức tạp, theo giải thuật leo-núi (hill climbing), bắt đầu từ cây rỗng, sau đó dần dần xem xét các giả thuyết phức tạp hơn mà có thể phân loại đúng các ví dụ rèn luyện. Hàm đánh giá được dùng để hướng dẫn tìm kiếm leo núi ở đây là phép đo lượng thông tin thu được.

Từ cách nhìn ID3 như là một giải thuật tìm kiếm trong không gian các giả thuyết, ta có một số nhận xét như sau:

- Không gian giả thuyết các cây quyết định của ID3 là một không gian đầy đủ các cây quyết định trên các thuộc tính đã cho trong tập rèn luyện. Điều này có nghĩa là không gian mà ID3 tìm kiếm chắc chắn có chứa cây quyết định cần tìm.
- Trong khi tìm kiếm, ID3 chỉ duy trì một giả thuyết hiện tại. Vì vậy, giải thuật này không có khả năng biểu diễn được tất cả các cây quyết định khác nhau có khả năng phân loại đúng dữ liệu hiện có.
- Giải thuật thuần ID3 không có khả năng quay lui trong khi tìm kiếm. Vì vậy, nó có thể gặp phải những hạn chế giống như giải thuật leo núi, đó là hội tụ về cực tiểu địa phương.



Hình 7.5 - Không gian tìm kiếm của ID3.

- Vì ID3 sử dụng tất cả các ví dụ ở mỗi bước để đưa ra các quyết định dựa trên thống kê, nên kết quả tìm kiếm của ID3 rất ít bị ảnh hưởng bởi một vài dữ liệu sai (hay dữ liệu nhiễu).
- Trong quá trình tìm kiếm, giải thuật ID3 có xu hướng chọn cây quyết định ngắn hơn là những cây quyết định dài. Đây là tính chất thiên lệch quy nạp của ID3.

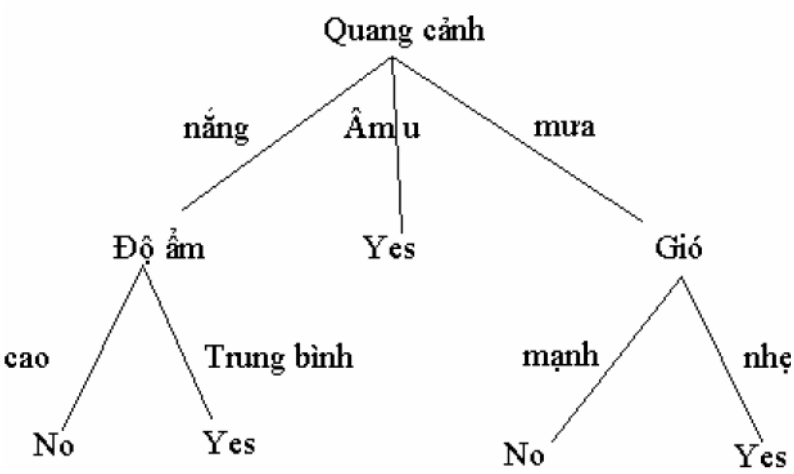
V. Đánh giá hiệu suất của cây quyết định

Một cây quyết định sinh ra bởi ID3 được đánh giá là tốt nếu như cây này có khả năng phân loại đúng được các trường hợp hay ví dụ sẽ gặp trong tương lai, hay cụ thể hơn là có khả năng phân loại đúng các ví dụ không nằm trong tập dữ liệu rèn luyện.

Để đánh giá hiệu suất của một cây quyết định người ta thường sử dụng một tập ví dụ tách rời, tập này khác với tập dữ liệu rèn luyện, để đánh giá khả năng phân loại của cây trên các ví dụ của tập này. Tập dữ liệu này gọi là tập kiểm tra (validation set). Thông thường, tập dữ liệu sẵn có sẽ được chia thành hai tập: tập rèn luyện thường chiếm 2/3 số ví dụ và tập kiểm tra chiếm 1/3.

VI. Chuyển cây về các luật

Thông thường, cây quyết định sẽ được chuyển về dạng các luật để thuận tiện cho việc cài đặt và sử dụng. Ví dụ cây quyết định cho tập dữ liệu rèn luyện trong bảng 9.1 có thể được chuyển thành một số luật như sau :



If (Quang-cảnh =nắng) ∧ (Độ ẩm = Cao) Then Chơi-Tennis = No

If (Quang-cảnh =nắng) ∧ (Độ ẩm = TB) Then Chơi-Tennis = Yes

If (Quang-cảnh =Âm u) Then Chơi-Tennis = Yes

…

VII. Khi nào nên sử dụng ID3

Giải thuật ID3 là một giải thuật học đơn giản nhưng nó chỉ phù hợp với một lớp các bài toán hay vấn đề có thể biểu diễn bằng ký hiệu. Chính vì vậy, giải thuật này thuộc tiếp cận giải quyết vấn đề dựa trên ký hiệu (symbol – based approach).

Tập dữ liệu rèn luyện ở đây bao gồm các ví dụ được mô tả bằng các cặp "Thuộc tính – giá trị", như trong ví dụ 'Chơi tennis' trình bày trong suốt chương này, đó là 'Gió – mạnh', hay 'Gió – nhẹ',… và mỗi ví dụ đều có một thuộc tính phân loại, ví dụ như 'chơi_tennis', thuộc tính này phải có giá trị rời rạc, như có, không.

Tuy nhiên, khác với một số giải thuật khác cũng thuộc tiếp cận này, ID3 sử dụng các ví dụ rèn luyện ở dạng xác suất nên nó có ưu điểm là ít bị ảnh hưởng bởi một vài dữ liệu nhiễu. Vì vậy, tập dữ liệu rèn luyện ở đây có thể chứa lỗi hoặc có thể thiếu một vài giá trị ở một số thuộc tính nào đó. Một giải pháp thường được áp dụng đối với các dữ liệu bị thiếu là sử dụng luật đa số, chương trình tiền xử lý dữ liệu sẽ điền vào các vị trí còn trống giá trị có tần số xuất hiện cao nhất của thuộc tính đó.

Bên cạnh các vấn đề cơ bản được trình bày trong phần này, ID3 còn được thảo luận nhiều vấn đề liên quan như làm sao để tránh cho cây quyết định không bị ảnh hưởng quá nhiều (overfitting) vào dữ liệu rèn luyện, để nó có thể tổng quát hơn, phân loại đúng được cho các trường hợp chưa gặp. Có nhiều giải pháp đã được đưa ra như cắt tỉa lại cây quyết định sau khi học, hoặc cắt tỉa các luật sau khi chuyển cây về dạng luật. Một vấn đề khác nữa đó là nếu như một vài thuộc tính nào đó có giá trị liên tục thì sao. Giải quyết các vấn đề này dẫn đến việc sinh ra nhiều thế hệ sau của ID3, một

giải thuật nổi bật trong số đó là C4.5 (Quinlan 1996). Ngoài ra, một số kỹ thuật được tạo ra để thao tác trên dữ liệu nhằm tạo ra các cây quyết định khác nhau trên cùng tập dữ liệu rèn luyện đã cho như kỹ thuật bagging and boosting.

Chương VIII

TIẾP CẬN KẾT NỐI: **MẠNG NEURON**

I. Giới thiệu

Các mô hình học theo tiếp cận này bắt chước theo cách học của các hệ thần kinh sinh vật. Các hệ thống theo mô hình này có khi còn được gọi là các hệ kết nối (connectionist systems), tính toán neural (Neural computing), mạng neural (Neural Networks), các hệ xử lý phân tán song song (parallel distributed processing – PDP).

Không giống như các giải thuật của tiếp cận ký hiệu, các mô hình này không sử dụng ký hiệu một cách tường minh để giải quyết vấn đề. Thay vào đó, chúng giữ cho trí tuệ phát triển trong các hệ thống gồm các thành phần (neuron sinh học hay neuron nhân tạo) đơn giản, tương tác thông qua một quá trình học hay thích nghi mà nhờ đó kết nối giữa các thành phần này được điều chỉnh. Việc xử lý của các hệ thống này được phân tán trên một tập hợp các lớp neuron. Các hệ thống này giải quyết vấn đề song song theo nghĩa rằng tất cả các neuron trong tập hợp hay trong các lớp sẽ xử lý tín hiệu vào một cách đồng thời và độc lập.

Trong khi các giải thuật của tiếp cận ký hiệu sử dụng ký hiệu để mô tả các mẫu của bài toán như ta đã thấy trong giải thuật ID3 thì những nhà thiết kế mạng neuron phải tạo ra một sơ đồ mã hóa các mẫu (pattern) của bài toán thành các đại lượng số để đưa vào mạng. Việc chọn lựa một sơ đồ mã hóa thích hợp đóng vai trò quyết định cho sự thành công hay thất bại trong việc học của mạng.

Các mẫu (pattern) của bài toán được mã hóa thành các vector số. Các kết nối giữa các thành phần, hay neuron, cũng được biểu diễn bằng các giá trị số. Cuối cùng, sự biến đổi của các mẫu cũng là kết quả của các phép toán số học, thông thường là phép nhân ma trận. Sự chọn lựa kiến trúc kết nối của nhà thiết kế mạng neuron góp phần vào tính thiên lệch quy nạp (inductive bias) của hệ thống.

Các giải thuật và kiến trúc dùng để cài đặt mạng neuron thường được huấn luyện (trained) hay tạo điều kiện (conditioned) chứ không được lập trình một cách tường tận. Và đây chính là sức mạnh chủ yếu của tiếp cận này.

Các phương pháp của tiếp cận này phát huy sức mạnh của chúng trong các bài toán mà khó có thể giải quyết bằng các mô hình ký hiệu. Tiêu biểu là các bài toán đòi hỏi các kỹ năng dựa vào nhận thức, hay các bài toán thiếu một cú pháp định nghĩa rõ ràng.

Các bài toán thích hợp với tiếp cận kết nối thường là:

- Bài toán phân loại (classification): quyết định một giá trị đưa vào thuộc loại hay nhóm nào
- Bài toán nhận dạng mẫu (pattern recognition): nhận dạng cấu trúc trong các dữ liệu có thể là bị nhiễu.
- Bài toán dự đoán (prediction): chẳng hạn như nhận dạng bệnh từ các triệu chứng, nhận dạng tác nhân từ các hiệu ứng,…
- Bài toán tối ưu (optimization): tìm một tổ chức ràng buộc tốt nhất

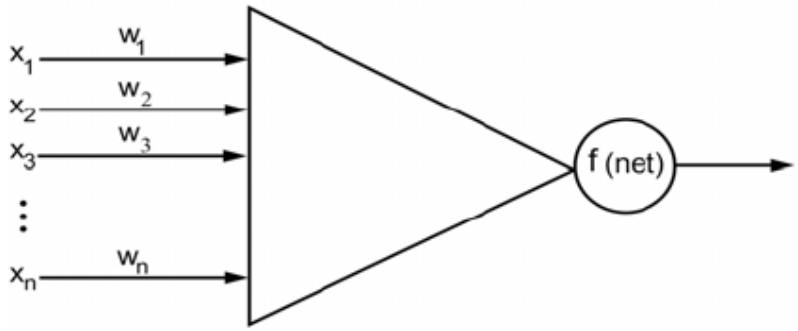
- Bài toán lọc nhiễu (noise filtering): hay phân biệt các tín hiệu với nền, tìm ra các thành phần không quan trọng trong một tín hiệu.

II. Cơ bản về mạng kết nối

Thành phần cơ bản của một mạng neuron là một neuron nhân tạo, như mô tả trong hình 9.6 sau đây.

II.1 Một neuron nhân tạo

Hình 8. 1 minh họa một neuron nhân tạo bao gồm:



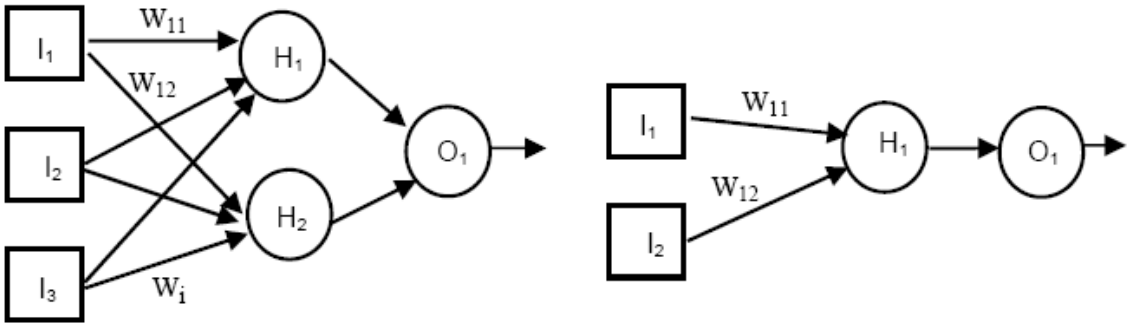
Hình 8.1- Một neuron nhân tạo.

- **Các tín hiệu đầu vào** x_i. Các dữ liệu này có thể đến từ môi trường, hay được kích hoạt từ các neuron khác. Các mô hình khác nhau có thể có miền giá trị của đầu vào khác nhau; thông thường các giá trị đầu vào này là các số rời rạc (discrete) lấy từ tập {0,1} hay {-1,1} hay số thực.
- **Một tập các trọng số** (weight) có giá trị thực w_i. Các trọng số này dùng để mô tả sức mạnh kết nối, hay sức mạnh của các kết nối thiên lệch (bias link)
- **Một mức kích hoạt** (activation level) hay hàm kích hoạt $\Sigma w_i x_i$. Mức kích hoạt của một neuron được xác định bởi sức mạnh tích lũy từ các tín hiệu đầu vào của nó nơi mà mỗi tín hiệu đầu vào được tỷ lệ lại bằng trọng số kết nối w_i ở đầu vào đó. Vì vậy, mức kích hoạt được tính toán bằng cách lấy tổng các giá trị đầu vào sau khi được tỉ lệ hóa, $\Sigma w_i x_i$.
- **Một hàm ngưỡng** (threshold function) f. Hàm này tính kết quả đầu ra của neuron bằng cách xác định xem mức kích hoạt nằm dưới hay trên một giá trị ngưỡng là ít hay nhiều. Hàm ngưỡng này có khuynh hướng tạo ra trạng thái tắt/mở của các neuron.

II.2 Các đặc trưng của một mạng Neuron

Ngoài các tính chất của một neuron đơn lẻ, một mạng neuron còn được đặc trưng bởi các tính chất toàn cục như sau:

- **Hình thái mạng** (network topology): là mô hình hay mẫu kết nối giữa các neuron đơn lẻ.



Hình 8.2 - Các hình thái mạng neuron khác nhau.

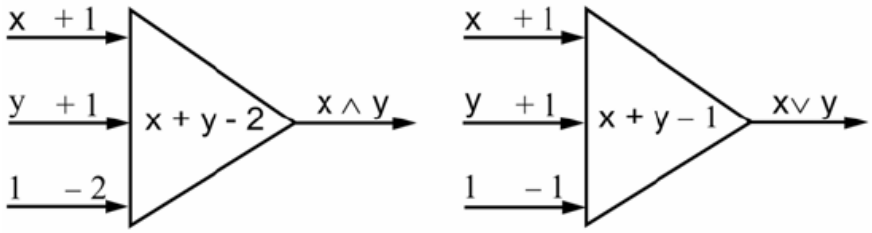
- **Giải thuật học** (learning algorithm): là giải thuật dùng để điều chỉnh các trọng số ở các đầu vào của các neuron. Trong các phần tiếp theo của chương này sẽ trình bày một số giải thuật học tiêu biểu.
- **Sơ đồ mã hóa** (encoding schema): Bao gồm việc thông dịch dữ liệu thực tế thành các giá trị đầu vào của mạng, và việc thông dịch giá trị đầu ra của mạng thành một kết quả có ý nghĩa.

II.3 Mạng neuron McCulloch-Pitts

Ví dụ đầu tiên về tính toán neural được MacCulloch và Pitts đưa ra vào 1943. Đầu vào của một neuron McCulloch-Pitts là +1 (kích thích) hoặc –1 (ức chế). Hàm kích hoạt nhân mỗi đầu vào với giá trị trọng số tương ứng và cộng chúng lại; nếu tổng lớn hơn hay bằng không, thì neuron trả về 1, ngược lại, là –1.

McCulloch-Pitts cho thấy các neuron này có thể được xây dựng để tính toán bất cứ hàm logic nào, chứng minh rằng các hệ thống gồm các neuron này cung cấp một mô hình tính toán đầy đủ.

Hình 8.3 minh họa các neuron McCulloch-Pitts dùng để tính hàm logic and và or.



Hình 8.3 - Các neuron McCulloch-Pitts dùng để tính toán các hàm logic **and** và **or**.

Các neuron này có 3 đầu vào: x và y là các giá trị cần đưa vào, còn đầu vào thứ ba, đôi khi còn được gọi là một thiên lệch (bias), có giá trị hằng là +1.

Ba đầu vào của neuron **and** có 3 trọng số tương ứng là +1, +1 và –2. Vì vậy, với các giá trị bất kỳ của x, y, neuron tính giá trị x+y-2; nếu giá trị này nhỏ hơn 0, nó trả về –1. Ngược lại trả về 1.

Bảng bên dưới minh họa cho tính toán neuron x and y.

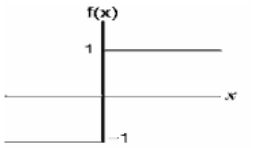
x	y	x + y – 2	Output
1	1	0	1
1	0	–1	–1
0	1	–1	–1
0	0	–2	–1

Mặc dù McCulloch-Pitts chứng minh cho sức mạnh của tính toán neural, nhưng sức hấp dẫn của tiếp cận này chỉ thực sự bắt đầu khi các giải thuật học thực tế bắt đầu phát triển. Phiên bản đầu tiên của mạng neuron có kèm giải thuật học được Frank Rosenblatt đưa ra vào cuối thập niên 1950, có tên gọi là perceptron.

III. Học perceptron

III.1 Giải thuật học perceptron

Perceptron là mạng neuron đơn tầng. Cách lan truyền tín hiệu của perceptron tương tự với neuron McCulloch-Pitts. Các giá trị đầu vào và các mức kích hoạt của perceptron là -1 hoặc 1; trọng số là các số thực. Mức kích hoạt được xác định qua tổng $\Sigma w_i x_i$. Perceptron sử dụng một hàm ngưỡng giới hạn cứng, khi một kích hoạt nằm bên trên ngưỡng, hàm sẽ cho kết quả là 1, và -1 nếu ngược lại. Cho trước các giá trị đầu vào x_i, các trọng số w_i, và một ngưỡng, t, hàm ngưỡng f của perceptron sẽ trả về:



$$\begin{array}{ll} 1 & \text{nếu } \sum w_i x_i >= t \\ -1 & \text{nếu } \sum w_i x_i < t \end{array}$$

Perceptron sử dụng một hình thức đơn giản của học có giám sát (supervised learning). Sau khi perceptron cố gắng giải quyết một mẫu bài toán (mẫu này rút ra từ tập dữ liệu rèn luyện), chương trình đóng vai trò như một người thầy giáo sẽ cung cấp cho nó kết quả đúng của mẫu bài toán đó (giá trị này cũng lấy từ tập dữ liệu rèn luyện). Dựa vào sự khác biệt giữa kết quả đúng được cung cấp và kết quả mà perceptron tính toán được, nó sẽ tự điều chỉnh các trọng số của nó để làm thu hẹp khoảng cách lỗi. Perceptron sử dụng luật như sau: với c là một hằng số cho trước, hằng số này thể hiện tốc độ học và d là giá trị đầu ra mong muốn, perceptron sẽ điều chỉnh trọng số trên thành phần thứ i của vectơ đầu vào một lượng $\Delta w_i = c(d - f(\Sigma w_i x_i))x_i$ $f(\Sigma w_i x_i)$ chính là giá trị đầu ra của perceptron, nó có giá trị +1 hoặc -1. Vì vậy, hiệu giữa d và $f(\Sigma w_i x_i)$ là 0, 2 hoặc -2. Vì vậy, với mỗi thành phần của vectơ đầu vào:

- Nếu giá trị đầu ra mong muốn và giá trị đầu ra thật bằng nhau, thì không làm gì cả.
- Nếu giá trị đầu ra thực là -1 và 1 là giá trị mong muốn, thì tăng trọng số của đường thứ i lên $2cx_i$.

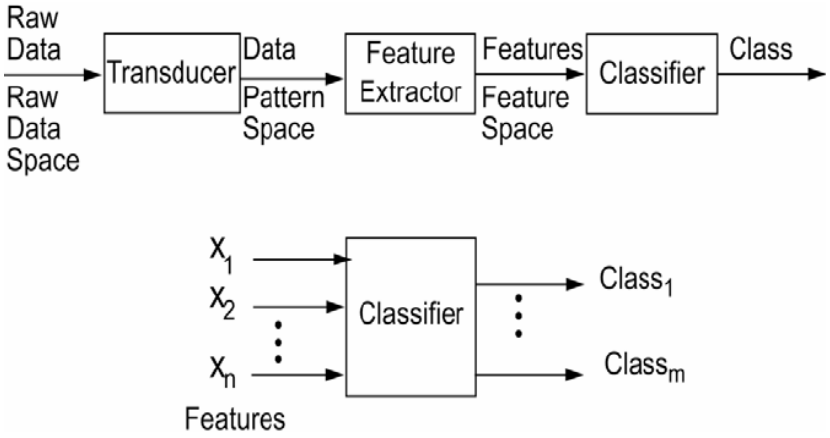
- Nếu giá trị đầu ra thực là 1 và -1 là giá trị mong muốn, thì giảm trọng số của đường thứ i $-2cx_i$

Sở dĩ c được gọi là hằng số thể hiện tốc độ học vì nếu c lớn thì các giá trị điều chỉnh Δw_i sẽ lớn, như vậy, đẩy nhanh quá trình w_i hội tụ về giá trị đúng của nó.

Sau khi được huấn luyện bằng một tập hợp khá lớn các ví dụ rèn luyện cho trước, thủ tục này sẽ sinh ra một tập các trọng số có tính chất làm giảm thiểu trung bình lỗi trên toàn tập ví dụ rèn luyện. Theo Minsky và Papert 1969, nếu tồn tại một tập hợp các trọng số đem lại đầu ra đúng cho mọi ví dụ rèn luyện, thì thủ tục học perceptron sẽ học được nó.

III.2 Sử dụng mạng perceptron cho bài toán phân loại

- Bài toán phân loại



Hình 8.4 - Một hệ thống phân loại đầy đủ.

Hình trên đưa ra một cái nhìn khái quát về bài toán phân loại. Dữ liệu thô từ một không gian các điểm có thể có sau khi qua bộ chuyển đổi (transducer) sẽ được chọn và chuyển đổi thành một không gian các dữ liệu hay mẫu mới. Trong không gian mẫu mới này, bộ trích lọc đặc trưng (feature extractor) sẽ chọn ra các đặc trưng của dữ liệu, và cuối cùng, dữ liệu thể hiện qua các đặc trưng sẽ được đưa vào máy phân loại (classifier) để cho ra kết quả lớp phân loại (class) của dữ liệu đó.

Trong dây chuyền này, mạng perceptron nói riêng và mạng neuron nói chung đóng vai trò như một máy phân loại.

Một dữ liệu đầu vào sẽ được biểu diễn như một vectơ gồm n thành phần (thể hiện cho n đặc trưng) $x_1, x_2, \ldots, x_n$. Các dữ liệu này có thể thuộc một trong m lớp $class_1, class_2, \ldots class_m$. Máy phân loại sẽ có nhiệm vụ xác định xem dữ liệu đầu vào thuộc về lớp nào.

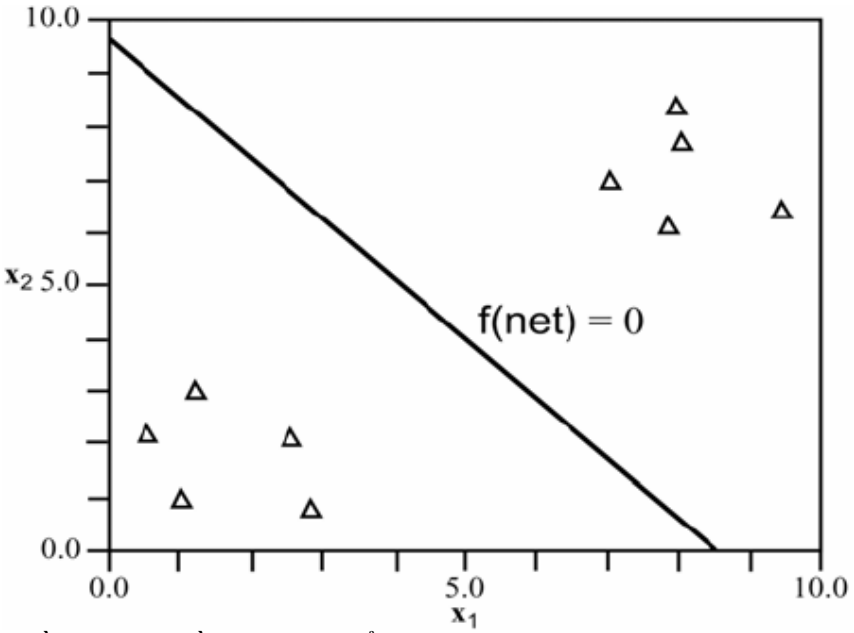
- Ví dụ perceptron

Trong ví dụ đơn giản dưới đây, bộ chuyển đổi và bộ trích lọc đặc trưng đã chuyển thông tin của bài toán thành các tham số của không gian hai chiều. Bảng 8.1 thể hiện dữ liệu rèn luyện của perceptron gồm có 2 đặc trưng (x_1 và x_2) mang giá trị thực, và kết quả mong muốn (output) gồm hai giá trị 1 hoặc -1, thể hiện cho hai lớp phân loại

của dữ liệu. Hình 8.5 thể hiện hình ảnh của các điểm dữ liệu trong bảng 8.1 cùng với đường phân cách hai lớp dữ liệu được tạo ra sau khi mạng percpetron được huấn luyện trên tập dữ liệu này.

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	−1
2.5	2.1	1
8.0	7.7	−1
0.5	2.2	1
7.9	8.4	−1
7.0	7.0	−1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	−1

Bảng 8.1 - Tập dữ liệu cho bài toán phân loại của perceptron.



Hình 8.5 - Đồ thị hai chiều của các điểm dữ liệu trong bảng 8.1. Perceptron cung cấp một phép tách tuyến tính của các tập hợp dữ liệu.

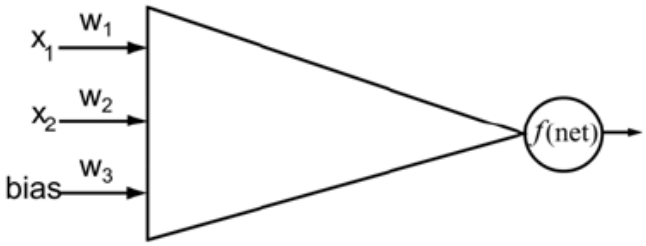
Perceptron dùng để phân loại bài toán này được thiết kế như sau:

Tín hiệu đầu vào: 2 tham số x_1 và x_2, cùng với một đầu vào thiên lệch (bias) luôn có giá trị 1.

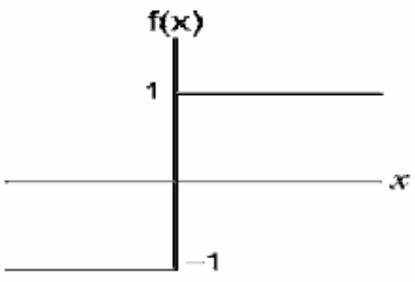
Mức kích hoạt:

$$net = w_1 x_1 + w_2 x_2 + w_3$$

Hàm ngưỡng f(net) là một hàm dấu, hay còn gọi là hàm ngưỡng hai cực tuyến tính



Hình 8.6 - Mạng perceptron cho ví dụ của bảng 8.1.



f(net) = +1, nếu net>0
f(net) = -1, nếu net<=0

Tín hiệu thiên lệch phục vụ cho việc chuyển dịch hàm ngưỡng trên trục tung. Phạm vi của chuyển dịch này sẽ được học bằng cách điều chỉnh trọng số w_3 trong khi huấn luyện mạng.

Bây giờ chúng ta sử dụng các điểm dữ liệu của bảng 8.1 để luyện tập perceptron của hình 8.1.

Giả thiết ban đầu các trọng số w_i có giá trị ngẫu nhiên lần lượt là: [0.75,0.5,-0.6]

Sử dụng giải thuật học perceptron trình bày ở trên với tốc độ học c được chọn là một số dương nhỏ 0.2

Chúng ta bắt đầu bằng ví dụ đầu tiên trong bảng 8.1:

$$f(net)^1 = f(0.75 * 1 + 0.5 * 1 – 0.6 * 1) = f(0.65) = 1$$

Ta thấy $f(net)^1$ có giá trị đúng với giá trị đầu ra mong muốn, nên ta không điều chỉnh trọng số. Cho nên $W^2 = W^1$, W là vectơ trọng số đại diện cho 3 trọng số w_1, w_2,w_3.

Đến ví dụ thứ 2:

$$f(net)^2 = f(0.75 * 9.4 + 0.5 * 6.4 – 0.6 * 1) = f(9.65) = 1$$

Nhưng giá trị mong đợi ở đây là -1, vì vậy, ta cần điều chỉnh trọng số theo luật học:

$$W^t = W^{t-1} + c(d^{t-1} – f(net)^{t-1})X^{t-1}$$

trong đó c là hằng số học

W, X là vectơ trọng số, và vectơ dữ liệu đầu vào

t là thời điểm

Trong trường hợp này: c = 0.2, d^2 = -1, và $f(net)^2$ = 1

Áp dụng luật học trên, ta có:

$$W^3 = W^2 + 0.2(-1-1)X^2 = \begin{bmatrix} 0.75 \\ 0.5 \\ -0.6 \end{bmatrix} - 0.4 \begin{bmatrix} 9.4 \\ 6.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.0 \end{bmatrix}$$

Bây giờ chúng ta xét ví dụ thứ 3:

$$f(net)^3 = f(-3.01 * 2.5 - 2.06 * 2.1 - 1.0 * 1) = f(-12.84) = -1$$

Trong khi giá trị mong đợi của ví dụ này là 1, nên các trọng số tiếp tục được điều chỉnh

$$W^4 = W^3 + 0.2(-1-(-1))X^3 = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.6 \end{bmatrix} + 0.4 \begin{bmatrix} 2.5 \\ 2.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -2.01 \\ -1.22 \\ -0.60 \end{bmatrix}$$

Cứ tiếp tục như thế, sau 10 lần lặp, đường phân cách tuyến tính như trong hình 8.5 xuất hiện. Sau khi lặp lại việc huấn luyện perceptron trên tập dữ liệu đã cho, khoảng 500 lần lặp tổng cộng, vectơ trọng số hội tụ về giá trị [-1.3, -1.1, 10.9]. Và đây chính là các hệ số của phương trình đường phân cách tuyến tính trong hình 8.5:

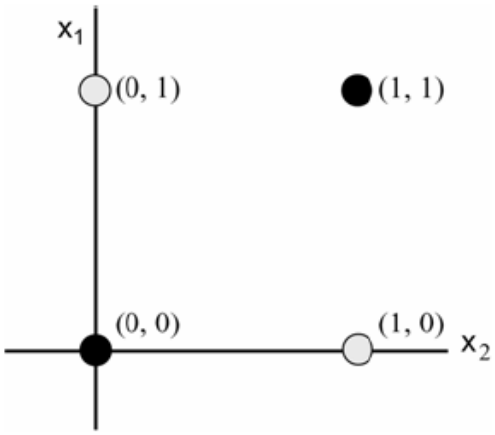
$-1.3 * x_1 - 1.1 * x_2 + 10.9 = 0.$

III.3 Giới hạn của perceptron – tính tách rời tuyến tính của bài toán

Ban đầu, mạng perceptron được chào đón một cách nhiệt tình. Tuy nhiên, Nils Nilsson (1965) và những người khác đã phân tích những giới hạn của mô hình perceptron. Họ chứng minh rằng perceptron không thể giải quyết một lớp các bài toán khó, cụ thể là các bài toán mà các điểm dữ liệu không thể tách rời tuyến tính. Một ví dụ cho bài toán phân loại không tuyến tính đó là phép toán ex-or. Ex-or có bảng chân lý như sau:

x_1	x_2	Output
1	1	0
1	0	1
0	1	1
0	0	0

Bảng 8.2 - Bảng chân lý của phép toán logic ex-or.



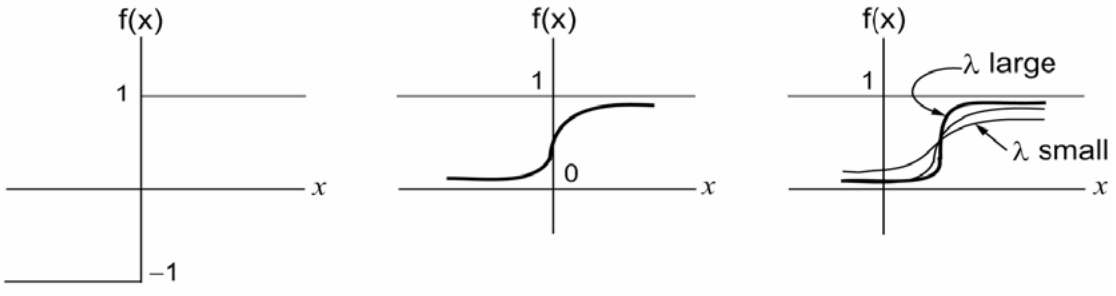
Hình 8.7 - Đồ thị thể hiện các điểm dữ liệu của bài toán Ex-Or

Từ đồ thị hình 8.7, ta thấy không thể tìm được bất cứ một đường thẳng nào có thể tách rời các điểm dữ liệu của lớp 0: {(0,0), (1,1)} ra khỏi các điểm dữ liệu của lớp 1: {(0,1),(1,0)}.

Chúng ta có thể quan niệm tập hợp các giá trị dữ liệu đối với một mạng neuron như dùng để định nghĩa một không gian. Mỗi tham số của dữ liệu đầu vào tương ứng với một chiều trong không gian, và mỗi giá trị đầu vào sẽ định nghĩa một điểm trong không gian đó. Trong bài toán ex-or trên, bốn giá trị đầu vào, được đánh chỉ số theo các tọa độ x_1, x_2, tạo thành các điểm dữ liệu trong hình 8.7. Bài toán học một phân loại nhị phân của các dữ liệu rèn luyện rút lại thành bài toán tách các điểm này thành hai nhóm. Như vậy, đối với một không gian n chiều, một sự phân loại là phân tách được một cách tuyến tính nếu như các lớp của nó có thể được tách ra bởi một mặt phẳng n-1 chiều. (Trong không gian hai chiều thì mặt siêu phẳng n-1 chiều là một đường thẳng, trong không gian 3 chiều thì nó là một mặt phẳng, …).

III.4 Luật Delta

Một cách dễ dàng để tổng quát hóa mạng perceptron là thay hàm ngưỡng giới hạn cứng bằng một hàm kích hoạt kiểu khác. Chẳng hạn như, hàm kích hoạt liên tục (để có khả năng lấy vi phân) tạo điều kiện cho các giải thuật học phức tạp hơn.



a. A hard limiting and bipolar linear threshold

b. A sigmoidal and unipolar threshold

c. The sigmoidal, biased and squashed. As get λ larger the sigmoid approximates a linear threshold

Hình 8.8 - Các hàm ngưỡng.

Hình 8.8 minh họa đồ thị của một số hàm ngưỡng: hình 8.8a là một hàm ngưỡng hai cực, hình 8.8b minh họa một hàm kích hoạt sigmoidal thông dụng (hàm sigmoidal là hàm có hình cong như chữ S), được gọi là hàm **logistic**, hàm này có công thức như sau:

$f(net) = 1/(1 + e^{-\lambda * net})$ với $net = \Sigma_i w_i x_i$

Cũng như các hàm định nghĩa trước đây, x_i là đầu vào của đường thứ i, w_i là trọng số trên đường vào i, và λ là "tham số nén" được sử dụng để điều chỉnh độ cong của đường. Khi λ càng lớn, thì đường cong càng tiệm cận với hàm ngưỡng tuyến tính (trong hình 8.8a). Khi càng tiến gần đến 1, nó càng gần như là một đường thẳng.

Hàm logistic có một tính chất đặc biệt là, đạo hàm của hàm này có một công thức rất đơn giản:

$$f'(net) = f(net) * (1- f(net)) \qquad (1\text{-}1)$$

Từ hình 8.8 ta thấy với các hàm ngưỡng liên tục, neuron sẽ cho kết quả chính xác hơn nhờ vào việc điều chỉnh tham số λ.

Việc đưa ra các hàm kích hoạt liên tục đã làm đề xuất các tiếp cận mới để làm giảm lỗi trong khi học. Qui luật học do Widrow-Hoff đưa ra vào khoảng 1960 độc lập với hàm kích hoạt, tối thiểu hóa bình phương của lỗi giữa giá trị đầu ra mong muốn và kích hoạt của neuron, $net_i = WX_i$. Một trong số luật học quan trọng cho các hàm kích hoạt liên tục là luật delta (Rumelhart et al. 1986).

Để sử dụng luật delta, mạng phải sử dụng một hàm ngưỡng liên tục để có thể lấy vi phân. Hàm logistic đã trình bày bên trên có được tính chất này. Khi đó công thức học theo luật delta cho việc điều chỉnh trọng số ở đầu vào thứ j của nút thứ i là:

$$\Delta w = c(d_i - O_i)\, f'(net_i)x_j$$

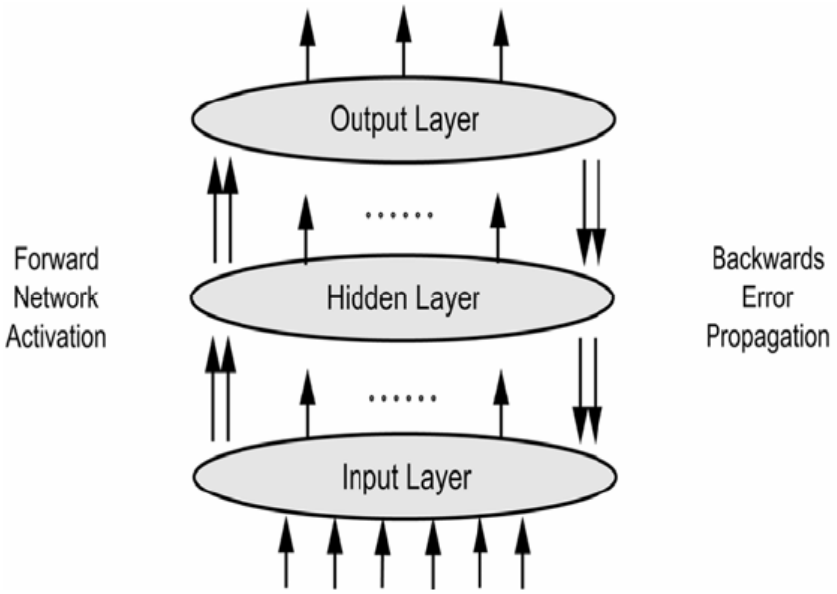
trong đó, c là hằng số điều khiển tốc độ học, d_i và O_i là các giá trị đầu ra thực sự và mong muốn của nút thứ i. f'(net_i) là đạo hàm của hàm kích hoạt cho nút thứ i, và x_j là đầu vào thứ j của nút thứ i. Thay thế công thức đạo hàm (1-1) của hàm logistic f'(net), ta được công thức để điều chỉnh trọng số như sau:

$$\Delta w = c(d_i - O_i)\, O_i\, (1 - O_i)\, x_j \qquad (1\text{-}2)$$

Từ công thức này cho thấy, công thức điều chỉnh trọng số này chỉ có thể áp dụng cho các nút của mạng perceptron đơn tầng, vì tại đó ta mới có các giá trị đầu ra mong muốn d_i.

IV. Học Lan truyền ngược

Như đã phân tích ở trên, ta thấy các mạng perceptron đơn tầng có khả năng giới hạn, chúng không thể phân loại được các bài toán không tách rời tuyến tính. Trong phần tiếp theo, chúng ta sẽ thấy rằng các mạng đa tầng có thể giải quyết được các bài toán này.



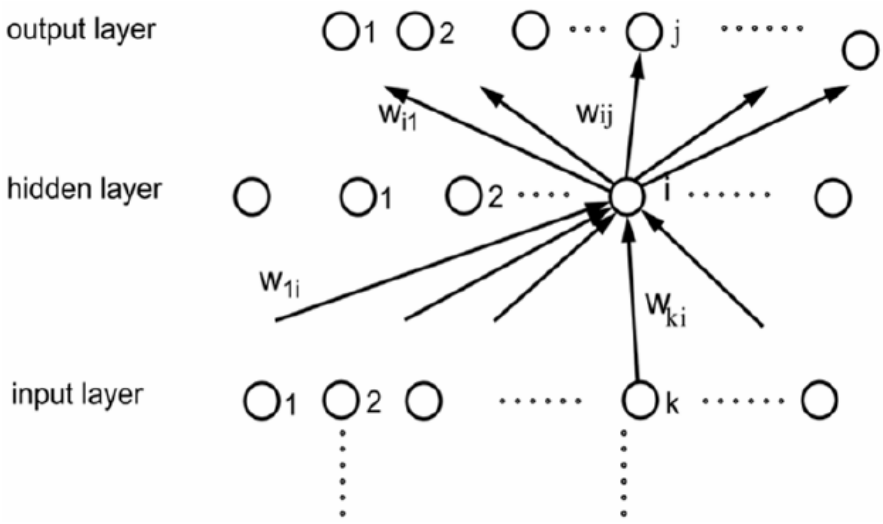
Hình 8.9 - Học lan truyền ngược trong mạng kết nối có một tầng ẩn.

Các neuron trong một mạng đa tầng (xem hình 8.9) được kết nối với nhau theo từng lớp, trong đó các neuron ở tầng k sẽ truyền kích hoạt của chúng chỉ cho các neuron ở tầng k+1. Xử lý tín hiệu đa tầng có nghĩa là các lỗi nằm sâu bên trong mạng có thể lan ra và phát triển một cách phức tạp thông qua các tầng liên tiếp. Vì vậy, việc phân tích nguyên nhân gây ra lỗi ở tầng ra (output layer) là rất phức tạp. Giải thuật học lan truyền ngược sẽ cung cấp một phương pháp điều chỉnh trọng số trong trường hợp này.

IV.1 Giải thuật học lan truyền ngược

Từ lập luận cho rằng tại các nút của một mạng đa tầng, lỗi mà một nút phải chịu trách nhiệm cũng phải được chia phần cho các nút ở tầng ẩn trước nó và vì vậy các trọng số phải được điều chỉnh một cách phù hợp.

Giải thuật lan truyền ngược bắt đầu tại tầng ra và truyền các lỗi ngược về xuyên qua các tầng ẩn (như hình 8.10).



Hình 8.10 - Σ_j –$delta_j$ *w_{ij} là tổng đóng góp của nút i vào lỗi ở tầng ra.

Luật delta tổng quát để điều chỉnh trọng số của đầu vào thứ k của nút thứ i:

$\Delta w_k = c(d_i - O_i)\, O_i\, (1 - O_i)\, x_k$ cho nút ở tầng ra

$\Delta w_k = c\, \Sigma_j\, (delta_j\, w_{ij})\, O_i\, (1 - O_i)\, x_k$ cho nút ở tầng ẩn

với $delta_j = (d_j - O_j)\, O_j\, (1 - O_j)$

j chạy trên các nút của tầng kế tiếp mà tại đó nút i truyền các đầu ra của nó.

Đối với mạng có nhiều hơn một tầng ẩn, ta cũng áp dụng thủ tục tương tự một cách đệ quy để truyền lỗi từ tầng ẩn thứ n vào tầng ẩn thứ n-1.

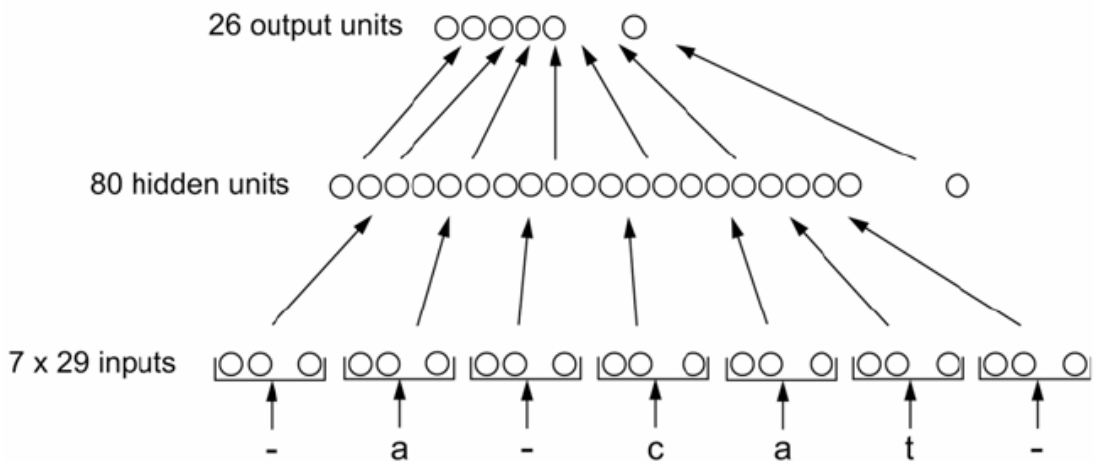
IV.2 Ví dụ 1: Mạng NetTalk

Mạng NETtalk là một ví dụ hay cho việc sử dụng giải pháp mạng neuron để giải quyết một vấn đề học khó. NETtalk học để đọc được văn bản tiếng Anh. Đây là một nhiệm vụ khó khăn đối với tiếp cận học dựa trên ký hiệu, vì phát âm trong tiếng Anh mang tính bất quy tắc. Mặc dù có các chương trình dựa trên luật (rule-based) đã được viết để giải quyết vấn đề này, nhưng chúng đều phức tạp và thực hiện chưa hoàn hảo.

NETtalk học để đọc một chuỗi văn bản và trả về một âm vị cùng với trọng âm liên hệ cho mỗi chữ cái trong chuỗi. Vì phát âm của một chữ cái đơn nhất phụ thuộc vào các chữ cái xung quanh nó, người ta đưa vào NETtalk một cửa sổ gồm 7 ký tự. Khi văn bản dịch chuyển qua cửa sổ này, NETtalk trả về một cặp âm vị/trọng âm cho mỗi chữ cái.

Hình 8.11 minh họa kiến trúc của mạng NETtalk. Mạng gồm có 3 tầng neuron. Các neuron đầu vào tương ứng với cửa sổ 7 ký tự của văn bản. Mỗi vị trí trong cửa sổ được biểu diễn bởi 29 neuron đầu vào, 26 neurons cho 26 ký tự alphabet, và 3 neurons cho dấu và khoảng trắng. Ký tự ở mỗi ví trí trong cửa sổ sẽ kích hoạt neuron tương ứng. Các neuron đầu ra mã hóa âm sử dụng 21 đặc điểm khác nhau của cách phát âm

của con người. 5 neurons còn lại mã hóa dấu nhấn và giới hạn âm tiết. NETtalk có 80 neuron ở tầng ẩn, 26 giá trị đầu ra và 18.629 kết nối.



Hình 8.11 - Hình thái mạng của NETtalk.

Kết quả của NETtalk là có thể phát âm đúng 60% sau khi rèn luyện với một tập dữ liệu rèn luyện gồm 500 ví dụ và lặp lại 100 lượt.

Ngoài kết quả đạt được trên, NETtalk còn cho thấy một số tính chất đáng chú ý của mạng neuron, có nhiều tính chất trong số đó phản ánh bản chất tự nhiên của việc học ở người. Chẳng hạn như, việc học, khi được đo bằng phần trăm câu trả lời đúng, sẽ tiến triển nhanh lúc đầu, sau đó chậm dần khi tỉ lệ đúng tăng lên. Và cũng như con người, khi neuron càng học phát âm được nhiều từ, thì nó càng phát âm đúng các từ mới nhiều hơn.

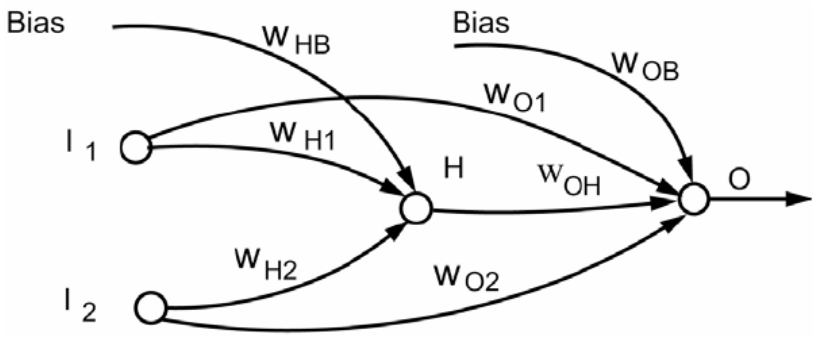
IV.3 Ví dụ 2: Exclusive–or

Một ví dụ khác cho mạng đa tầng là dùng để giải quyết bài toán Ex-or mà mạng đơn tầng không thể phân loại được.

Hình 8.12 minh họa mạng với hai đầu vào, một nút ẩn và một nút đầu ra. Mạng cũng có hai đầu vào thiên lệch (bias), một đi vào nút ẩn và một đi vào nút đầu ra. Một điểm đặc biệt là các đầu vào cũng được nối trực tiếp vào nút đầu ra. Liên kết thêm vào này cho phép nhà thiết kế mạng neuron đạt được một mạng với ít nút hơn trên tầng ẩn và hội tụ nhanh hơn.

Giá trị net cho nút ẩn và nút đầu ra cũng được tính như cách thông thường, là tổng của các tích giữa giá trị đầu nhân với trọng số. Các trọng số được điều chỉnh theo giải thuật học lan truyền ngược và sử dụng hàm kích hoạt sigmoidal.

Thật ra, mạng neuron trong hình 8.12 không phải là một mạng duy nhất có thể giải quyết bài toán này.



Hình 8.12 - Một mạng lan truyền ngược dùng để giải quyết bài toán exclusive-or.

Mạng này được rèn luyện với 4 ví dụ: (0,0) → 0; (1,0) → 1; (0,1) → 1; (1,1) → 0

Sau khi được huấn luyện 1400 lượt với 4 dữ liệu trên, các trọng số hội tụ về các giá trị như sau:

$W_{H1} = -7.0$ $W_{HB} = 2.6$ $W_{O1} = -5.0$ $W_{H2} = -7.0$

$W_{OB} = 7.0$ $W_{O2} = -4.0$ $W_{OH} = -11.0$

Với giá trị đầu vào là (0,0), giá trị đầu ra của nút ẩn sẽ là:

f(0 * (-7.0) + 0 * (-7.0) + 1* 2.6) = f(2.6) → 1

Kết quả trả về của nút đầu ra cho (0,0) sẽ là:

f(0 * (-5.0) + 0 * (-4.0) + 1 * (-11.0) + 1 * (7.0)) = f(-4.0) → 0

Như vậy, ta thấy rằng mạng lan truyền ngược đã phân loại được các điểm dữ liệu không tuyến tính.

V. Nhận xét chung về mạng neuron

Nói chung các mạng đa tầng là đầy đủ về mặt tính toán (computationally complete), có nghĩa là có thể giải quyết được mọi bài toán. Tuy nhiên, để thiết kế một mạng neuron đa tầng thì nhà thiết kế phải giải quyết được những vấn đề sau:

- Làm sao để chọn số nút ẩn và số tầng ẩn thích hợp?
- Khi nào sử dụng các nút thiên lệch?
- Cách chọn một tập rèn luyện?
- Điều chỉnh các trọng số như thế nào?
- Nên chọn tốc độ học như thế nào?

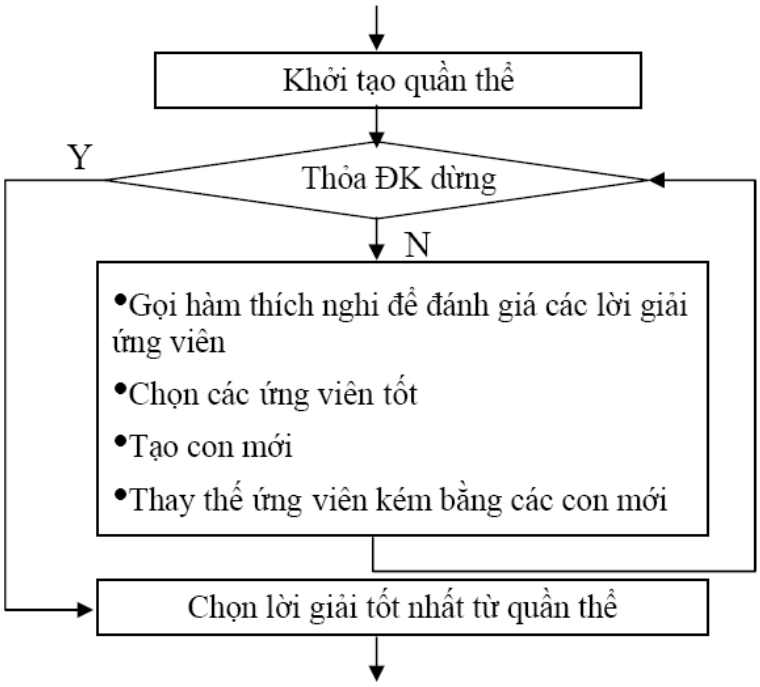
Nói chung, không có một quy luật nào về tất cả những điều này, nó phụ thuộc vào kinh nghiệm của nhà thiết kế, cũng như là kết quả của quá trình thử-sai lặp đi lặp lại.

Chương IX

TIẾP CẬN XÃ HỘI VÀ NỔI TRỘI: **GIẢI THUẬT DI TRUYỀN** (GENETIC ALGORITHM - GA)

Cũng như các mạng neuron, các thuật toán di truyền cũng dựa trên một ẩn dụ sinh học: Các thuật toán này xem việc học như là sự cạnh tranh trong một quần thể gồm các lời giải ứng viên đang tiến hóa của bài toán. Một hàm 'thích nghi' (fitness function) sẽ đánh giá mỗi lời giải để quyết định liệu nó có đóng góp cho thế hệ các lời giải kế tiếp hay không. Sau đó, thông qua các phép toán tương tự với biến đổi gene trong sinh sản hữu tính, giải thuật sẽ tạo ra một quần thể các lời giải ứng viên mới.

I. Giải thuật



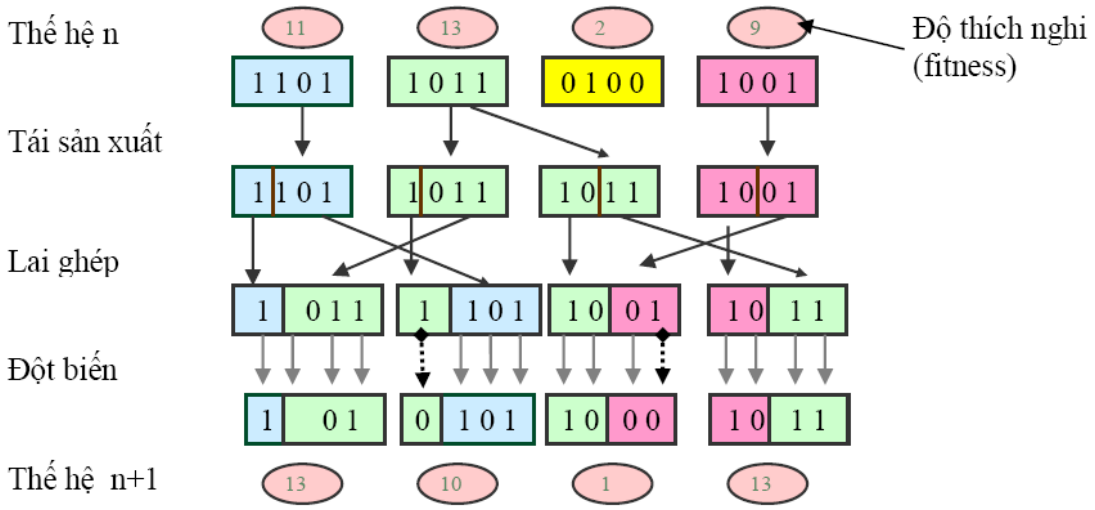
Hình 9.1- Giải thuật di truyền.

Hình 9.1 mô tả giải thuật di truyền tổng quát. Tùy theo từng bài toán mà nhà thiết kế giải thuật sẽ phải mô tả chi tiết hơn về:

- Phương pháp biểu diễn một cá thể trong quần thể các lời giải ứng viên của bài toán, hay nói khác hơn là hình thức biểu diễn một lời giải tiềm năng của bài toán. Không phải lời giải của mọi bài toán đều có thể được mã hóa một cách dễ dàng và tự nhiên dưới dạng biểu diễn mức bit như trong bài toán thỏa mãn CNF dưới đây.
- Độ lớn của quần thể là số lượng ứng viên có trong quần thể. Thông thường các ứng viên của quần thể ban đầu được chọn một cách ngẫu nhiên. Độ lớn của quần

thể là không đổi qua các thế hệ, vì vậy, sẽ có một quá trình chọn lọc và loại bỏ một số lời giải ứng viên có độ thích nghi thấp.

- Điều kiện dừng của vòng lặp: có thể là chương trình đạt tới một số lần lặp nhất định nào đó, hay đạt tới trung bình độ tốt nào đó của quần thể,…
- Hàm đánh giá (fitness function): Dùng để đánh giá một ứng viên có tốt hay không. Một ứng viên càng tốt nghĩa là độ thích nghi của nó càng cao và tiến đến trở thành lời giải đúng của bài toán. Việc thiết kế một hàm đánh giá tốt là rất quan trọng trong thuật toán di truyền. Một hàm đánh giá không chính xác có thể làm mất đi các ứng viên tốt trong quần thể.
- Chọn lựa bao nhiêu phần trăm lời giải tốt để giữ lại? Hay chọn bao nhiêu lời giải ứng viên để kết hợp với nhau và sinh ra lời giải con?
- Phương pháp tạo thành viên mới từ thành viên hiện có, còn gọi là toán tử di truyền (genetic operators): Các toán tử di truyền phổ biến là
  - Lai ghép (cross-over): Toán tử lai ghép lấy hai lời giải ứng viên và chia từng lời giải ra thành hai phần, sau đó trao đổi các phần với nhau để tạo ra ứng viên mới. Ví dụ xem hình 9.18.
  - Đột biến (mutation): Đột biến lấy một ứng viên đơn lẻ và thay đổi một cách ngẫu nhiên một khía cạnh nào đó của nó. Ví dụ xem hình 9.2.

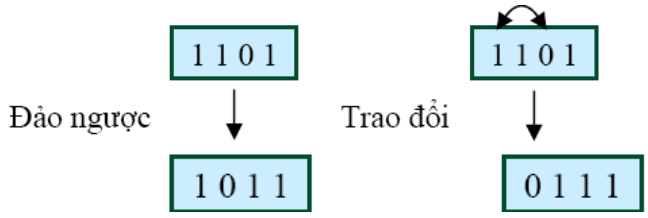


Hình 9.2 - Ví dụ minh họa giải thuật và toán tử di truyền.

Trong ví dụ minh họa bằng hình 9.2, ta thấy tại thế hệ thứ n ta có một lời giải có độ thích nghi rất thấp (2), và vì vậy, nó không được sử dụng trong quá trình tái sản xuất. Thay vào đó, lời giải có độ thích nghi cao nhất (13) sẽ được nhân đôi và đưa vào quá trình tái sản xuất.

Hoặc ít phổ biến hơn là các toán tử di truyền:

- Đảo ngược (inversion): Đảo ngược thứ tự các bit trong mẫu lời giải.
- Trao đổi (Exchange): Trao đổi hai bit bất kỳ trong mẫu lời giải với nhau.



Một toán tử di truyền tốt đóng một vai trò quan trọng trong thuật toán di truyền. Toán tử di truyền phải bảo toàn những mối quan hệ cốt yếu trong quần thể; ví dụ, sự có mặt và sự duy nhất của tất cả các thành phố trong hành trình của người bán hàng trong bài toán người đi bán hàng.

- Thay thế thành viên mới cho các thành viên hiện có như thế nào?

- …

II Ví dụ 1: Bài toán thỏa CNF

Bài toán thỏa mãn dạng chuẩn hội (Conjunctive normal form – CNF) là một bài toán đơn giản: Một biểu thức của các mệnh đề (clause) ở dạng chuẩn hội hay CNF khi nó là một dãy các biến mệnh đề được kết nối với nhau bởi toán tử quan hệ and (∧). Mỗi mệnh đề có dạng là một tuyển (disjunction), gồm các toán tử quan hệ or (∨) trên các biến mệnh đề (literal).

Ví dụ : Nếu ta có 6 biến mệnh đề a, b, c, d, e và f, thì biểu thức sau đây là một CNF:

$(\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \square d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee f)$ (1-3)

Thỏa mãn CNF có nghĩa rằng chúng ta phải tìm ra một phép gán true hoặc false (1 hoặc 0) cho mỗi biến mệnh đề a, b, c, d, e, f sao cho biểu thức CNF có giá trị là TRUE.

Một cách biểu diễn tự nhiên cho lời giải của bài toán này là một dãy sáu bit, mỗi bit theo thứ tự a, b, c, d, e, f biểu diễn true (1) hoặc false (0) cho mỗi biến mệnh đề. Như vậy mẫu bit:

1 0 1 0 1 0

cho biết a, c, và e là true và b, d, và f là false, và do đó khi thay các giá trị này vào biểu thức (1-3), thì cho giá trị false.

Chúng ta muốn rằng các toán tử di truyền sinh ra các thế hệ lời giải sao cho biểu thức CNF mang trị true, vì vậy mỗi toán tử phải sinh ra một mẫu 6-bit của phép gán true cho biểu thức. Cách biểu diễn lời giải dưới dạng một mẫu các bit như trên mang lại cho ta rất một điều rất thuận lợi là bất kỳ toán tử di truyền nào (lai ghép, đột biến, đảo ngược, hay trao đổi) đều tạo ra một mẫu bit mới là một lời giải khả dĩ hợp lệ.

Việc chọn lựa một hàm thích nghi cho quần thể các chuỗi bit này không phải hoàn toàn dễ dàng. Thoạt nhìn chuỗi bit, ta khó có thể xác định một hàm thích nghi để đánh giá được chất lượng của nó như thế nào, nghĩa là khó đoán được độ tốt của nó so với

đáp án đúng. Đáp án đúng ở đây chính là chuỗi bit sao cho biểu thức CNF có giá trị true.

Tuy nhiên có một số cách khác. Nếu ta chú ý đến biểu thức CNF (1-3), thì ta thấy rằng nó được tạo thành từ hội của 5 mệnh đề. Do đó chúng ta có thể thiết lập một hệ phân hạng cho phép chúng ta sắp hạng các lời giải (mẫu bit) tiềm năng trong khoảng giá trị từ 0 đến 5, tùy thuộc vào số mệnh đề mà mẫu đó thỏa mãn. Do đó mẫu:

1 1 0 0 1 0 có độ thích nghi là 1

0 1 0 0 1 0 có độ thích nghi là 2

0 1 0 0 1 1 có độ thích nghi là 3

1 0 1 0 1 1 có độ thích nghi là 5, và nó chính là một lời giải.

III. Ví dụ 2: Bài toán người đi bán hàng TSP

Bài toán người bán hàng (traveling salesperson problem – TSP) là một bài toán cổ điển đối với AI và khoa học máy tính[1]. Như chúng đã giới thiệu ở chương III, toàn bộ không gian trạng thái của nó đòi hỏi phải xem xét N! trạng thái để có thể tìm ra lời giải tối ưu, trong đó N là số thành phố cần đi qua. Khi N khá lớn thì bài toán sẽ bị bùng nổ tổ hợp, vì vậy người ta đặt vấn đề là có cần thiết hay không cho việc chạy một máy trạm làm việc đắt tiền trong nhiều giờ để cho một lời giải tối ưu hay chỉ nên chạy một PC rẻ tiền trong vài phút để có được những kết quả "đủ tốt". Giải thuật di truyền chính là một giải pháp cho lựa chọn thứ hai.

Ở bài toán này, dùng mẫu bit để biểu diễn cho lời giải của bài toán không phải là một cách hay. Chẳng hạn, ta có chín thành phố cần ghé thăm 1, 2, …9, ta xem mỗi thành phố như một mẫu 4 bit 0001, 0010,… 1001. Khi đó một lời giải khả dĩ sẽ có hình thức như sau:

0001 0010 0011 0100 0101 0110 0111 1000 1001

Với cách biểu diễn như vậy, việc thiết kế các toán tử di truyền sẽ trở nên rất khó khăn. Toán tử lai ghép nhất định là không được, vì chuỗi mới được tạo từ hai cha mẹ khác nhau hầu như sẽ không biểu diễn một đường đi trong đó ghé thăm mỗi thành phố đúng một lần. Trong thực tế, với lai ghép, một số thành phố có thể bị xóa bỏ trong khi các thành phố khác được ghé thăm nhiều hơn một lần, và vì vậy đó không phải là một lời giải hợp lệ. Còn toán tử đột biến thì thế nào? Giả sử bit trái nhất của thành phố thứ sáu, 0110, được đột biến thành 1? 1110, hay là 14, thì nó không còn là một thành phố hợp lệ.

Một cách tiếp cận khác là sẽ bỏ qua biểu diễn dạng mẫu bit và đặt cho mỗi thành phố một tên theo bảng chữ cái hoặc số, ví dụ 1, 2, …9; xem đường đi qua các thành phố là một sự sắp thứ tự của chín ký số này, và sau đó chọn toán tử di truyền thích hợp để tạo ra các đường đi mới. Ở đây ta thấy phép trao đổi (exchange) ngẫu nhiên hai thành phố trong đường đi có thể sử dụng được, còn phép toán lai ghép (crossover) thì không. Việc trao đổi các đoạn của một đường đi với những đoạn khác của cùng đường đi đó, hoặc bất cứ toán tử nào sắp xếp lại các chữ cái của đường đi ấy (mà không xóa bỏ, thêm, hay nhân đôi bất cứ thành phố nào) đều có thể sử dụng được. Tuy nhiên, những phương pháp này gây khó khăn cho việc đưa vào thế hệ con cháu những thành phần

"tốt hơn" của các mẫu trong các đường đi qua của các thành phố của hai cha mẹ khác nhau.

Nhiều nhà nghiên cứu đã đưa ra các toán tử lai ghép có khả năng khắc phục những vấn đề này, trong đó có toán tử lai ghép có thứ tự (order crossover) do Davis đưa ra vào năm 1985. Lai ghép có thứ tự xây dựng con cháu bằng cách chọn một dãy con các thành phố trong đường đi của một mẫu cha mẹ. Nó cũng bảo toàn thứ tự tương đối các thành phố từ cha mẹ kia. Đầu tiên, chọn hai điểm cắt, biểu thị bởi dấu "|", điểm cắt này được chen vào một cách ngẫu nhiên vào cùng một vị trí của mỗi mẫu cha mẹ. Những điểm cắt này là ngẫu nhiên, nhưng một khi được chọn, thì những vị trí như nhau sẽ được sử dụng cho cả hai cha mẹ. Ví dụ, có hai mẫu cho mẹ p_1 và p_2, với các điểm cắt sau thành phố thứ ba và thứ bảy:

p_1 = (1 9 2 | 4 6 5 7 | 8 3)

p_2 = (4 5 9 | 1 8 7 6 | 2 3)

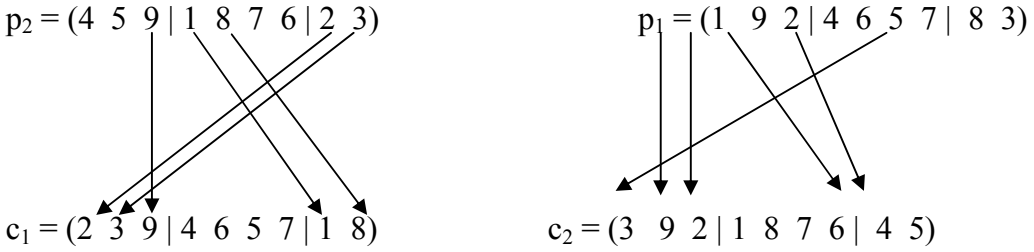
[1] Phát biểu của bài toán TSP: Một người bán hàng có nhiệm vụ ghé thăm N thành phố như là một phần của lộ trình bán hàng. Đường đi giữa mỗi cặp thành phố có một chi phí (ví dụ như độ dài đoạn đường, giá vé máy bay). Hãy tìm ra đường đi có chi phí thấp nhất cho người bán hàng để bắt đầu lên đường tại một thành phố, thăm tất cả các thành phố khác chỉ đúng một lần rồi quay lại thành phố xuất phát.

Hai mẫu con c_1 và c_2 sẽ được sinh ra theo cách sau. Đầu tiên, các đoạn giữa hai điểm cắt sẽ được chép vào các mẫu con:

c_1 = (x x x | 4 6 5 7 | x x)

c_2 = (x x x | 1 8 7 6 | x x)

Bước kế tiếp là bắt đầu từ điểm cắt thứ hai của một trong hai mẫu cha mẹ, nếu ta đang muốn hoàn tất mẫu c_1, thì ta sẽ bắt đầu từ điểm cắt thứ hai của mẫu p_2, ta chép các thành phố từ điểm cắt này theo thứ tự vào các chỗ còn trống của c_1, bỏ qua những thành phố mà c_1 đã có (các ký số được in đậm và nghiêng trong sơ đồ bên dưới). Khi đến cuối mẫu p_2, thì quay lại đầu mẫu p_2 tiếp tục chép sang c_1 cho đến khi c_1 đủ.



Với giải thuật lai ghép này, các đường đi của thế hệ con sẽ được đảm bảo là các đường đi hợp lệ, đi qua mỗi thành phố một lần duy nhất.

Tóm lại, trong lai ghép thứ tự, các mảnh của một đường đi được truyền từ một cha mẹ, p_1, sang một con, c_1, trong khi sắp xếp của các thành phố còn lại của con c_1 được thừa kế từ cha mẹ kia, p_2. Điều này ủng hộ cho trực giác hiển nhiên là thứ tự của các thành phố đóng vai trò quan trọng trong việc tạo ra đường đi với chi phí thấp nhất, và vì vậy việc truyền lại các đoạn thông tin có thứ tự này từ các cha mẹ có độ thích nghi cao sang con cái là một điều rất quan trọng.

Ngoài toán tử lai ghép thứ tự trên, còn có rất nhiều toán tử lai ghép và đột biến khác được đưa ra để giải quyết bài toán này. Bảng 9.1 liệt kê các toán tử lai ghép, bảng 9.2 liệt kê các toán tử đột biến.

Tên toán tử	Năm	Tác giả
Alternating Position Crossover (AP)	(1999)	Larranaga, Kuijpers, Poza, Murga
Cycle Crossover (CX)	(1987)	Oliver, Smith and Holland
Distance Preserving Crossover (DPX)	(1996)	Freisbein and Merz
Edge Assembly Crossover (EAX)	(1997)	Nagata and Kobayashi
Edge Recombination Crossover (ER)	(1989)	Whitley, Timothy, Fuquay
Heuristic Crossover (HEU)	(1987)	Grefenstette
Inver-over Operator (IOO)	(1998)	Tao and Michalewicz
Maximal Preservative Crossover (MPX)	(1988)	Mühlenbein, Schleuter and Krämer
Position Based Crossover (POS)	(1991)	Syswerda
Order Crossover (OX1)	(1985)	Davis
Order Based Crossover (OX2)	(1991)	Syswerda
Partially mapped Crossover (PMX)	(1985)	Goldberg and Lingle
Voting Recombination Crossover (VR)	(1989)	Mühlenbein

Bảng 9.1 - Danh sách các toán tử lai ghép cho bài toán TSP.

Tên toán tử	Năm	Tác giả
Displacement Mutation (DM)	(1992)	Michalewicz
Exchange Mutation (EM)	(1990)	Banzhaf
Insertion Mutation (ISM)	(1988)	Fogel
Inversion Mutation (IVM)	(1990)	Fogel
Scramble Mutation (SM)	(1991)	Syswerda
Simple Inversion Mutation (SIM)	(1975)	Holland

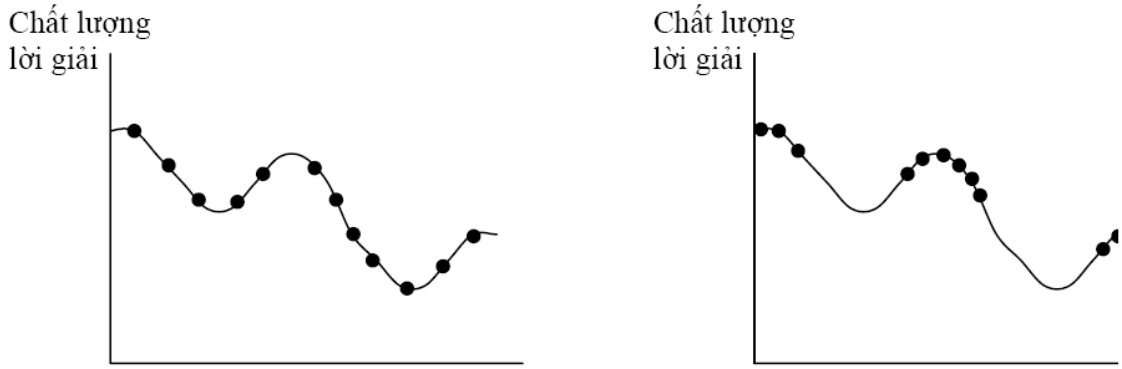
Bảng 9.2 - Danh sách các toán tử đột biến cho bài toán TSP.

IV. Đánh giá giải thuật

Các ví dụ vừa nêu trên làm nổi bật những vấn đề mang tính duy nhất của thuật toán di truyền về biểu diễn tri thức, chọn toán tử di truyền, và thiết kế hàm thích nghi. Biểu diễn được chọn phải hỗ trợ cho các toán tử di truyền. Một điểm dáng lưu ý nữa là các toán tử di truyền phải được thiết kế sao cho bảo lưu được những mảnh thông tin có ý nghĩa trong lời giải tiềm năng từ thế hệ này sang các thế hệ tiếp theo.

Một sức mạnh quan trọng của thuật toán di truyền là bản chất song song trong tìm kiếm của nó. Các thuật toán này thực hiện một dạng mạnh của leo núi (hill climbing)

trong đó duy trì nhiều lời giải (trong quần thể các lời giải), loại bỏ những lời giải không có triển vọng, và nâng cao chất lượng của những lời giải tốt. Hình 9.3 phỏng theo Holland (1986), cho thấy nhiều lời giải hội tụ về các điểm tối ưu trong không gian tìm kiếm. Trong hình này, trục hoành biểu diễn các điểm có thể có trong không gian lời giải, trong khi trục tung phản ánh chất lượng của những lời giải đó. Các điểm chấm nằm trên cung là các thành viên của quần thể hiện tại. Khởi đầu, những lời giải được rải trong không gian những lời giải có thể có. Sau một vài thế hệ, chúng có khuynh hướng cụm lại xung quanh những vùng có chất lượng lời giải cao hơn.



Hình 9.3- Các thuật toán di truyền được xem là leo núi song song (theo Holland 1986)

Tuy nhiên, với sức mạnh như vậy, giải thuật genetic cũng không thể áp dụng cho tất cả các bài toán có thể có. Vì như ta thấy qua hai ví dụ trên, lời giải của bài toán phải được biểu diễn dưới một dạng mẫu thích hợp cho các toán tử di truyền hoạt động. Trong thực tế có nhiều bài toán không thể làm được điều này. Vì vậy, khi nghiên cứu về giải thuật này, có rất nhiều câu hỏi đã được đưa ra nhằm hiểu rõ hơn nữa về bản chất hoạt động của nó:

1. Liệu chúng ta có thể đưa ra những đặc điểm về các loại bài toán mà giải thuật di truyền (GA) có thể thực hiện tốt
2. Các loại bài toán nào thì không thích hợp với GA.
3. Dựa vào đâu để ta có thể nói là GA thực hiện tốt hay không tốt đối với một loại bài toán nào đó?
4. Liệu có những qui luật nào mô tả hành vi của GA ở mức vĩ mô? Hay cụ thể hơn, là liệu có bất kỳ sự phán đoán nào về sự thay đổi của độ thích nghi của các nhóm con trong quần thể theo thời gian?
5. Có cách nào để mô tả các hiệu ứng khác nhau của các toán tử di truyền như lai ghép, đột biến, đảo ngược, v.v…
6. Trong những trường hợp nào (bài toán nào, toán tử di truyền nào) thì GA sẽ thực hiện tốt hơn các phương pháp nghiên cứu của TTNT truyền thống.

Những câu hỏi này (và còn nhiều hơn nữa) vẫn đã và đang được các nhà khoa học như Holland, Mitchell, Golderg,… nghiên cứu.

TỔNG KẾT PHẦN 3

Nội dung chính của chương này bao gồm:

- Giới thiệu tổng quát về một nhánh nghiên cứu mới của Trí Tuệ Nhân Tạo, đó là Học máy. Học được định nghĩa như là bất cứ sự thay đổi nào trong một hệ thống cho phép nó tiến hành tốt hơn trong lần thứ hai khi lặp lại cùng một nhiệm vụ hoặc với một nhiệm vụ khác rút ra từ cùng một quần thể các nhiệm vụ đó.
- Có ba tiếp cận học: Tiếp cận thứ nhất là tiếp cận ký hiệu, hai là tiếp cận mạng neuron hay kết nối và tiếp cận thứ ba là tiếp cận nổi trội hay di truyền và tiến hóa.
- Các chương trình học theo tiếp cận ký hiệu sẽ biểu diễn vấn đề dưới dạng các ký hiệu. Chương này trình bày một giải thuật được sử dụng rộng rãi của tiếp cận này, đó là ID3. ID3 sẽ học từ tập dữ liệu rèn luyện bao gồm rất nhiều ví dụ, mỗi ví dụ bao gồm một tập các cặp 'thuộc tính – giá trị'. Thuộc tính và giá trị ở đây là các ký hiệu. Sau khi học xong, ID3 biểu diễn khái niệm học được bằng một cây quyết định.
- Tiếp cận kết nối hay mạng neuron mô phỏng hệ thần kinh của con người để học được các khái niệm mà không sử dụng ký hiệu để biểu diễn vấn đề. Mạng đơn tầng perceptron cho thấy sức mạnh của mạng neuron, tuy nhiên khả năng áp dụng của chúng chỉ hạn chế cho các bài toán có tính tách rời tuyến tính. Mạng đa tầng áp dụng giải thuật học lan truyền ngược đã vượt qua những hạn chế của mạng perceptron, chứng tỏ được sức mạnh thực sự của tiếp cận này.
- Tương tự như tiếp cận kết nối, tiếp cận di truyền và tiến hóa có cảm hứng bắt nguồn từ tri thức của con người về sự tiến hóa của sinh vật: chỉ có những cá thể có khả năng thích nghi với sự thay đổi của môi trường thì mới tồn tại và phát triển. Thuật toán di truyền mô phỏng theo nguyên lý đó.

Bài tập

1. Cho một tập hợp các ví dụ rèn luyện như sau:

STT	Phân loại	A_1	A_2	A_3
1	+	T	T	F
2	+	T	F	T
3	–	F	T	T
4	–	F	F	T
5	+	F	T	F
6	+	F	F	F

An muốn áp dụng giải thuật ID3 để xây dựng cây quyết định với tập dữ liệu rèn luyện trên. Áp dụng các công thức tính entropy và gain, hãy giúp An xác định thuộc tính nào (A_1, A_2, hay A_3) là thuộc tính tốt nhất để hỏi đầu tiên nhằm tạo ra một cây quyết định đơn giản nhất. (Lưu ý: phải trình bày các tính toán entropy và gain để đi đến kết luận).

2. Cho một tập hợp gồm 10 ví dụ rèn luyện như sau:

STT	Cuối-tuần (A1)	Đang-đói (A2)	TG-chờ (phút) (A3)	Sẽ-chờ-bàn
1	Đúng	Có	0-10	***Có***
2	Sai	Có	>30	***Không***
3	Sai	Không	0-10	***Có***
4	Đúng	Có	10-30	***Có***
5	Đúng	Không	10-30	***Không***
6	Sai	Có	0-10	***Có***
7	Sai	Không	10-30	***không***
8	Sai	Có	10-30	***Có***
9	Đúng	Không	>30	***Không***
10	Đúng	Có	>30	***Không***

Tập dữ liệu trên thể hiện quyết định sẽ chờ bàn hay không của một người khi bước vào một nhà hàng đông khách không còn bàn trống. Quyết định của anh ta sẽ phụ thuộc vào một số yếu tố như hôm đó có phải là ngày cuối tuần không (cuối-tuần) – A1, anh ta có đang đói không (đang-đói) – A2, thời gian chờ bàn (TG-chờ) – A3: dưới 10 phút (0-10), từ 10 đến 30 phút (10-30) hay trên 30 phút (>30).

Áp dụng các công thức tính **entropy** và **gain**, để xác định thuộc tính tốt nhất để hỏi kế tiếp nhằm tạo ra một cây quyết định đơn giản nhất theo giải thuật ID3. Trình bày các tính toán entropy và gain ở mỗi bước.

Tài liệu tham khảo

- **Đinh Mạnh Tường.** ***Trí tuệ nhân tạo*****. Nhà xuất bản Khoa học kỹ thuật, 2002**
- **Stuart Russell, Peter Norvig.** ***Artificial Intelligence: A modern Approach*****, Prentice- Hall, 2002**
- **Chin-Liang Chang, Richard Char-Tung Lee,** ***Symbolic Logic and Mechanical Theorem Proving*****, Academic Press, Inc, 1973**
- **Enn Tyugu.** ***Algorithms and Architectures of Artificial Intelligence,*** **IOS Press 2007**
- **Nguyễn Thanh Thuỷ.** ***Trí tuệ nhân tạo: Các phương pháp giải quyết vấn đề và xử lý tri thức*****. Nhà xuất bản Giáo dục, 1999**

Giáo trình Nhập môn trí tuệ nhân tạo

TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN –TIN

Trương Chí Tín

GIÁO TRÌNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Đà Lạt, 04 - 2009

LỜI MỞ ĐẦU

Giáo trình "Nhập môn Trí tuệ nhân tạo" được viết dành cho sinh viên ngành Toán – Tin, Tin học và Công nghệ thông tin.

Để đọc giáo trình này, sinh viên cần có kiến thức cơ bản về lôgic, cấu trúc dữ liệu và thuật toán. Nội dung giáo trình này gồm 4 chương:

Chương 1: Khái niệm về trí tuệ nhân tạo
Chương 2: Các phương pháp giải quyết vấn đề
Chương 3: Biểu diễn và xử lý tri thức
Chương 4: Lập trình lôgic

Chương 1 giới thiệu tóm tắt lịch sử hình thành và phát triển cũng như các khái niệm chung nhất, các lĩnh vực nghiên cứu và ứng dụng chính của trí tuệ nhân tạo. Chương 2 trình bày các phương pháp biểu diễn và giải quyết vấn đề cơ bản: biểu diễn vấn đề trong không gian trạng thái bằng đồ thị thông thường, đồ thị VÀ/HOẶC, các phương pháp xác định trực tiếp lời giải, các phương pháp thử – sai (trong đó trình bày các phương pháp tìm kiếm theo chiều rộng, chiều sâu, theo hướng cực tiểu giá thành trên cây và đồ thị, thuật giải di truyền, phương pháp GPS, …) và các kỹ thuật heuristic. Chương 3 đề cập đến các phương pháp biểu diễn tri thức bằng: lôgic, luật sinh, mạng ngữ nghĩa, khung và các phương pháp xử lý tri thức bằng suy diễn dựa trên lôgic tất định và bất định. Chương 4 giới thiệu kỹ thuật lập trình lôgic thông qua ngôn ngữ lập tình Prolog.

Cuối mỗi chương có phần bài tập nhằm củng cố chắc hơn kiến thức lý thuyết và rèn luyện kỹ năng thực hành cho học viên. Các phần được in chữ nhỏ dành cho học viên đọc thêm.

Chắc chắn tài liệu này không tránh khỏi sơ suất, tác giả rất mong nhận được và chân thành biết ơn các ý kiến đóng góp quí báu của các bạn đồng nghiệp và độc giả nhằm làm cho giáo trình hoàn chỉnh hơn trong lần tái bản sau.

Đà lạt, 04 - 2009
Tác giả

MỤC LỤC

***CHƯƠNG IV*. LẬP TRÌNH LÔGIC**

Chương I

KHÁI NIỆM VỀ TRÍ TUỆ NHÂN TẠO

I.1. Lược sử hình thành và phát triển

* Trí tuệ nhân tạo (TTNT hay AI – Artificial Intelligence) là một trong những ngành mới trong lĩnh vực công nghệ thông tin. Có *nhiều quan điểm về trí tuệ nhân tạo*.

- Năm 1950, *Alan Turing đã đưa ra các "trắc nghiệm thông minh"* để nhận biết máy tính có thông minh hay không. Tuy vậy, cũng theo ông ta, tuy máy tính có thể thất bại trong các trắc nghiệm thông minh nhưng nó vẫn có thể thông minh.

- Theo quan điểm của *Minsky*, trí tuệ nhân tạo là một ngành khoa học nhằm *nghiên cứu, mô phỏng trên máy tính các hành vi và tư duy thông minh tương tự như con người*. Nó giúp máy tính có khả năng nhận thức, suy luận và phản ứng. Có *hai hướng tiếp cận trí tuệ nhân tạo*: dùng máy tính để bắt chước quá trình xử lý của con người và thiết kế những máy tính thông minh độc lập với cách suy nghĩ của con người.

- *Từ điển bách khoa toàn thư Webster* thì định nghĩa: "Trí tuệ là khả năng:
  1. *Phản ứng một cách thích hợp với những tình huống mới* thông qua hiệu chỉnh hành vi một cách thích đáng;
  2. *Hiểu rõ những mối liên hệ qua lại giữa các sự kiện* của thế giới bên ngoài nhằm đưa ra những hành động phù hợp để đạt tới một mục đích nào đó".

- Theo những *nhà tâm lý học nhận thức* thì *quá trình hoạt động trí tuệ của con người bao gồm 4 thao tác cơ bản:*
  1. Xác định tập đích (goal) cần đạt tới;
  2. Thu thập các sự kiện (facts) và các luật suy diễn (inference rules) để đạt tới tập đích đặt ra;
  3. Thu gọn (prunning) quá trình suy luận nhằm xác định một cách nhanh chóng tập các luật suy diễn có thể sử dụng được để đạt tới một đích trung gian nào đó;
  4. Áp dụng các cơ chế suy diễn (tiến hoặc lùi) cụ thể (inference mechanisms), dựa trên các thao tác thu gọn quá trình suy luận và những sự kiện trung gian mới được tạo ra, để dẫn dắt từ những sự kiện ban đầu đến những đích đã đặt ra.

** TTNT ra đời* dựa trên các thành quả của các ngành *tâm lý học nhận thức, lôgic hình thức*, … *Từ trên 2000 năm trước*, các nhà *triết học và tâm lý*

học đã cố gắng tìm hiểu *cách thức, cơ chế của quá trình nhớ, học tập, nhận thức và suy lý*.

- Vào đầu những *năm 50* của thế kỷ XX, nhờ sự *ra đời và cải tiến liên tục về hiệu suất hoạt động của máy tính,* đã xuất hiện xu hướng không chỉ nghiên cứu trí tuệ về mặt lý thuyết mà còn kiểm nghiệm các kết quả lý thuyết thông minh đó trên máy tính. Trong thời gian đầu mới hình thành, *nhiều công trình lý thuyết về TTNT vẫn chưa được kiểm nghiệm và triển khai trên thực tế* do chưa có ngôn ngữ lập trình đặc trưng cho TTNT, do hạn chế về kỹ thuật máy tính, giới hạn về bộ nhớ đặc biệt là tốc độ thực hiện và do vấn đề bùng nổ tổ hợp nảy sinh trong những thuật toán tìm kiếm lời giải cho các bài toán khó trong TTNT.

- Dựa trên các *thành quả về kỹ thuật phần cứng*, cùng với sự xuất hiện các ngôn ngữ lập trình đặc thù cho TTNT, *chuyên xử lý ký hiệu hình thức phục vụ cho lập trình lôgic* như IPL.V, *LISP* (viết tắt của LISt Processing, do Mc Cathy tại đại học MIT đề xuất năm 1960), PLANNER, *PROLOG* (viết tắt của PROgramming in LOGic, do Alain Colmerauer và nhóm công sự của ông tại đại học Marseilles xây dựng năm 1972), *nhiều giả thuyết hay kết quả thú vị về lý thuyết trong TTNT có điều kiện được kiểm nghiệm và trở thành các sản phẩm tin học cụ thể trên thị trường mang tính thông minh, hoạt động như các nhóm chuyên gia* nhiều kinh nghiệm trong từng lĩnh vực hẹp nào đó như y học, địa chất, dạy – học, chơi cờ,... Chẳng hạn các sản phẩm, chương trình: dẫn xuất kết luận trong hệ hình thức, chứng minh các định lý hình học phẳng, tính tích phân bất định, giải phương trình đại số sơ cấp, chơi cờ (Samuel), phân tích và chữa bệnh tâm lý (ELIZA), chuyên gia về y khoa (MYCIN ở đại học Stanford), phân tích và tổng hợp tiếng nói, điều khiển Robot theo đồ án "Mắt - tay", thăm dò khoáng sản (PROSPECTOR)...

  Khi sử dụng những sản phẩm chuyên dụng thông minh này, đặc biệt là lần đầu tiên, ta không khỏi ngạc nhiên về tính "thông minh" đến mức đôi khi ta có cảm giác chúng vượt trội hẳn khả năng của những người không chuyên nghiên cứu về lĩnh vực đặc thù đó.

- Trong những năm *1990*, ngành TTNT càng phát triển mạnh hơn nữa theo các hướng: *cơ sở tri thức và hệ chuyên gia, xử lý ngôn ngữ tự nhiên, lý thuyết nhận dạng hình ảnh, tiếng nói* và ứng dụng vào các kỹ thuật đa phương tiện, siêu văn bản, *mạng nơron, máy học, lý thuyết mờ trong lập luận xấp xỉ, lập trình tiến hoá, khai thác tri thức từ dữ liệu, ...*

* Có *vài dấu hiệu quan trọng của trí tuệ máy là các khả năng*: *học; mô phỏng các hành vi sáng tạo của con người; trừu tượng hóa, tổng quát hóa và suy diễn; tự giải thích hành vi; thích nghi với tình huống mới gồm khả năng thu*

nạp dữ liệu tích hợp, rút tri thức từ dữ liệu; *xử lý các biểu diễn hình thức (các ký hiệu tượng trưng, danh sách); vận dụng các tri thức heuristics sẵn có; xử lý các thông tin bất định, không đầy đủ, không chính xác, ...Trí tuệ máy khác trí tuệ người* ở chỗ nó *không thể nhìn trước được một phần hay toàn thể quá trình giải trong những tình huống mới* và *không tự sinh ra được các heuristics của chính bản thân chúng.*

* TTNT gồm *các phương pháp và kỹ thuật cơ bản* sau: phương pháp biểu diễn và giải quyết vấn đề; kỹ thuật heuristics; phương pháp biểu diễn và xử lý tri thức; phương pháp học và nhận dạng, xử lý ngôn ngữ tự nhiên và các ngôn ngữ lập trình cho TTNT. TTNT vẫn kế thừa các kỹ thuật cơ bản của tin học truyền thống như: xử lý danh sách, kỹ thuật đệ qui và quay lui, cú pháp hình thức, ... Trong bất kỳ một hệ thống TTNT nào cũng đều có *2 thành phần cơ bản liên quan mật thiết với nhau*: các phương pháp biểu diễn vấn đề và tri thức, các phương pháp tìm kiếm trong không gian bài toán, các chiến lược thu hẹp không gian lời giải và suy diễn.

I.2. Những lĩnh vực nghiên cứu của trí tuệ nhân tạo

I.2.1. Từ thuật toán đến thuật giải

* Đặc trưng của *thuật toán (Algorithm)*: yêu cầu *thỏa mãn nghiêm ngặt 3 tính chất: xác định, hữu hạn, đúng đắn. Ưu điểm*: những bài toán giải được bằng thuật toán có độ phức tạp không quá đa thức được áp dụng tốt trong thực tế. *Nhược điểm*: những thuật toán có phức tạp trên đa thức chỉ được áp dụng với không gian bài toán nhỏ; trên thực tế, lớp các bài toán khó chưa có thuật toán giải hoặc chưa biết được thuật toán giải hiệu quả rộng hơn rất nhiều.

Một hướng để giải quyết khó khăn đó là *mở rộng tính xác định, tính đúng* và *đưa vào thêm các thông tin đặc trưng về bài toán*, đưa vào máy tính một kiểu kinh nghiệm và "tư duy" của con người là sự *ước lượng*, để thu được các thuật giải heuristic.

* Đặc trưng của *thuật giải heuristic*: *độ phức tạp bé*, cho phép nhanh chóng *tìm ra các lời giải*, nhưng không phải luôn luôn tìm ra mà *có thể tìm thấy lời giải chỉ trong đa số trường hợp*; vả lại, các lời giải này *chưa chắc luôn đúng* hay *tối ưu* mà thường *gần đúng* hay *gần tối ưu*.

I.2.2. Phân loại các phương pháp giải quyết vấn đề

* *Biểu diễn vấn đề*: Xét vấn đề: A $\rightarrow$ B.

- Dạng *chuẩn*: Cho A, B tìm $\rightarrow$ (hai loại thuật toán, chương trình: thuật toán cần xác định trước chính là $\rightarrow$ và thuật toán tổng quát để tìm ra $\rightarrow$).
- Cho A, $\rightarrow$, tìm B (suy diễn *tiến*).
- Cho B, $\rightarrow$, tìm A (suy diễn *lùi*).

* *Nhóm các phương pháp* ***xác định trực tiếp lời giải***: phương pháp chính xác, phương pháp xấp xỉ gần đúng, phương pháp không tường minh, đệ qui, nguyên lý qui hoạch động

* *Nhóm các phương pháp* ***xác định gián tiếp lời giải*** *hoặc* ***tìm kiếm lời giải****:*

- *Phương pháp thử – sai*: vét cạn, nguyên lý mắt lưới, phương pháp nhánh cận, sinh và thử lời giải, phương pháp ngẫu nhiên (phương pháp Monte – Carlo, thuật giải di truyền GA), nguyên lý mê cung (dạng đệ qui), vét cạn dần (dưới dạng lặp) bằng cách quay lui và xác định dần thông tin về bài toán trong quá trình giải thông qua các cấu trúc không tuyến tính (chẳng hạn: cây, đồ thị hoặc đồ thị VÀ/HOẶC như các phương pháp tìm kiếm: theo chiều rộng, sâu, sâu dần, cực tiểu A^T), phương pháp GPS, …

- *Phương pháp heuristic trong trí tuệ nhân tạo*: là hướng tiếp cận quan trọng để xây dựng các hệ thống TTNT. Nó bao gồm các phương pháp và kỹ thuật tìm kiếm có sử dụng các tri thức đặc biệt từ chính bản thân lớp bài toán cần giải nhằm rút ngắn quá trình giải và nhanh chóng đi đến kết quả mong muốn mặc dù có thể không chắc chắn đó là cách giải quyết tối ưu nhưng lại có tính khả thi trong điều kiện thiết bị hiện có và thời gian yêu cầu. Trong kỹ thuật này người ta thường sử dụng kỹ thuật heuristis định lượng thông qua các hàm đánh giá. Chúng ta sẽ minh họa các phương pháp heuristics thông qua các phương pháp vét cạn thông minh (tìm kiếm tối ưu được bổ sung bằng tri thức đặc trưng về bài toán trên cây hoặc đồ thị tổng quát: A^{KT}, A*), nguyên lý tham lam, nguyên lý hướng đích (thuật giải leo núi), nguyên lý sắp thứ tự, nguyên lý khớp nhất, …

Những thông tin heuristic này vẫn *được gián tiếp đưa vào máy* tính thông qua con người. Vậy máy tính có thể *tự tạo ra các "tri thức", biết suy luận, chứng minh, tự học qua kinh nghiệm, máy có khả năng rút ra tri thức và vận dụng chúng vào việc giải quyết bài toán* hay không ?

Các *phương pháp trong trí tuệ nhân tạo* đã giúp máy tính thực hiện được trong một chừng mực nào đó các vấn đề đặt ra ở trên: các phương pháp biểu diễn

và xử lý tri thức, lập trình tiến hoá, mạng neuron nhân tạo, máy học, khai thác tri thức từ dữ liệu, …

I.2.3. Biểu diễn và xử lý tri thức

Có 4 phương pháp cơ bản biểu diễn và xử lý tri thức - dữ liệu tích hợp: phương pháp hình thức sử dụng cách tiếp cận logic (lôgic cổ điển - tất định: lôgic mệnh đề, lôgic vị từ; lôgic bất định: lôgic xác suất, lôgic khả xuất, lôgic mờ), các luật sinh (thường dùng trong các hệ chuyên gia), mạng ngữ nghĩa, bộ ba liên hợp OAV, cách biểu diễn bằng khung (hay dàn - Frame),... Các hệ chuyên gia là những thể hiện của việc kết hợp của các phương pháp biểu diễn và phương pháp xử lý tri thức (ví dụ: DENDRAL, MOLGEN, PROSPECTOR, MYCIN, ...).

I.2.4. Xử lý ngôn ngữ tự nhiên, các ngôn ngữ lập trình dựa trên việc xử lý danh sách, ký hiệu và lập trình logic: các ngôn ngữ lập trình LISP, PROLOG có hạn chế là chi phí lớn và khó phát triển hệ thống; còn CLIPS nhằm biểu diễn tri thức theo hướng đối tượng và xử lý các luật suy dẫn. Ta có thể thấy sự khác nhau cơ bản giữa lập trình truyền thống và lập trình xử lý ký hiệu trong TTNT qua bảng so sánh I.1.

Lập trình truyền thống	**Lập trình xử lý ký hiệu và lôgic**
- Xử lý dữ liệu	- Xử lý tri thức - dữ liệu tích hợp
- Dữ liệu trong bộ nhớ được đánh địa chỉ số	- Tri thức được cấu trúc trong bộ nhớ làm việc theo ký hiệu
- Xử lý theo các thuật toán	- Xử lý theo các thuật giải heuristics và cơ chế lập luận
- Định hướng xử lý các đại lượng định lượng số	- Định hướng xử lý các đại lượng định tính, logic, ký hiệu tượng trưng, danh sách
- Xử lý tuần tự hoặc theo lô	- Xử lý theo chế độ tương tác cao (hội thoại, theo ngôn ngữ tự nhiên,...)
- Không giải thích trong quá trình thực hiện	- Có thể tự giải thích hành vi hệ thống trong quá trình thực hiện

Bảng I.1

I.2.5. Lý thuyết nhận dạng *theo hướng thống kê, cấu trúc, đại số và heuristics* gồm: nhận dạng *hình ảnh và âm thanh* (HEARSAY-II, …).

I.2.6. Lập trình tiến hoá, mạng nơron, máy học, khai thác dữ liệu

- *Lập trình tiến hóa (Revolution Programming)* sử dụng ý tưởng *qui luật tiến hoá và học thuyết di truyền* của ngành sinh học: những gì hợp lý, *thích*

nghi tốt với môi trường sẽ có khả năng tồn tại lâu dài hơn trong quá trình đào thải, sinh tồn; một số đặc điểm của thế hệ trước sẽ di truyền, ảnh hưởng đến thế hệ sau thông qua *lai chéo*; thỉnh thoảng vẫn xuất hiện vài cá thể có đặc điểm khác hẳn (hoặc nổi trội lại các đặc điểm tiềm tàng) với thế hệ trước của chúng thông qua *đột biến*, … Khởi điểm của hướng nghiên cứu này là thuật giải di truyền (*GA - Genetic Algorithm*). *Ưu điểm* của các thuật giải GA là *có thể áp dụng đối với các bài toán chưa biết thuật toán nào hay chưa có thuật giải nào khả dĩ, hiệu quả* để giải. Có thể xem GA thuộc vào lớp các *thuật toán ngẫu nhiên* thông qua việc tạo ngẫu nhiên quần thể ban đầu cũng như các vị trí lai chéo hay tỉ lệ lai chéo và đột biến của chúng.

- *Mạng nơron nhân tạo* (hay vắn tắt hơn là mạng nơron, *ANN - Artificial Neural Networks)* mô phỏng mô hình và cơ chế hoạt động hưng phấn và ức chế thần kinh để điều khiển hoạt động của con người. Mô hình ANN có thể gồm nhiều lớp, trong đó ít nhất phải có lớp nhập (input layer) và lớp xuất (output layer), ngoài ra có thể có nhiều lớp ẩn (hidden layers) trung gian. Mỗi lớp gồm nhiều nút. Các kích thích từ môi trường ngoài được truyền vào mạng thông qua các nút của lớp nhập. Tổng hợp các kích thích này (phụ thuộc các trọng số mà mạng này cần học), nếu vượt quá một ngưỡng nào đó, sẽ gây kích thích (hưng phấn hay ức chế) đến các nút của lớp kế tiếp. Cứ thế, quá trình tiếp tục lan truyền đến lớp xuất. Một mô hình ANN thường được ứng dụng nhiều trong thực tế là mạng nơron lan truyền ngược. Lặp lại quá trình này, dựa trên việc cập nhật trọng số qua mỗi thế hệ sao cho giảm dần sai số giữa giá trị thật và giá trị do mạng kết xuất. Qua một số thế hệ luyện, mạng sẽ học được bộ trọng số thích hợp. *Ưu điểm* của các phương pháp luyện mạng ANN là *có thể áp dụng đối với các bài toán chưa biết thuật toán nào hay chưa có thuật giải nào khả dĩ, hiệu quả* để giải.

- *Máy học (LM - Learning Machine)* là quá trình rút ra qui luật từ dữ liệu, chẳng hạn học thông qua lôgic, học bằng quan sát dựa trên các độ đo phù hợp, học dựa trên cây định danh thông qua độ đo hỗn loạn thông tin trung bình, … Ta có thể dùng ANN để ứng dụng vào máy học.

- *Khai thác dữ liệu (DM - Data Mining)* nhằm rút ra tri thức từ dữ liệu thô, chẳng hạn đo độ phụ thuộc của một lớp các thuộc tính xác định vào lớp các thuộc tính phổ biến khác trong tập dữ liệu thô cho trước thông qua luật kết hợp, ...

I.3. Những ứng dụng của TTNT

- Điều khiển học, Robotic, giao diện người máy thông minh
- Trò chơi máy tính
- Thiết bị điện tử thông minh nhờ sử dụng lôgic mờ
- Hệ chuyên gia trong: giáo dục, y khoa, địa chất, quản lý, …
- Xử lý ngôn ngữ tự nhiên

- Nhận dạng hình ảnh, âm thanh, …
- Các hệ thống xử lý tri thức và dữ liệu tích hợp: cho phép xử lý đồng thời tri thức và dữ liệu (cơ sở dữ liệu suy diễn, biểu diễn luật đối tượng, hệ hỗ trợ quyết định)
- Mô hình hoá các giải pháp giải bài toán

Chương II
CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

II.1. Các phương pháp xác định trực tiếp lời giải

Đặc điểm của các phương pháp này là xác định *trực tiếp* được lời giải thông qua một *thủ tục tính toán* hoặc *các bước căn bản* để có được lời giải. Có ba loại phương pháp chính để xác định trực tiếp lời giải. Loại thứ nhất được áp dụng để giải các bài toán đã biết cách giải bằng các *công thức chính xác* (như công thức toán học). Loại thứ hai được dùng cho các bài toán đã biết cách giải bằng các *công thức xấp xỉ* (như các công thức xấp xỉ trong phương pháp tính). Loại thứ ba được áp dụng vào các bài toán đã biết *cách giải không tường minh thông qua các hệ thức truy hồi hay kỹ thuật đệ qui*.

II.1.1. Phương pháp giải chính xác: thông qua các công thức giải chính xác. Chẳng hạn, thuật toán giải phương trình bậc hai.

II.1.2. Phương pháp giải xấp xỉ: thông qua các công thức giải gần đúng. Chẳng hạn, phương pháp lặp tính tích phân xác định theo công thức hình thang trong học phần *"Phương pháp tính"*.

II.1.3. Phương pháp giải không tường minh: thông qua các hệ thức truy hồi hoặc kỹ thuật đệ qui.

* <u>*Thao tác đệ qui*</u> *F(x)* trên 1 đối tượng $x \in D$ nào đó. Xét hai trường hợp:

- Nếu đối tượng x thuộc một tập đặc biệt X_0 nào đó ($X_0 \subset D$) mà đã biết cách giải đơn giản thì thực hiện các thao tác sơ cấp tương ứng;

- Ngược lại, trước hết có thể thực hiện một thao tác G(x) nào đó, biến đổi x thành x'= H(x) $\in$ D rồi thực hiện thao tác tương tự F(x') trên x', sau đó có thể thực hiện thêm một thao tác K(x) nào đó trên x, sao cho sau một số hữu hạn bước này, các điểm $x^{n(*)}$ sẽ rơi vào tập X_0.

```
F(x) //x ∈ D
{
  if (x ∈ X0) ThaoTácSơCấp(x);    // điều kiện dừng
  else { G(x);
         x' = H(x);                //H(x) ∈ D
```

```
        F(x');                    // lời gọi đệ qui
        K(x);
      }
}
```

Các thao tác đệ qui thường gặp trong tin học là: *định nghĩa đệ qui, hàm hoặc thủ tục đệ qui, thuật toán đệ qui.*

* <u>*Chú ý*</u>: Khi thiết kế một thao tác đệ qui, ta cần có hai phần:

- *Phần cơ sở* (phần neo hay *điều kiện dừng*): thao tác sơ cấp đã biết cách thực hiện ngay trên tập con $X_0 \subset D$.
- *Phần gọi đệ qui* F(x'): cần phải bảo đảm *sau một số hữu hạn bước biến đổi x thì ta sẽ gặp điều kiện dừng: $H(H(\ldots H(x))) = x_0 \in X_0$.*
- Trên đây, ta xét đệ qui đuôi trực tiếp. Các trường hợp phức tạp hơn một chút như đệ qui nhánh trực tiếp và đệ qui gián tiếp (hay đệ qui hỗ tương) được xét tương tự.

* <u>*Ví dụ 1a*</u> (dãy số *Fibonacci*): Ở đầu tháng thứ 1 có 1 cặp thỏ con mới ra đời (F(0) = 0, F(1) = 1). Giả sử:
- Cứ sau mỗi tháng một cặp thỏ (từ sau hai tháng tuổi) sẽ sinh thêm một cặp thỏ con;
- Các con thỏ không bao giờ chết.

Hỏi số cặp thỏ F(n) sau n tháng là bao nhiêu?

Ta có công thức truy hồi để tính F(n) như sau:

$F(0) = 0; F(1) = 1$ ($X_0 = \{0; 1\}$)

$F(n) = F(n-1) + F(n-2), \forall n \geq 2$

Để tính F(n), ta có thể thực hiện theo các cách sau:

- *Thuật toán đệ qui* sau đây có độ phức tạp thuật toán với số phép cộng là $O(F(n)) = O(((1+\sqrt{5})/2)^n)$: độ phức tạp mũ, quá lớn, không khả thi !

```
Nguyên Fibonaci_De_Qui(n)
{       if (n ≤ 1) return n;
        else return (Fibonaci_De_Qui(n-1) + Fibonaci_De_Qui(n-2));
}
```

- *Thuật toán lặp* sau đây có độ phức tạp thuật toán với số phép cộng là O(n): hiệu quả hơn nhiều ! Ta có thể *khử đệ qui* bằng cách *dùng vòng lặp và vài biến phụ hoặc dùng cơ chế ngăn xếp*.

```
Nguyên Fibonacci_Lặp(n)
{  if (n==0 or n==1) return n;
   else { j = 1;
          Truoc = 0;
          HienTai = 1;
          while (j < n) do
          {     Sau = Truoc + HienTai;
                Truoc = HienTai;
```

```
                HienTai = Sau;
                j = j + 1;
            }
            return Sau;
        }
}
```

- *Tìm công thức tường minh từ hệ thức truy hồi*

$$F(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n \right]$$

* Phương pháp tổng quát tìm công thức tường minh cho F_n từ hệ thức truy hồi tuyến tính (hệ số hằng):

$\boldsymbol{F_n = b_1 F_{n-1} + b_2 F_{n-2}}$, $\forall n \geq 2$ với các trị $\{F_0, F_1\}$ cho trước.

Gọi $\{\Phi_1, \Phi_2\}$ là 2 nghiệm của *đa thức đặc trưng* tương ứng:

$$\Phi^2 - b_1\Phi - b_2 = 0.$$

Khi đó: $\mathbf{F_n = c_1 \Phi_1^n + c_2 \Phi_2^n}$, (c_1, c_2 sẽ được xác định từ các điều kiện đầu của F_0, F_1).

* <u>*Nhận xét*</u>: Trong nhiều bài toán khó, ta có thể dùng chiến lược "***Chia để trị***" để *tách nó thành nhiều bài toán con có cùng cách giải* như bài toán ban đầu thông qua *kỹ thuật đệ qui*.

- <u>*Ví dụ 1b*</u>: Tráo đổi hai phần *a[1..k]* và *a[k+1..n]* của mảng *a* gồm *n* phần tử (không nhất thiết có độ dài bằng nhau) mà không dùng mảng phụ.

Nếu k ≤ n-k thì trước tiên trao đổi hai phần bằng nhau a[1..k] và a[n-k..n]; sau đó trong mảng con a[1..n-k], ta chỉ cần tráo đổi *k* phần tử đầu với phần còn lại. Trường hợp k ≥ n-k, giải tương tự, ...

```
TraoHaiPhanBangNhau(a, Tu1, Tu2, SoPTuTrao)
{  for (i=1; i ≤ SoPTuTrao; i++)
        DoiCho(a[Tu1+i], a[Tu2+i]);
}

TraoHaiPhanBatKy(a, n, k) // k >= 0
{ i = 1; j = n;
  while (k >= i)
  { if (k < (i+j)/2)
        { TraoHaiPhanBangNhau(a, i, i+j-k, k-i+1);
          j = i + j - k - 1;
        }
    else { TraoHaiPhanBangNhau(a, i, k+1, j-k);
          i = i + j - k;
        }
  }
}
```

II.1.4. Phương pháp qui hoạch động:

* *Ý tưởng của nguyên lý qui hoạch động*: nghiệm của một bài toán con (của một bài toán) là sự kết hợp các nghiệm của các bài toán con nhỏ hơn của nó (trong trường hợp rời rạc có tính đệ qui thì *nghiệm trong n bước sẽ có được từ lời giải của k bước trước và lời giải trong n-k bước)*. Ta thường dùng phương pháp này để giải các bài toán tối ưu mà thỏa mãn nguyên lý trên.

Có thể xem nguyên lý tối ưu là một sự thể hiện tốt của phương pháp chia để trị trong việc giải quyết vấn đề. Khi thực hiện các tính toán trong phương pháp qui hoạch động, để thực hiện tính toán tại bước thứ n, nên *tận dụng các kết quả đã tính ở các bước trước* thông qua các hệ thức truy hồi và một vài biến phụ để lưu các kết quả trung gian trước đó (chẳng hạn, xét bài tập: tính tất cả các số tổ hợp C^k_n, với mọi k: $0 \le k \le n$).

* *Mô hình toán học của nguyên lý tối ưu*

- *Định nghĩa II.1 (hàm phân tích được):* Cho hàm f: $D \to R$, $D \subset R^n$,
$D_1 = \{x_1 \in R^1: \exists y \in R^{n-1}$ & $(x_1,y) \in D\}$, $D(x_1) = \{y \in R^{n-1} : (x_1, y) \in D\}$ $\forall x_1 \in D_1$.
Ta nói hàm f là *phân tích được* nếu tồn tại hai hàm g: $R^2 \to R$ và h: $R^{n-1} \to R^1$ sao cho:

. $f(x_1, y) = g(x_1, h(y))$, $\forall x = (x_1, y) \in D$, $x_1 \in D_1$, $y \in D(x_1)$

. Hàm *g đơn điệu không giảm theo biến thứ hai*:

$\forall x_1 \in D_1$, $\forall z_1, z_2 \in R^1$: $z_1 \ge z_2 \Rightarrow g(x_1, z_1) \ge g(x_1, z_2)$

(thật ra chỉ cần: $\forall x_1 \in D_1$, $\forall z_1, z_2 \in R^1$, $\exists y_1, y_2 \in D(x_1)$: $z_1 = h(y_1)$, $z_2 = h(y_2)$,
$z_1 \ge z_2 \Rightarrow g(x_1, z_1) \ge g(x_1, z_2)$)

- *Mệnh đề II.1:* Cho f là hàm phân tích được (trong định nghĩa II.1). Khi đó, ta có:

$$\underset{x \in D}{opt}\, f(x) = \underset{x_1 \in D_1}{opt}[g(x_1, \underset{y \in D(x_1)}{opt}[h(y)])]$$

- *Nhận xét:*

. Kết quả của mệnh đề trên cho phép ta *đưa việc tối ưu hàm nhiều biến về tối ưu các hàm theo các biến thành phần có số chiều bé hơn.*

. Ta thường gặp trường hợp *hàm g có dạng tuyến tính theo biến thứ hai:*

$$g(x_1, z) = r(x_1) + z$$

và *f có dạng cộng tính theo từng thành phần:*

$$f(x_1, x_2, \ldots, x_n) = r(x_1) + r(x_2) + \ldots + r(x_n)$$

* *Ví dụ 2* (bài toán người giao hàng *Salesman*): Hàng ngày, người giao hàng phải chuyển hàng qua n địa điểm, mỗi địa điểm đúng một lần, rồi quay lại địa điểm xuất phát. Bài toán đặt ra là: làm thế nào để anh ta có được một hành trình với đường đi ngắn nhất.

Ta biểu diễn bài toán bằng đồ thị định hướng G = (V, A), với V={1, 2, …, n} và độ dài cung C(i,j) > 0, nếu (i,j) ∈ A và C(i,j) = ∞, nếu (i,j) ∉ A. Không mất tính tổng quát, ta có thể giả sử đường đi của anh ta xuất phát từ đỉnh 1. Bất kỳ đường đi nào (chấp nhận được) của người giao hàng cũng có thể phân thành: cung (1,k) với k ∈ V\{1} và đường đi từ k tới 1 qua mỗi đỉnh thuộc V\{1} đúng một lần. Nếu đường đi của anh ta ngắn nhất thì đường đi từ k tới đỉnh 1 qua các đỉnh thuộc V\{1, k} phải ngắn nhất. Do đó

nguyên lý tối ưu được thỏa mãn. Gọi d(j, S) là độ dài đường đi ngắn nhất từ j đến đỉnh 1 qua mỗi đỉnh k ∈ S (∀S ⊂ V và S ≠ ∅) đúng một lần. Ta có công thức truy hồi:

Nghiệm tối ưu cần tìm là:

Rõ ràng, d(j, ∅) = C(j,1), ∀ j ∈ [2, n]. Từ công thức truy hồi trên, ta tính được d(j, S) với mọi S chỉ chứa 1 đỉnh. Từ đó, ta tính được d(j, S) với mọi S chỉ chứa 2 đỉnh, ... Cứ thế tiếp tục, ta tính được d(k, S) với mọi S = V\{1,k}, ∀ k ∈ [2, n]. Từ đó, ta tìm được

$$d(1, V \setminus \{1\}) = \min_{2 \le k \le n}(C(1,k) + d(k, V \setminus \{1,k\}))$$

$$d(j, S) = \min_{k \in S}(C(j,k) + d(k, S \setminus \{k\}))$$

nghiệm tối ưu.

- Cụ thể, xét đồ thị định hướng có 4 đỉnh và độ dài các cung được cho bởi ma trận C như sau:

$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

Ta có: d(2, ∅) = C(2,1) = 5
d(3, ∅) = C(3,1) = 6
d(4, ∅) = C(4,1) = 8

Từ công thức truy hồi, ta tính được:

d(2, {3}) = C(2,3) + d(3, ∅) = 15
d(2, {4}) = C(2,4) + d(4, ∅) = 18
d(3, {2}) = C(3,2) + d(2, ∅) = 18
d(3, {4}) = C(3,4) + d(4, ∅) = 20
d(4, {2}) = C(4,2) + d(2, ∅) = 13
d(4, {3}) = C(4,3) + d(3, ∅) = 15

Tương tự:

d(2, {3, 4}) = min{C(2,3) + d(3, {4}), C(2,4) + d(4, {3})}
= min {29, 25} = 25

d(3, {2, 4}) = min{C(3,2) + d(2, {4}), C(3,4) + d(4, {2})}
= min {31, 25} = 25

d(4, {2, 3}) = min{C(4,2) + d(2, {3}), C(4,3) + d(3, {2})}
= min {23, 27} = 23

Cuối cùng, ta có:

d(1, {2, 3, 4}) = min{ C(1,2) + d(2, {3, 4}), C(1,3) + d(3, {2, 4}),
C(1,4) + d(4, {2, 3})}
= min {35, 40, 43} = 35

Để tìm được hành trình ngắn nhất, gọi K(j,S) là đỉnh k, tại đó nó đạt min của công thức truy hồi để tính d(j,S). Từ đó, ta có:

K(1, {2, 3, 4}) = 2
K(2, {3, 4}) = 4
K(4, {3}) = 3

Vậy đường đi {1, 2, 4, 3, 1} là ngắn nhất và đạt giá trị 35.

Tương tự, có thể áp dụng nguyên lý qui hoạch động để giải bài toán sau.

- *Ví dụ 3* (*bài toán sắp ba lô)*: một chiếc ba lô có thể chứa được một khối lượng w. Có n loại đồ vật được đánh số 1, 2, …, n. Mỗi đồ vật loại i có khối lượng a_i và có giá trị c_i (các trị w, a_i, c_i đều nguyên dương, i = 1, 2, …, n). Cần sắp xếp các đồ vật vào ba lô để ba lô có giá trị lớn nhất có thể được. Giả sử rằng mỗi loại đồ vật có đủ nhiều để xếp (*bài tập*).

II.2. Các phương pháp thử - sai

II.2.1. Phương pháp vét cạn, nguyên lý mắt lưới, phương pháp sinh và thử, phương pháp nhánh cận

**** Bài toán 1**: Tìm tập LờiGiải = {x ∈ D: tính chất P(x) đúng}.* ***(BT1)***

a. Phương pháp vét cạn

* *Thuật toán vét cạn V1.1*: giải BT1

```
{   B1: LờiGiải = ∅;
    B2: Trong khi D ≠ ∅ thực hiện:
            x ← get(D);  //lấy khỏi D một phần tử x
            if (P(x)) LờiGiải = LờiGiải ∪ {x};
    B3: if (LờiGiải == ∅) write("Không có lời giải");
        else XuấtLờiGiải(LờiGiải);
}
```

* *Chú ý*: - Nếu hạn chế miền D càng bé thì thuật toán chạy càng nhanh.

* *Ví dụ 4*: Cho trước các số M, K nguyên dương. Tìm các bộ số nguyên dương x, y, z sao cho:

Thay vì chọn D = {(x,y,z) ∈ $\mathbf{N}^3$ ∩ $[1;M-2]^3$} (**N** là tập các số tự nhiên), ta lấy: D = $\begin{cases} x+y+z=M \\ x^3+y^3+z^3=K \end{cases}$

{(x,y,z) ∈ $\mathbf{N}^3$ ∩ $[1; \min\{M-2; \sqrt[3]{K-2}\}]^3$}.

Khi lập trình, ta thể hiện miền D bởi 3 vòng for theo x, y, z:

```
Tìm_xyz(M, K)
{ Max = min(M-2, pow(K-2,1.0/3));
  for (x = 1; x ≤ Max; x++)
    for (y = 1; y ≤ Max; y++)
      for (z = 1; z ≤ Max; z++)
        if (x+y+z==M && pow(x,3)+ pow(y,3)+ pow(z,3) == K)
            XuấtLờiGiải(x,y,z);
}
```

Thật ra, vẫn có thể cải tiến chương trình trên để thu hẹp miền D hơn nữa (*bài tập*) !

- Dựa vào thuật toán trên, ta có thể sửa đổi chút ít để giải bài toán tìm kiếm chỉ một lời giải sau:

**** Bài toán 2**: Tìm một lời giải $x_0 \in D$: mệnh đề $P(x_0)$ đúng.* ***(BT2)***

Thuật toán tìm một lời giải V1.2: giải BT2

```
{   Trong khi D ≠ ∅ thực hiện:
    {   x ← get(D);  //lấy khỏi D một phần tử x
        if (P(x))
            { XuấtLờiGiải(x);
              Dừng;      // điểm chính khác với thuật toán V1
            }
    }
    write("Không có lời giải");
}
```

* Đối với một lớp bài toán nào đó mà có thể tìm được một *điều kiện đủ Q(x) cho lời giải x*, ta có thể dùng *thuật giải* sau: *với mỗi x ∈ D mà Q(x) đúng thì xuất lời giải*. Chú ý rằng, ngoài các lời giải trên, trong D còn *có thể chứa các lời giải khác mà không thỏa điều kiện đủ này* ! Nguyên lý mắt lưới sau đây là một thể hiện của ý tưởng trên.

b. Nguyên lý mắt lưới:

* *Ý tưởng*: Những con cá lớn hơn kích thước mắt lưới lớn nhất sẽ còn lại trong lưới !

Để giải bài toán (1), nếu chứng minh được:

> "nếu tìm được điều kiện Q(x) đúng với một vài x thuộc một miền con của D có kích thước nhỏ hơn ε thì P(y) đúng với mọi y ∈ miền con đó"

thì: Chia lưới D thành n miền con D_i (mỗi miền D_i có kích thước nhỏ hơn ε). Với mỗi i = 1 .. n, xét nếu tồn tại $x \in D_i$ mà Q(x) đúng thì P(x) đúng.

* *Ví dụ 5*: Tìm nghiệm gần đúng (với độ chính xác eps > 0) của phương trình f(x) = 0 trên miền [a, b], với f là hàm liên tục.

Ta đã biết nếu $f(x_i).f(x_{i+1}) < 0$, với $[x_i, x_{i+1}] \subset [a, b]$ và $abs(x_{i+1}-x_i) < eps$ thì $x_k = (x_{i+1}+x_i)/2$ sai khác với một nghiệm chính xác của phương trình f(x) = 0 không quá eps/2 (*một điều kiện đủ để tìm nghiệm* của phương trình liên tục).

- *Thuật giải tìm nghiệm gần đúng*:

```
Nghiem_Gan_Dung(a, b, eps)
{ Sai_so = 1e-3;
  n = (b-a)/eps;
  xi = a;
```

```
  for  i=1 to n do
  { xi_1 = xi + eps;
    if (f(xi)*f(xi_1) < - Sai_so)  writeln((xi + xi_1)/2);
    xi = xi_1;
  }
}
```

c. ***Phương pháp sinh và thử***

* *Ý tưởng: Sinh dữ liệu* (*cấu trúc của lời giải* có thể không xác định trước khi giải mà *chỉ được tạo ra dần* trong quá trình tìm kiếm lời giải*), sau đó kiểm tra nó có thỏa điều kiện dừng hay không ?*

- *Thuật toán sinh và thử*:

```
Sinh_Thử
{ Init;                              // khởi tạo dữ liệu xuất phát
  Stop = False;
  While not Stop do
  { if (Accept(C)) Show(C); // nếu phương án C là chấp nhận thì xuất C
    if (Generate(C)=False) Stop = True;
        // nếu không sinh được thêm phương án C nào nữa thì dừng
  }
}
```

* *Ví dụ 6*: bài toán chỉnh hợp lặp chập k của n phần tử $X = \{0,1, \ldots,n-1\}$: $\{x_1, x_2, \ldots, x_k\}$, với $x_i \in X$, $1 \le i \le k$.

Thủ tục *Init* sẽ khởi tạo 0 cho vectơ lời giải $x = \{x_1, x_2, \ldots, x_k\}$. Trong ví dụ này không cần đến điều kiện kiểm tra *Accept*: ta luôn cho nó nhận trị đúng. Thủ tục *Generate*(x, k) sẽ tăng dần x[j] một đơn vị, j bắt đầu từ k đến 1, cho đến khi x[j]=n-1 thì khởi động lại 0 cho x[j], rồi giảm j đi một. Khi nào j=0 thì dừng việc sinh dữ liệu.

```
Boolean Generate(x, k)
{ j = k;  // sinh chỉnh hợp kế tiếp
  while (j>0 and x[j] == n-1) do
  { x[j] = 0;
    j = j-1;
  }
   if (j==0) return False;
   else { x[j] = x[j]+1;
          return True;
        }
}
```

(Ngoài ra, ta có thể giải bài toán này bằng cách này sử dụng thuật toán *đệ qui TryRờiRạc*(i) trong II.2.3 với điều kiện kết thúc nghiệm là i = k).

Để các thuật toán vét cạn *không bị bùng nổ tổ hợp về thời gian và không gian* nhớ, ta cần: ***giảm độ phức tạp tính toán*** (*không tính lại các hằng trong vòng lặp, cần tận dụng lại các kết quả tính toán ở các bước trước, dùng kỹ thuật lính canh* để đơn giản các biểu thức điều kiện của vòng lặp, …); ***thu gọn không gian tìm kiếm***.

* ***Chiến lược thu hẹp không gian tìm kiếm***: Trong các thuật toán vét cạn để tìm kiếm lời giải trong không gian D, đối với một lớp các bài toán nào đó, dựa trên các *đánh giá toàn cục* (ví dụ: duyệt các bộ tổ hợp), *cục bộ* (ví dụ: bài toán sắp ba-lô), nếu ta *tìm ra được các* ***điều kiện cần*** *cho lời giải,* khi đó ta có thể *cải tiến thuật toán bằng cách loại bỏ ngay các phương án trong D không thỏa điều kiện cần này !* Khi đó:

. hoặc *xét tập D* (nếu có thể) *chỉ chứa những trạng thái thoả mãn điều kiện cần* cho lời giải mà thôi, do đó *D được thu hẹp ngay*;

. hoặc xét tập D như thông thường, nhưng trong các thuật toán vét cạn, ta *hiểu get(D) là lấy ra khỏi D phần tử x: nếu x không thỏa điều kiện cần cho lời giải thì loại ngay nó (và thực hiện việc này càng sớm tới mức có thể để tránh các thao tác thừa) rồi lấy ngay phần tử x tiếp theo của D,* ngược lại mới kiểm tra tính chất P(x).

Phương pháp nhánh cận là một thể hiện của chiến lược này.

d. Phương pháp nhánh cận

* *Ý tưởng*: *nhánh có chứa quả phải nặng hơn trọng lượng quả*. Khi xây dựng thêm thành phần cho lời giải, dùng các phép kiểm tra đơn giản để *xác định chi phí tối thiểu đến lời giải (điều kiện cần cho lời giải)*. Loại *bỏ ngay các hướng đi tiếp theo mà chi phí tối thiểu này còn lớn hơn cả chi phí thấp nhất hiện thời (hướng không thỏa điều kiện cần).*

* *Ví dụ 7* (Bài toán người du lịch): Có n thành phố (được đánh số từ 1 đến n). Một người du lịch xuất phát từ một thành phố, muốn đi thăm các thành phố khác, mỗi thành phố đúng một lần rồi lại quay về nơi xuất phát. Giả thiết giữa hai thành phố j, k khác nhau bất kỳ đều có đường đi với chi phí c(j,k). Hãy tìm một hành trình có tổng chi phí nhỏ nhất.

Một hành trình x[1], x[2], …, x[n] là một hoán vị của {1, 2, …, n}. Dùng mảng lôgic ChưaĐến để đánh dấu các thành phố đã đi qua: ChưaĐến[k] = True nghĩa là người du lịch chưa đến thành phố k. Nếu việc đến thành phố k có tổng chi phí dự đoán thấp nhất để hoàn thành toàn bộ hành trình lớn hơn chi phí thấp nhất hiện thời thì ta không chọn đi tiếp k, ngược lại thì chọn k.

Giả sử ta đã đi qua j-1 thành phố x[1], …, x[j-1] với chi phí là S. Nếu đi tiếp đến thành phố x[j] = k thì chi phí từ x[1] đến x[j] là T = S + c[x[j-1],k]. Đoạn còn lại của hành trình gồm (n-j+1) đoạn nữa, với chi phí trên mỗi đoạn không ít hơn Cmin (là chi phí trực tiếp thấp nhất giữa hai thành phố khác nhau trong ma trận chi phí). Tổng chi phí thấp nhất để hoàn thành hành trình là:

T + (n-j+1) * Cmin = S + c[x[j-1],k] + (n-j+1)*Cmin

Để giải bài toán này, ta sẽ gọi thủ tục chính *TryRờiRạc*(0, 2).

```
TryRờiRạc(S, j)
{ for  k =1 to n do
    if (ChưaĐến[k])
       { T = S + c[x[j-1],k];
         if (T + (n-j+1)*Cmin < Min)  // Min = MaxInt trước thủ tục Try
         { x[j] = k; ChưaĐến[k] = False;
           if   ( j == n)
                 { if (T + c[k, xp] < Min)        // xp là thành phố xuất phát
                     {  y = x;         // mảng y lưu trữ lời giải tốt nhất tạm thời
                        Min = T + c[k, xp];
                     }
                 }
           else Try(T, j+1);
           ChưaĐến[k] = True;
         }
       }
}
```

II.2.2. Phương pháp ngẫu nhiên

Phương pháp này ***<u>được sử dụng khi chưa biết thuật toán nào hiệu quả</u>*** hay *<u>chưa có nhiều thông tin để giải bài toán.</u>*

a. *<u>Phương pháp Monte – Carlo</u>*

* *<u>Ý tưởng</u>*: Cho hai hình $S \subset \Omega = [a, b]^m \subset R^m$, giả sử đã biết công thức tính độ đo S() của S phụ thuộc vào tham số nào đó cần tính. Tung ngẫu nhiên n lần (độc lập) vectơ $x = \{x_1, x_2, \ldots, x_m\} \in \Omega$ (có phân phối đều trên Ω), gọi n_S là số lần $x \in S$. Khi đó, với n đủ lớn, ta có: $S() \approx (b-a)^m.n_S/n$.
Từ đó rút ra tham số cần tính theo số liệu thực nghiệm n_S, n:

$$\approx S^{-1}((b-a)^m.n_S/n)$$

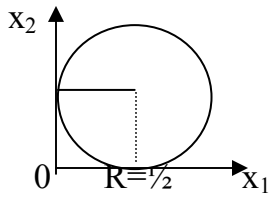
* *<u>Thuật toán ngẫu nhiên</u>*:

```
{     nS = 0;
      for  (j=1; j ≤ n; j++)
      { for  (k=1; k ≤ m; k++)
            x[k] = random(a,b); // tạo  các số ngẫu nhiên ∈ [a; b)
        if (x ∈ S) nS = nS + 1;
      }
      ≈ S^-1((b-a)^m.nS/n);
}
```

* *<u>Ví dụ 8</u>*: Dựa vào phương pháp trên, với m = 2, a = 0, b = 1 và S là hình tròn bán kính R=½, ta có thể tính số π như sau:

$$\pi = S/R^2 \approx 4*nS/n$$

$$x \in S <\text{---}> (x_1 - ½)^2 + (x_2 - ½)^2 \le ¼$$



- Trong một số trường hợp, ta có thể *dùng PP thử ngẫu nhiên để kiểm tra một tính chất hay giả thuyết nào đó.*

. *Ví dụ 9*: Kiểm tra giả thuyết Fermat bằng thực nghiệm: N ≥ 2 (N khá lớn) là nguyên tố nếu: x^{N-1} mod N = 1, ∀x nguyên dương < N ?

Để phù hợp với điều kiện thực nghiệm trên máy tính, ta xét *N* và số lần lặp *n* lớn vừa phải. Ta có thuật toán sau:

```
{       for  (j = 1; j ≤ n; j++)
        { x = 1 + random(N-1);  // 1 ≤ x < N
          y = x^(N-1) mod N;        // hãy cải tiến cách tính này hiệu quả hơn trên máy tính !
          if (y ≠ 1) {cout << N << " không là số nguyên tố"; Dừng; }
        }
        cout << N << " là số nguyên tố";
}
```

b. *Thuật giải di truyền GA (Genetic Algorithm) và lập trình tiến hoá*

* Lập trình tiến hoá:

Chương trình Tiến hóa = CTDL + GA

* *Các đặc trưng cơ bản của GA*

. Ngẫu nhiên;

. Duyệt toàn bộ giải pháp (lời giải), sau đó chọn giải pháp tốt nhất dựa trên *độ thích nghi* của chúng;

. Không quan tâm đến chi tiết vấn đề mà chỉ quan tâm đến giải pháp và phương pháp biểu diễn nó.

* *Các bước tiến hành chính của thuật giải di truyền GA*

- Bước 1: Chọn mô hình để biểu diễn vấn đề thông qua các *dãy ký hiệu* (số, chữ hoặc hỗn hợp) để biểu diễn cho mỗi giải pháp của vấn đề và *số cá thể* (số lời giải chấp nhận được) trong quần thể biểu diễn vấn đề.
- Bước 2: Tìm *hàm số thích nghi (Fitness function*) và tính số thích nghi *cho từng giải pháp*.
- Bước 3: Dựa trên các số thích nghi, thực hiện việc *sinh sản* và *tiến hoá* (gồm: *lai ghép* và *đột biến*) các giải pháp.
- Bước 4: Tính số thích nghi cho các giải pháp mới sinh sản, *loại bỏ giải pháp kém nhất*, *chỉ giữ lại một số nhất định các giải pháp* (có độ thích nghi cao).
- Bước 5: Nếu chưa tìm được *giải pháp tối ưu* hoặc chưa đến *thời hạn (hay số thế hệ) ấn định* thì trở lại bước 3 để tìm giải pháp mới.
- Bước 6: Nếu tìm được *giải pháp tối ưu* hay *hết thời hạn ấn định thì dừng và xuất kết quả.*

* *Các phương pháp tiến hoá của GA*: *sinh sản (Reproduction), lai ghép (*hay *lai chéo, Crossover), đột biến (Mutation)*. So với lai ghép, tần suất đột biến xảy ra ít hơn nhiều vì quá trình này tạo ra thông tin hoàn toàn mới.

* *Thuật giải GA*

```
Thuật_Giải_Di_ Truyền
{ t =0;
  Khởi tạo lớp P(t);
  Đánh giá lớp P(t);
  while (not(Điều kiện kết thúc)) do
  {     t = t + 1;
        Chọn lọc P(t) từ P(t-1);
        Kết hợp các cá thể của P(t);
        Đánh giá lớp P(t);
  }
}
```

* *Ví dụ 10*: Giải phương trình sau trên tập số tự nhiên: $x^2 = 64$.

. Bước 1: - Xác định số lượng cá thể (giải pháp, biến x): 4
- Dùng dãy ký hiệu nhị phân {0, 1} để biểu diễn mỗi biến

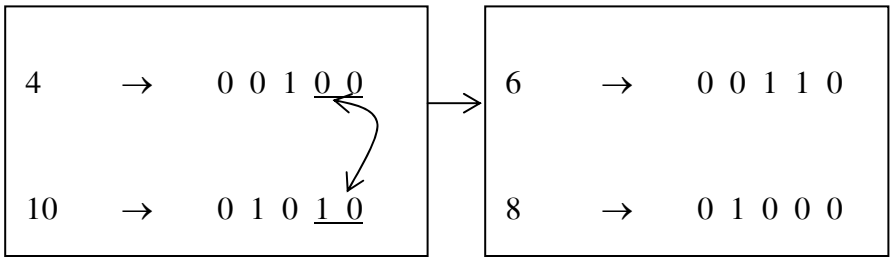
. Bước 2: - Chỉ định các cá thể: 4, 21, 10, 24
- Biểu diễn đáp số: dùng 5 bits

STT	Thập phân	Nhị phân
1	4	00100
2	21	10101
3	10	01010
4	24	11000

. Bước 3: - Chọn hàm số thích nghi: $F(x) = 1000 - |x^2 - 64| \geq 0, \forall x \in [-32..31]$
- Tính độ thích nghi phù hợp cho đáp số ≈ 1000

STT	Thập phân	Nhị phân	HS thích nghi	Chọn
1	4	00100	1048	X
2	21	10101	623	
3	10	01010	964	X
4	24	11000	488	

. Bước 4: - Chọn các đáp số (có độ thích nghi gần phù hợp): 4, 10
- Tiến hoá: lai ghép ở vị trí thứ 3:



. Bước 5: Tính độ thích nghi cho giải pháp mới:

STT	Thập phân	Nhị phân	HS thích nghi	Chọn
1	6	00110	1028	
2	21	10101	623	
3	8	01000	1000	X
4	24	11000	488	

. Bước 6: x = 8 có độ thích nghi 1000: phù hợp → Dừng.
. Bước 7: Đáp số: x = 8.

II.2.3. Nguyên lý mê cung

* *Vét cạn bằng cách quay lui:* Để tránh tràn bộ nhớ vì không gian D quá lớn, ta có thể *xác định dần các thành phần của lời giải* bài toán mặc dù các lời giải có *cấu trúc không tuyến tính* phức tạp. *Lời giải* $x_{LG} = (x_0, x_1, ...)$ *được xây dựng dần (xuất phát từ* $x_0 \in X_0$*) trong quá trình giải cho đến khi gặp điều kiện kết thúc* $P(x_{LG})$.

Các thành phần của lời giải được lưu dần vào tập DONG (có cấu trúc stack). Khi đó, tại mỗi thời điểm đang xét, phần tử ở *đỉnh của DONG chính là phần tử cuối của đường đi* có khả năng dẫn đến lời giải. Vì vậy, ta có thể *kiểm tra điều kiện kết thúc P(DONG) bởi Top(DONG)* $\in$ *DICH.*

Dưới dạng đệ qui, ta có các thuật toán sau để giải BT1 và BT2.

* *Thuật toán* ***vét cạn V2.1*** *(dưới dạng đệ qui giải BT1)*

```
{   for each x ∈ X0  do
    {DONG = {x}; Try_1(DONG);
    }
}

Try_1(DONG)
{ // điều kiện nhận biết trạng thái kết thúc của một lời giải          (*)
  if (P(DONG)) // hay (Top(DONG) ∈ DICH)
        XuấtLờiGiải(DONG);
  else  for each x ∈ B(Top(DONG))
          if (Q(x)) // nếu có yêu cầu thêm về tính chất của lời giải, chẳng hạn
                // (x∉DONG) khi đòi hỏi các thành phần của lời giải không trùng lặp
          {  Push (x, DONG);
             Try_1(DONG);
             Pop(y, DONG); // Trả lại trạng thái trước để quay lui
          }
}
```

trong đó: *Push(x,DONG), Pop(x,DONG)* và *x = Top (DONG)* lần lượt là các thao tác đưa vào, lấy ra và xem một phần tử *x* ở đỉnh ngăn xếp *DONG; B(x)* ($\subset D$) là tập các trạng thái kế tiếp có thể lấy từ *x* để xét tiếp.
Nếu tập xuất phát X_0 trong bài toán chỉ gồm một điểm x_0 thì để thi hành thuật toán, ta chỉ cần gọi *Try_1(*$\{x_0\}$*).*

Để đưa ra *thuật toán V2.2 giải BT2,* ta chỉ cần *thay điều kiện nhận biết trạng thái kết thúc một lời giải (*) bởi điều kiện dừng chương trình* sau đây*:*

```
if (P(DONG)) // hay (Top(DONG) ∈ DICH)
{ XuấtLờiGiải(DONG);
  Dừng;                                  // điều kiện dừng
}
```

Thuật toán *vét cạn kiểu đệ qui* trên đây *vét hết mọi khả năng nhưng tại mỗi bước chỉ lưu một khả năng*, những khả năng còn lại được lưu dần trong ngăn xếp thông qua cơ chế đệ qui. Nếu biểu diễn *không gian tìm kiếm của bài toán dưới dạng đồ thị hay cây* thì *thuật toán vét cạn* trên đây thực chất là thủ tục *tìm kiếm theo chiều sâu*.

Trong trường hợp lời giải là vectơ hữu hạn chiều (chẳng hạn, kích thước được biết trước là cố định hữu hạn), ta có phiên bản đơn giản và hiệu quả sau đây thường gặp trong các tài liệu tin học trước đây.

Thủ tục vét cạn Try trong trường hợp cấu trúc rời rạc tuyến tính đơn giản

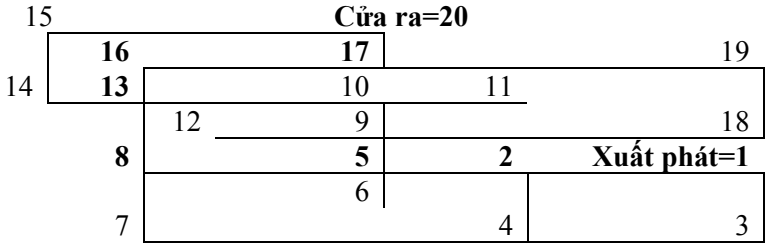
```
TryRờiRạc(j: integer)
{    for (k thuộc tập các khả năng) do
      if  (chấp nhận khả năng thứ k)
         { Xác định x_j theo khả năng thứ k;
           Đánh dấu (đã xét) trạng thái mới;
           if (x_j là trạng thái kết thúc)     // hay thoả điều kiện kết thúc
                    XuấtLờiGiải(x);
           else TryRờiRạc(j+1);
           Trả lại trạng thái cũ;              // Bỏ việc đánh dấu trạng thái cũ
         }
}
```

* *Ví dụ 11*: Tìm các đường đi từ điểm xuất phát đến cửa ra của mê cung trên *hình II.1*. Ta biểu diễn mê cung dưới dạng ma trận kề a[j,k] =1 hay 0 nếu có hay không có đường đi từ j đến k tương ứng, với số đỉnh n=20.

Gọi thủ tục *TryRờiRạcMêCung*(1) sau đây để giải bài toán mê cung.

```
TryRờiRạcMêCung (SoDinh)   // trước đó gán x[1] = XuấtPhát = 1
{    int k;
   for (k=1; k <= n; k++)
    // nếu có đường đi từ đỉnh x[SoDinh] đến đỉnh k và chưa đi qua k
     if  (a[x[SoDinh], k]==1 && DaDiQua[k]==0)
        {  x[SoDinh+1]=k;
           DaDiQua[k] = 1;                 // Đánh dấu đã đi qua k
           if (x[SoDinh+1] == Cửa_ra)
                    XuấtLờiGiải(x, SoDinh+1);
           else TryRờiRạcMêCung (SoDinh+1);
           DaDiQua[k] = 0;                 // Bỏ việc đánh dấu trạng thái cũ
```

```
        }
}
```



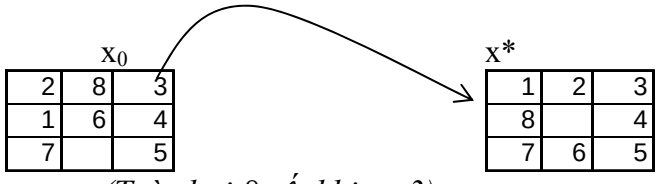
(Hình II.1)

II.2.4. Các phương pháp biểu diễn và giải quyết vấn đề trong không gian trạng thái bằng cây và đồ thị

a. Biểu diễn vấn đề trong không gian trạng thái

Trước tiên, ta xét hai ví dụ.

- *Ví dụ 12* (*bài toán Toci* hay *trò chơi n^2 - 1 số,* n là số tự nhiên, n > 2): Trong bảng ô vuông n hàng, n cột, mỗi ô chứa một số nguyên từ 1 đến n^2 -1 sao cho không có hai ô có cùng giá trị. Chỉ được có một ô trong bảng bị trống (ta có thể gán trị 0). Xuất phát từ một cách sắp xếp x_0 nào đó các số trong bảng, hãy dịch chuyển ô trống sang phải, trái, lên, xuống (nếu có thể được) để đưa về bảng mục tiêu x^* như sau:



(Trò chơi 8 số, khi n=3)

Để giải bài toán này, ta *biểu diễn nó trong không gian trạng thái S*. Mỗi trạng thái là một ma trận cấp *n x n* nhận các giá trị nguyên từ 0 đến n^2 -1 (trị 0 thay cho vị trí trống trên bảng) sao cho không có hai phần tử khác nhau có cùng trị, mỗi toán tử *o* là một phép dịch chuyển hợp lệ từ bảng này sang bảng khác. Số trạng thái chấp nhận được khá lớn: khoảng (1/2)*16! ≈ $10,5*10^{12}$ (khi n=4). Một cách *biểu diễn trực quan đối với không gian trạng thái và các toán tử là* ***đồ thị****. Trong đồ thị định hướng này các đỉnh tương ứng với các trạng thái, còn các cung tương ứng với các toán tử*. Ta cần *xây dựng dần các toán tử*, bắt đầu từ các toán tử có thể áp dụng cho trạng thái đầu, sau đó ở mỗi bước thêm vào một toán tử hợp lệ nào đó cho đến khi đạt được trạng thái đích.

Điểm mấu chốt khi giải quyết bài toán trong không gian trạng thái là lựa chọn một dạng mô tả nào đó của các trạng thái phù hợp với bản chất vật lý của bài toán. Ta *cần biểu diễn các trạng thái sao cho việc áp dụng các toán tử biến đổi trạng thái trở nên đơn giản hơn.*

* Có *hai cách biểu diễn: tập* ***O*** *các toán tử biến đổi trạng thái hoặc tập* ***P*** *những luật sinh để chuyển trạng thái:*

- Gọi ***O*** là tập các hàm *o* xác định và nhận trị trên không gian trạng thái S:

$o : S \to S$

Với ví dụ trên, tập các toán tử chuyển đổi trạng thái O gồm 4 toán tử $O=\{o_{len}, o_{xuong}, o_{trai}, o_{phai}\}$. Chẳng hạn, toán tử dịch chuyển vị trí ô trống lên trên o_{len} được xác định như sau:

$o_{len}(A) = B$

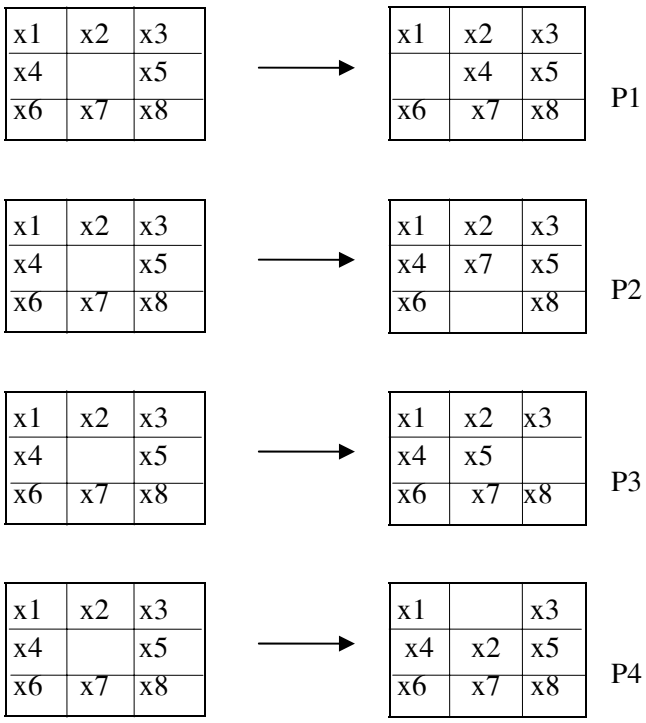
trong đó:

. B = $[b_{ij}]$, A = $[a_{ij}]$, giả sử ô trống trong ma trận A ở vị trí (i_0, j_0)

$$. b_{i,j} = \begin{cases} a_{io-1, jo} & \text{nếu } (i, j) = (i_o, j_o) \quad , i_o > 1 \\ a_{io, jo} & \text{nếu } (i, j) = (i_o - 1, j_o) \quad , i_o > 1 \\ a_{i,j} & \text{nếu ngược lại} \end{cases}$$

- Gọi ***P*** là tập các luật sinh (production rules) p_{ik}: $S_i \Rightarrow S_k$ để chuyển từ trạng thái S_i đến trạng thái S_k

Với ví dụ trên (với n=3), ta có các luật sinh sau:



x1	x2	x3
	x4	x5
x6	x7	x8

⟶

	x2	x3
x1	x4	x5
x6	x7	x8

P5

x1	x2	x3
	x4	x5
x6	x7	x8

⟶

x1	x2	x3
x6	x4	x5
	x7	x8

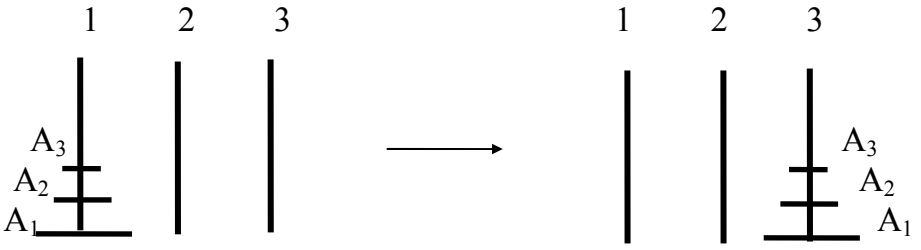
P6

- *Nhận xét*:

. Với cách biểu diễn dùng các *toán tử có biểu diễn tổng quát*: *số các toán tử ít (4, với bài toán $n^2 - 1$ số) , rất gọn và dễ cài đặt*.

. Với cách *liệt kê dưới dạng các luật sinh* tuy *trực quan* nhưng *số lượng quá lớn (4($n^2 - n$), với bài toán $n^2 - 1$ số).* Ta thường dùng cách liệt kê này với các bài toán mà phép chuyển đổi trạng thái rất khó khái quát.

- *Ví dụ 13* (bài toán *tháp Hà Nội*): Cho 3 cọc 1, 2, 3. Ở cọc 1 ban đầu có n đĩa sắp xếp theo thứ tự đĩa lớn ở dưới và đĩa nhỏ ở trên. Hãy dịch chuyển n đĩa đó sang cọc 3 sao cho: mỗi lần chỉ chuyển 1 đĩa, trong mỗi cọc không chấp nhận đĩa lớn nằm trên đĩa nhỏ. Với n=3, ta có:



Với bài toán này, ta có thể biểu diễn mỗi trạng thái là bộ ba (i, j, k) để mô tả đĩa A_1 ở cọc i, đĩa A_2 ở cọc j, đĩa A_3 ở cọc k. Khi đó, các toán tử dịch chuyển trạng thái sau là hợp lệ:

(i, j, k) $\rightarrow$ (i, j, j) , $k \neq j$

(i, j, k) $\rightarrow$ (i, j, i) , $k \neq i$

. . .

* Để *biểu diễn bài toán trong không gian trạng thái, cần xác định rõ:*

- *Không gian S biểu diễn các trạng thái.*
- *Tập O các toán tử biến đổi trạng thái hoặc tập P các luật sinh.*
- *Trạng thái đầu $x_o \in S$ và tập các trạng thái đích DICH $\subset S$*

* Một cách hình thức, ta có thể *phát biểu bài toán tìm kiếm trong không gian trạng thái dưới 3 dạng tương đương* như sau: Cho trạng thái đầu $x_o \in S$ *và tập các trạng thái đích DICH* $\subset S$.

Dạng 1 (toán tử): *Problem(S, O, x_o, DICH)*

Hãy tìm:

. dãy trạng thái $x_o,..., x_n$ sao cho $x_n \in$ DICH và có thể áp dụng dãy các toán tử biến đổi trạng thái $o_i \in O$ nào đó để chuyển từ x_{i-1} đến x_i

$$o_i : x_{i-1} \rightarrow x_i \quad \forall\, i = 1, 2, ..., n$$

. hoặc dãy toán tử $o_1 ,..., o_n \in O$: $o_n (o_{n-1} (...o_1 (x_o)...)) \in$ DICH

Dạng 2 (luật sinh): *Problem(S, P, x_o, DICH)*

Hãy tìm:

. dãy trạng thái $x_o,..., x_n$ sao cho $x_n \in$ DICH và có thể áp dụng dãy các luật sinh $p_i \in P$ nào đó để chuyển từ x_{i-1} đến x_i

$$p_i : x_{i-1} \Rightarrow x_i \ \forall\, i = 1, 2, ..., n$$

. hoặc dãy luật sinh $p_1 ,..., p_n \in P$ sao cho:

$$x_o \overset{p_1}{\Rightarrow} x_1 \Rightarrow \ ... \ \overset{p_n}{\Rightarrow} x_n \in \text{DICH}$$

Nếu không gây ra nhầm lẫn ta có thể dùng ký hiệu *Problem(x_o, DICH)*.

Dạng 3 (đồ thị):

Đồ thị định hướng G là cặp $G = (S, A)$, trong đó S là tập các đỉnh, A là tập các cung: $A = \{(a, b)/\ a, b \in S\} \subset S \times S$.

Với mỗi $n \in S$, ta ký hiệu tập các đỉnh con của n là: $B(n) = \{con \in S: (n,con) \in A\}$. Một cách tổng quát, ta định nghĩa:

$$B^1(n) \equiv B(n)$$

$$B^k(n) \equiv B(B^{k-1}(n)) = \bigcup_{m \in B^{k-1}(n)} B(m), k > 1$$

Khi đó:

$$\hat{B}(n) \equiv \bigcup_{i=1}^{\infty} B^i(n)$$

được gọi là *tập các đỉnh hậu duệ* của n và nếu $m \in B(n)$ thì n được gọi là *tổ tiên* của m. Dãy các đỉnh $p = \{n_1,..., n_k\}$ sao cho: $\forall\, i=1,..., k-1, a_i = (n_i , n_{i+1}) \in A$ được gọi là *đường đi* từ n_1 đến n_k hay còn có thể biểu diễn bởi: $p = \{a_1 ,..., a_{k-1}\}$, với $a_i = (n_i , n_{i+1})$. Nếu tồn tại một đường đi từ đỉnh *n* đến đỉnh *m* thì $m \in B(n)$ và ngược lại. Khi đó ta còn nói rằng m có thể đạt được từ n.

Khi đồ thị G có đỉnh gốc $n_o \in S$ và $\forall n \in S\backslash\{n_o\}$ tồn tại duy nhất đường đi từ n_o đến n ($\forall n \in S\backslash\{n_o\}$, $n \in B(n_o)$ và $\exists! m \in S$, $n \in B(m)$) thì G được gọi là *cây với gốc n_o*.

Thông thường, người ta thêm vào mỗi cung một đại lượng thể hiện *ý nghĩa được lượng hóa* của nó thông qua hàm giá *c:* $A \to R_+$, $c(n_i, n_k) \in R_+$, $\forall(n_i, n_k) \in A$. Khi đó, ta định nghĩa giá của đường đi $p = \{n_1, ..., n_k\}$:

$$c(p) = \sum_{i=1}^{k-1} c(n_i, n_{i+1})$$

Trường hợp $c \equiv 1$ thì c*(p)* được gọi là *độ dài đường đi của p.*

Theo ngôn ngữ đồ thị, *không gian trạng thái tương ứng với đồ thị định hướng trong đó: các trạng thái tương ứng với các đỉnh trong đồ thị và có một cung nối từ trạng thái s đến trạng thái t nếu tồn tại toán tử o sao cho o(s) = t* (hoặc tồn tại luật sinh *p* sao cho *p*: $s \Rightarrow t$).

* Bài toán tìm kiếm trong không gian trạng thái có thể phát biểu dưới *dạng đồ thị* như sau*:*

- <u>Bài toán 3</u>: Problem_3(G = (S, A), x_0, DICH) ***(BT3)***

*Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$. Hãy tìm **một** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

***- <u>Bài toán 3*</u>**: (bài toán tìm kiếm tối ưu)*

Problem_3*(G = (S, A), x_0, DICH, c) ***(BT3*)***

*Cho đồ thị G = (S, A), với hàm giá c: $A \to R_+$, đỉnh xuất phát $x_0 \in S$ và tập đích DICH $\subset S$. Hãy tìm **một** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH* và làm *tối ưu hàm giá.*

Từ tập các cung *A*, ta có thể *xây dựng các thuật toán tìm kiếm lời giải dựa trên toán tử B(x),* $\forall x \in S$. Trong nhiều bài toán thực tế, *B(x) chính là tập các trạng thái kế tiếp hợp lệ từ x. Khi không gian trạng thái S quá lớn,* lời giải $x_{LG} = \{x_0, x_1, ...\}$ thường *được xây dựng dần trong quá trình giải cho đến khi gặp điều kiện kết thúc $P(x_{LG})$ (hay P'(x*): khi tìm được trạng thái mục tiêu $x^* \equiv TrạngTháiCuối(x_{LG}) \in DICH$ ứng với BT3 hoặc thêm một điều kiện nào đó, chẳng hạn tối ưu một hàm mục tiêu hay hàm giá ứng với BT3*).*

** Hai phương pháp xây dựng đồ thị G = (S, A)*

- *PP tường minh*: *tập các nút S và tập các cung A đã biết và được xây dựng trước*. Thông thường, đối với các *đồ thị hữu hạn ứng với các bài toán đơn giản và có kích thước nhỏ* mới có thể biểu diễn tường minh chúng dưới dạng bảng.
- *PP không tường minh*: xuất phát từ đỉnh ban đầu x_0, *trong quá trình tìm kiếm lời giải, xây dựng dần các đỉnh con dựa trên toán tử B(n)* để tạo ra các đỉnh con của *n*. Nghĩa là chỉ khi nào xét đến đỉnh n, tập các đỉnh con *n* mới được xây dựng. Phương pháp này rất có ý nghĩa, đặc biệt là đối với các *đồ thị biểu diễn những bài toán phức tạp và có kích thước lớn*.

Nếu *xét BT3 dưới dạng BT1 thì D là một tập con của tập* $U_{n\geq 1}S^n$.Trong những bài toán *lớn và phức tạp,* do *khó xác định tập D ngay từ đầu* hoặc *có thể xác định D nhưng kích thước của nó quá lớn,* việc dùng các *thuật toán vét cạn V1.a* hay *V1.b là không hiệu quả.* Đối với lớp các bài toán mà *các thành phần của lời giải có thể xác định dần trong quá trình giải,* ta sẽ *cải biên các thuật toán đơn sơ V1* để *thu được dần các thuật toán hiệu quả hơn* trong phần tiếp theo như*: TìmKiếm,* A^T, A^{KT}, *A*, ...*

Trong các thuật toán hay thuật giải tìm kiếm lời giải sau đây, ta dùng tập DONG để lưu các trạng thái đã xét, tập MO dùng để lưu các trạng thái dự định sẽ xét trong các bước kế tiếp.

*b. Phương pháp tìm kiếm lời giải cho BT3 (dưới dạng **lặp**)*

Trong phần này, không có gì khó khăn, ta đưa ra thuật toán tìm kiếm lời giải cho bài toán mở rộng của *BT3* như sau: Cho X_0 ($\subset$ S) là tập những trạng thái có thể xuất phát. Thay vì tìm đường đi từ x_0 đến DICH, ta tìm đường đi mà có thể xuất phát từ một trong các trạng thái $x \in X_0$ đến DICH. Khi đó, để giải *BT3* ta chỉ cần gọi: *TìmKiếm({x_0}, DICH).*

*** *Thuật toán tìm kiếm***: (Giải *BT3'*: *Problem_3'(G = (S, A), X_0, DICH)*)

TìmKiếm(X_0, DICH)

```
{   MO = X0; DONG = ∅; // tập DONG có cấu trúc stack
    if (∃y ∈ X0: P'(y))              // hay if (∃y ∈ MO ∩ DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                   // lấy một phần tử  x ra khỏi tập MO
      DONG  = DONG U {x};            // đưa x vào tập DONG
      if (∃y ∈ B(x) : P'(y) đúng)    // hay if (∃y ∈ B(x)∩DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
      else MO = MO U B(x);           // đưa tập đỉnh con B(x) của x vào tập MO
    }
    write ("Không có lời giải");
```

}

Từ đây về sau, để gần hơn với dạng cài đặt thành chương trình máy tính, ta *qui ước*: Remove(x, MO) là thao tác rút phần tử x khỏi đầu hàng đợi MO, Add(x, MO) là thao tác thêm phần tử x vào đuôi hàng đợi MO, AddSet(B(x),MO) là thao tác thêm tập B(x) vào đuôi hàng đợi MO; Push(x, MO) là thao tác thêm phần tử x vào đầu (hay đỉnh) ngăn xếp MO, PushSet(B(x), MO) là thao tác thêm tập B(x) vào đỉnh ngăn xếp MO, Pop(x, MO) là thao tác rút x khỏi đỉnh ngăn xếp MO, x = Top(MO) là hàm trả lại (nhưng không lấy ra khỏi) phần tử ở đầu danh sách MO.

- Thuật toán *XuấtLờiGiải*(y, DONG) cho *cây G (tổng quát hơn cho đồ thị* mà mỗi nút có *không quá một nút cha)*, với tập DONG có cấu trúc stack, dưới dạng đường đi từ x_0 đến y được lưu trong ngăn xếp *LờiGiải* như sau:

```
XuấtLờiGiải(y, DONG)
{ LờiGiải = {y};
  while (not(EmptyStack(DONG)))
  {     Pop(x, DONG);
        // nếu có cung nối từ x đến phần tử ở đỉnh ngăn xếp LờiGiải thì đưa // x vào thành
        phần LờiGiải
         if ((x, Top(LờiGiải)) ∈ A) // hay Top(LờiGiải) ∈ B(x)
                Push(x, LờiGiải);
  }
   ShowStack(LờiGiải); // xuất các phần tử của stack LờiGiải
}
```

Dựa trên thuật toán *TìmKiếm* nhằm tìm lời giải đầu tiên cho bài toán *BT3*, ta có thể cải biên để thu được thuật toán vét cạn dưới dạng lặp *VétCạn để tìm <u>mọi</u> lời giải của bài toán sau (bài tập):*

- <u>Bài toán 4</u>: Problem_4(G = (S, A), x_0, DICH) ***(BT4)***

*Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$. Hãy tìm **mọi** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

***c. <u>Các dạng đặc biệt thường gặp</u>**: Các thuật toán tìm kiếm lời giải trong không gian trạng thái theo chiều rộng, chiều sâu, chiều sâu dần, tìm kiếm cực tiểu giá thành A^T.*

Để *giải BT3 trên cây G*, trong thuật toán *TìmKiếm*, với các *cách chọn cấu trúc dữ liệu khác nhau của tập MO, ta sẽ có các phương pháp tìm kiếm khác nhau*. Sau đây, ta luôn xem *x = Get(MO)* là thao tác *lấy phần tử x ra khỏi đầu* danh sách *MO*.

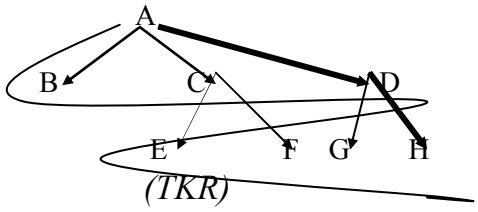
**** Phương pháp tìm kiếm theo chiều rộng (TKR)**:* chính là *thuật toán TìmKiếm,* với tập *MO có cấu trúc hàng đợi (queue)* hay thay MO = MO U B(x) bởi phép toán *AddSet(B(x),MO).*

- *Nhận xét:* Khi G là cây với gốc x_0,

. nếu tồn tại ít nhất một đường đi từ x_o tới tập DICH thì thuật toán TKR dừng và cho ta đường đi p có độ dài ngắn nhất (thậm chí khi G là cây vô hạn);

. nếu không tồn tại đường đi như vậy thì thuật toán dừng nếu và chỉ nếu cây G là hữu hạn.

- *Ví dụ 14*: Xét cây như hình dưới đây với tập DICH = {H}. Thủ tục TKR cho kết qủa là đường đi ADH theo hành trình duyệt các đỉnh ABCDEFGH.



(TKR)

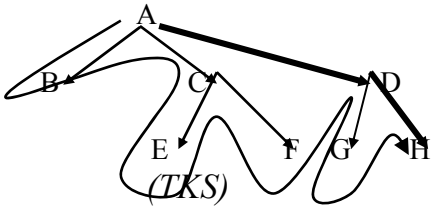
**** Phương pháp tìm kiếm theo chiều sâu (TKS)**:* chính là *thuật toán TìmKiếm,* với tập *MO có cấu trúc ngăn xếp (stack),* hay thay MO = MO U B(x) bởi phép toán *PushSet(B(x),MO)*.

- *Nhận xét:* Khi G là cây với gốc x_0,

. nếu cây *G hữu hạn* thì thủ tục *TKS sẽ dừng* và nếu tồn tại đường đi từ x_0 đến DICH thì sẽ cho kết qủa là một đường đi từ x_o tới tập DICH, khi đó *đường đi* nhận được trong thủ tục TKS *không nhất thiết là đường đi ngắn nhất*. Hơn nữa, nếu cây *G vô hạn*, thủ tục *TKS có thể lặp đến vô hạn, thậm chí trong trường hợp tồn tại đường đi từ x_o đến tập DICH*.

. nếu *không tồn tại đường đi* từ x_0 đến DICH thì thủ tục *TKS chỉ dừng khi cây G hữu hạn*.

- *Ví dụ 15*: Xét cây như hình dưới đây với tập DICH = {D, H}. Thủ tục TKS cho kết qủa là đường đi AD theo hành trình duyệt các đỉnh ABCEFD.



(TKS)

Để *khắc phục tình trạng không dừng của thuật toán TKS* ngay cả khi tồn tại đường đi từ gốc của cây đến DICH, ta đưa vào đại lượng DS đặc trưng cho giới hạn sâu và dùng khái niệm *độ sâu d(x)* của đỉnh x trong cây G = (S, A) được định nghĩa đệ qui như sau:

$d(x_o) = 0$

$d(con) = d(cha) + 1$, nếu $con \in B(cha)$ hay $(cha, con) \in A$.

**** Phương pháp tìm kiếm <u>sâu dần</u> (TKSD)*** *giải BT3 với độ sâu DS = k≥ 1: điều chỉnh lại thuật toán TìmKiếm, với tập MO được trang bị các phép toán cơ bản của cả hàng đợi lẫn ngăn xếp.*

```
TKSD(x0, DICH, k)
{   MO = {x0};
    DONG = ∅;                          // chọn DONG có cấu trúc stack
    DS = k;
    if (P'(x0))                        // hay if (x0 ∈ DICH)
            {XuấtLờiGiải(x0, DONG); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                     // lấy phần tử x từ đầu danh sách MO
      Push(x, DONG);                   // đưa x vào tập DONG
      if (∃y ∈ B(x)∩DICH) {XuấtLờiGiải(y, DONG); Dừng;}

      if (d(x) > DS) DS = DS + k;      // khi đó d(x) <= DS
      if (d(x) < DS) PushSet(B(x), MO);// tìm theo chiều sâu
      else AddSet(B(x), MO);           // d(x) = DS: việc tìm được lan theo chiều rộng
    }
    write ("Không có lời giải");
}
```

- <u>Nhận xét</u>: Khi G là cây với gốc x_0,

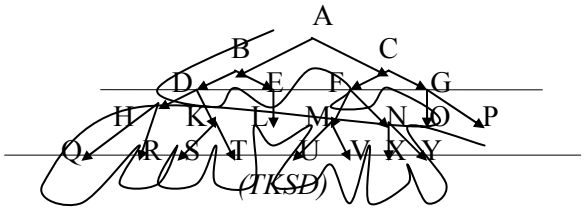
. với *k = 1, thủ tục TKSD trở thành thủ tục TKR*. Khi *k rất lớn* (chẳng hạn k ≥ chiều cao của cây G hay $k \to \infty$ đối với cây vô hạn) thủ tục *TKSD sẽ giống* hay *gần giống tương ứng như thủ tục TKS*;

. với k ≥ 2 thủ tục TKSD *tìm kiếm theo chiều sâu* đối với các đỉnh *có độ sâu nằm trong khoảng từ tk đến (t+1)k* với t = 0, 1, 2, ...;

. nếu *tồn tại ít nhất một đường đi* từ gốc tới DICH thì thủ tục TKSD sẽ *dừng* và cho kết qủa là đường đi *có độ dài khác đường đi ngắn nhất không quá k-1;*

. nếu trong cây *không tồn tại đường đi* như vậy thì thủ tục TKSD *dừng khi và chỉ khi cây G là hữu hạn*.

- *Ví dụ 16*: Xét thủ tục TKSD trên cây như hình dưới đây, với độ sâu k = 2, thứ tự duyệt các đỉnh là : ABDECFG__HQRKSTLMUVNXYOP



**** Phương pháp tìm kiếm cực tiểu A^T:*** Giải *bài toán tối ưu BT3** trên cây G - *Problem_3*(G = (S, A), x_0, DICH, **c**)* - với tiêu chuẩn tối ưu: *tìm một đường đi* từ x_0 đến một trong các trạng thái thuộc tập DICH *có hàm giá đường đi g^0 cực tiểu*. Trong đó, hàm giá đường đi g^0 được xác định theo kiểu đệ qui như sau:

$g^0(x_0) = 0$

$g^0(con) = g^0(cha) + c(cha, con), \ \forall \ (cha, con) \in A$.

Thuật toán sau đây tương tự như thuật toán *TìmKiếm*, trong đó: tập DONG có cấu trúc stack và tập MO chứa các phần tử $(x, g^0(x))$, thao tác *lấy ra một phần tử* được thực hiện *ở đầu* danh sách MO, việc đưa một phần tử x vào danh sách MO *ChènTăng*($(x, g^0(x))$, MO) được thực hiện theo kiểu *chèn $(x, g^0(x))$ vào MO tăng dần theo $g^0()$ từ đầu đến cuối* danh sách MO.

```
A^T(x0, DICH)
{   DONG = ∅;                          // DONG có cấu trúc stack
    g^0(x0)=0; MO = {(x0, g^0(x0))};
    while (MO ≠ ∅)
    { Pop((x, g^0(x)), MO);            // trong MO, x ở đỉnh và có g^0(x) nhỏ nhất !
      if (x ∈ DICH) {XuấtLờiGiải(x, DONG); Dừng; }
      Push(x, DONG);                   // đưa x vào đỉnh stack DONG
    //ChènTăng(y,MO) ∀y∈B(x)={(con,g^0(con)):(x,con)∈A & g^0(con)=g^0(x)+c(x, con)}
      ChènTăngTập(B(x), MO);
    }
    write ("Không có lời giải");
}
```

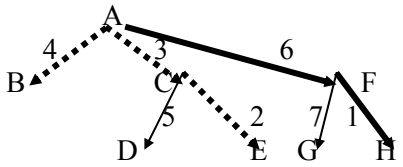
- *Nhận xét*:

. Thủ tục *TKR là trường hợp riêng của thủ tục A^T* khi hàm giá c≡1.

. Thủ tục *TKS cũng là trường hợp riêng của thủ tục A^T* khi: hoặc thay thao tác *ChènTăngTập*(B(x), MO) bởi *ChènGiảmTập*(B(x), MO); hoặc xem hàm giá c: $A \to R$ và $c \equiv -1$.

. Nếu trong cây G *tồn tại đường đi p: $x_o \rightarrow DICH$* thì *thủ tục A^T sẽ dừng* và cho kết qủa là đường đi p_o sao cho $c(p_o) \rightarrow$ min, nếu: hoặc cây *G hữu hạn*, hoặc cây *G vô hạn nhưng tổng giá theo mọi nhánh vô hạn phải phân kỳ hoặc hội tụ về một giá trị lớn hơn giá tối ưu* !

- *Ví dụ 17*: Xét cây và giá các cung được cho như hình dưới đây và tập DICH = {D, H}. Thủ tục tìm kiếm cực tiểu A^T cho kết qủa là đường đi AFH.



(Thuật toán A^T)

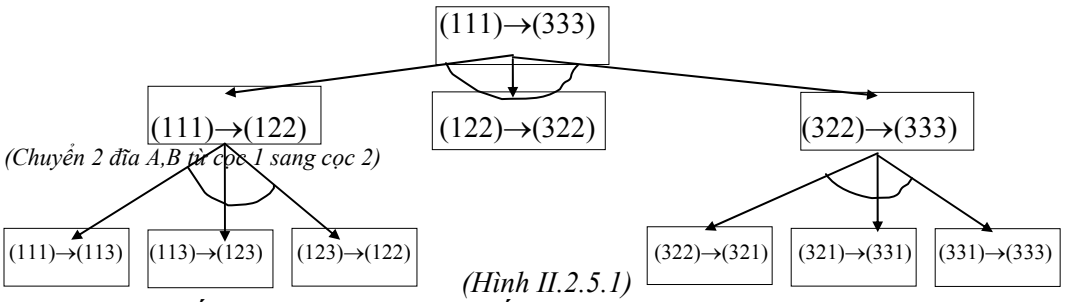
- *Chú ý:* Trong thuật toán $\mathbf{A^T}$, ta *chọn phương án tối ưu nhất* trong *tất cả những phương án đã đi qua (được lưu lại để quay lui khi lạc hướng)*. Chính vì thế, với các bài toán lớn, có thể *xảy ra tình trạng nhanh tràn bộ nhớ* nếu các bước kế tiếp cần lưu quá nhiều; khi đó ta có thể *dùng heuristic* sau*: thay vì lưu hết 100% các phần tử B(x) vào tập MO, ta chỉ cần lưu p%, $1 \leq p \leq 100$,* mà thôi ! *Giá trị của p phụ thuộc vào từng bài toán cụ thể* và *tài nguyên máy tính hiện có.*

II.2.5. Quy bài toán về bài toán con và các chiến lược tìm kiếm trên đồ thị VÀ/HOẶC

a. Qui bài toán về bài toán con

* *Ý tưởng*: dựa trên ý tưởng của phương pháp chia để trị, tách bài toán lớn và phức tạp ban đầu thành những bài toán con nhỏ, đơn giản tới mức sơ cấp (lời giải của chúng đã biết).

- *Ví dụ 18*: Xét bài toán tháp Hà nội với n = 3 và dùng cách biểu diễn như trong ví dụ 2 phần 2.2.4.a. Ta *qui bài toán về các bài toán con dạng VÀ* như sau:



(Hình II.2.5.1)

Liên kết những lời giải sơ cấp trên từ trái sang phải, ta có lời giải của bài toán ban đầu:

(111)→(113)→(123)→ (122)→(322) →(321)→(331)→(333).

Cây bình thường là cây HOẶC, dạng đặc biệt của cây VÀ/HOẶC. Cây VÀ/HOẶC thường được dùng để *biểu diễn quá trình chia bài toán lớn thành nhiều nhóm các bài toán con được nối kết lôgic với nhau bởi dãy các phép toán lôgic: and, or*.

b. <u>Biểu diễn bài toán dưới dạng đồ thị VÀ/HOẶC</u>

Đồ thị định hướng VÀ/HOẶC là đồ thị định hướng thông thường G = (S, A) và thêm vào tính chất: với mỗi đỉnh n ∈ S tất cả các đỉnh con của nó B(n) cùng thuộc vào một trong 2 kiểu: đỉnh VÀ hay đỉnh HOẶC. Khi các đỉnh con m của n là đỉnh VÀ thì các cung nối các đỉnh con của nó m ∈ B(n) được nối với nhau bởi ngoặc tròn (như dấu góc tròn trong hình học phẳng). Sự tương ứng giữa quá trình qui bài toán về bài toán con với đồ thị VÀ/HOẶC được cho trong bảng sau:

<u>Qui bài toán về bài toán con</u>	*<u>Đồ thị VÀ/HOẶC</u>*
Bài toán	Đỉnh
Toán tử qui bài toán về bài toán con	Cung
Bài toán ban đầu	Đỉnh gốc (đỉnh xuất phát)
Bài toán sơ cấp	Đỉnh lá
Các bài toán con phụ thuộc	Đỉnh con dạng VÀ
Các bài toán con độc lập	Đỉnh con dạng HOẶC
Lời giải bài toán	Đồ thị con lời giải
Giải bài toán	Tìm đồ thị con lời giải

<u>Định nghĩa đỉnh giải được</u> (đỉnh gđ):

- Các đỉnh lá tương ứng với bài toán con *sơ cấp giải được* (hay *đỉnh kết thúc*) là đỉnh giải được.
- Nếu đỉnh n có các đỉnh con là đỉnh HOẶC thì nó giải được khi và chỉ khi *tồn tại* một đỉnh con của nó giải được.
- Nếu đỉnh n có các đỉnh con là đỉnh VÀ thì nó giải được khi và chỉ khi *mọi* đỉnh con của nó giải được.

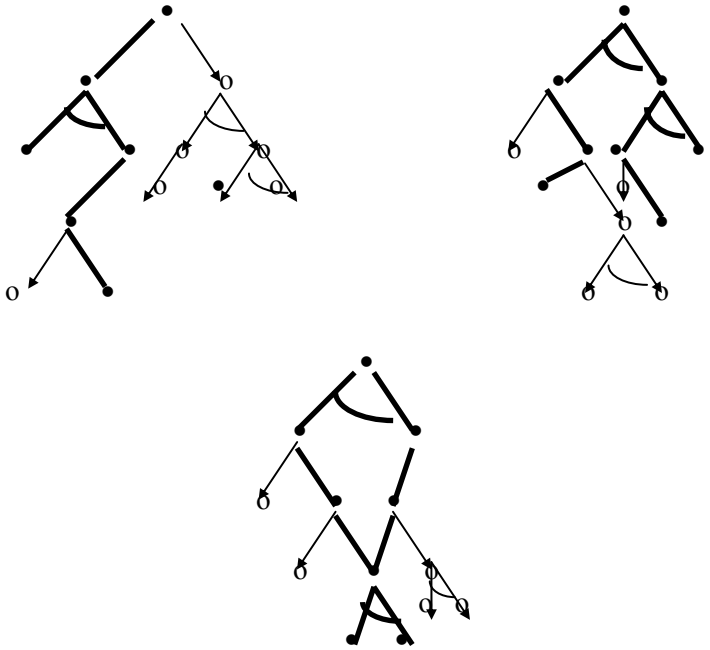
Đồ thị lời giải là đồ thị con của đồ thị VÀ/HOẶC chỉ bao gồm đỉnh xuất phát và các đỉnh giải được liên quan đến nó.

<u>Định nghĩa đỉnh không giải được</u> (đỉnh kgđ):

- Các đỉnh lá tương ứng với bài toán con *sơ cấp không giải được* là đỉnh không giải được.
- Nếu đỉnh có các đỉnh con là đỉnh VÀ thì nó là không giải được khi và chỉ khi *tồn tại* một đỉnh con của nó không giải được.
- Nếu đỉnh có các đỉnh con là đỉnh HOẶC thì nó là không giải được khi và chỉ khi *mọi* đỉnh con của nó không giải được.

- *<u>Ví dụ 19</u>*: Trên hình II.2.5.2 là các đồ thị VÀ/HOẶC, trong đó các đỉnh giải được tương ứng với các hình tròn tô đậm và các cung trong đồ thị con lời giải cũng được tô đậm, các đỉnh không giải được được đánh dấu bởi vòng tròn.

- <u>*Nhận xét*</u>: Khi đồ thị VÀ/HOẶC *không có đỉnh VÀ* thì nó trở thành *đồ thị thông thường*. Lúc đó *đồ thị lời giải suy biến thành một đường đi* từ đỉnh xuất phát đến một đỉnh kết thúc nào đó. *Mục đích của quá trình tìm kiếm trên đồ thị VÀ/HOẶC là chứng tỏ đỉnh xuất phát có thể giải được hay không giải được theo hướng ngược từ các đỉnh lá đến đỉnh xuất phát* và trong trường hợp khẳng định thì *chỉ ra đồ thị con lời giải (*có thể thỏa mãn thêm một tính chất nào đó).



(Hình II.2.5.2)

c. <u>Các phương pháp tìm kiếm trên cây VÀ/HOẶC</u> *(theo chiều rộng, chiều sâu, cực tiểu giá thành)*

Để đơn giản, ta *chỉ xét các phương pháp tìm kiếm trên cây VÀ/HOẶC*. Tùy theo *phương pháp (PP) lựa chọn thứ tự các đỉnh sẽ xét* mà ta có các phương pháp tìm kiếm khác nhau theo: chiều sâu, chiều rộng, tìm kiếm cây lời giải có giá thành nhỏ nhất, ...

Sự *khác biệt chủ yếu của các PP tìm kiếm trên đồ thị hay cây VÀ/HOẶC với các PP tìm kiếm trong không gian trạng thái* ở mục trước là việc *kiểm tra tính giải được của đỉnh xuất phát* (xem nó giải được hay không giải được) và các PP sắp xếp, lựa chọn đỉnh để mở phức tạp hơn nhiều. Với mỗi đỉnh n, ta sẽ cần dùng đến 2 thủ tục gán nhãn giải được *gđ(n* $\in$ *S)* hay gán nhãn không giải được *Kgđ(n* $\in$ *S)*.

Với mỗi đỉnh n ∈ S, ta dùng các ký hiệu:

- Nếu n là đỉnh kết thúc thì kt(n) = true, ngược lại kt(n) = false.

-Nhãn(n) = "*gđ*" , khi n là đỉnh giải được
"*kgđ*" , khi n là đỉnh không giải được
"*kxđ*" , (không xác định) nếu n chưa đủ thông tin để quyết định hoặc n chưa được xét tới

Chú ý rằng, một *đỉnh không phải là giải được* không nhất thiết là không giải được mà còn có thể là không xác định !

- KiểuVÀ(n)=true nếu các đỉnh con của n là đỉnh VÀ và nhận trị false nếu ngược lại.

- Để đơn giản hoá trong các thuật toán sẽ trình bày sau đây, ngay từ đầu, ta sẽ gán nhãn:

. "*gđ*" : đối với các đỉnh lá giải được
. "*kgđ*" : đối với các đỉnh lá không giải được
. "*kxđ*" : đối với mọi đỉnh còn lại

* ***Thủ tục gán nhãn giải được***

gđ(n ∈ S)

```
{1 if (nhãn(n) = "kxđ") then                // do đó B(n) ≠ ∅
      if (n ∈ MO U DONG) then               // nếu n ∉ MO U DONG: chưa xét n
            if (KiểuVÀ(n))
            then {2 bien = true;
                  while (B(n) ≠∅ and bien) do
                  {     con ← get(B(n));
                        gđ(con); bien = (nhãn(con) == "gđ");
                  }
                  if (bien) then      // mọi đỉnh con dạng VÀ của n đều gđ
                        nhãn(n) = "gđ"
                 }2
            else {3 bien = false;
                  repeat { con ← get(B(n));
                           gđ(con); bien = (nhãn(con) == "gđ")
                         }
                  until (bien or B(n) == ∅);
                  if (bien) then      // có một đỉnh con dạng HOẶC của n gđ
                        nhãn(n) = "gđ"
                 }3
}1
```

Tương tự, có thể viết thủ tục *không giải được* ***Kgđ***(x ∈ *S) (bài tập).*

*** *Dạng chung của các thủ tục tìm kiếm trên cây VÀ/HOẶC***

Input : Cây VÀ/HOẶC G = (S,A) với gốc x_o.

Output: Thông báo "Không thành công" nếu x_o *không giải được* và "thành công" nếu x_o *giải được* và khi đó xuất cây lời giải.

Tính *gđ* hay *kgđ* của *gốc được lan truyền ngược từ lá.* Trong các thuật toán của phần này, ta sử dụng tập *DONG* có cấu trúc stack.

```
TìmKiếm_VÀ_HOẶC(x_o, DONG)
{1 MO = {x_o}; DONG = ∅;
  while (MO ≠ ∅) do
  {2 n ← get(MO);                          // lấy n ra khỏi đầu danh sách MO
   Push(n, DONG);
   if (B(n) ≠∅)                            // đỉnh n có con
   then {3 MO ← MO U B(n);
           bool = false;
           while (B(n) ≠ ∅ and not bool) do
           { con ← get(B(n)); bool = (nhãn(con) == "gđ");
           }
         if (bool) then   // nếu có ít nhất 1 con gđ thì xem lại gốc có gđ không ?
                { gđ(x_o);
                  if (nhãn(x_o) == "gđ")
                  then {  write("Thành công");
                          XuấtCâyLờiGiải(x_0, DONG); exit;
                       }
                  else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
                }
        }3
   else                                              // n là đỉnh lá
         if (kt(n))              // n là đỉnh lá gđ, xem lại gốc có gđ không ?
         then { gđ(x_o);
                  if (nhãn(x_o) == "gđ")
                  then {  write("Thành công");
                          XuấtCâyLờiGiải(x_0, DONG); exit;
                       }
                  else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
                }
         else                    // n là đỉnh lá kgđ, xem lại gốc có kgđ không ?
               { Kgđ(x_o);
                  if (nhãn(x_o) == "Kgđ")
                  then exit("Không thành công")
```

else *Loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được*
}
}2
}1

* ***Thuật toán*** *tìm kiếm theo chiều rộng trên cây VÀ/HOẶC*

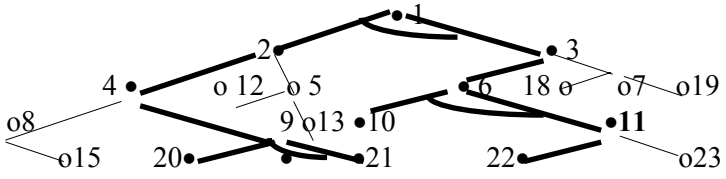
Chính là thuật toán *TìmKiếm_VÀ_HOẶC(x_o, DONG),* với cấu trúc tập *MO* là *hàng đợi.*

Nếu cây lời giải tồn tại thì thủ tục tìm kiếm theo chiều rộng sẽ dừng và cho kết qủa là cây lời giải có độ cao nhỏ nhất.

- *Ví dụ 20*: Xét cây VÀ/HOẶC như trên hình *II.2.5.3*. Thứ tự đưa các nút vào danh sách DONG được chỉ ra bởi các số ở bên cạnh các đỉnh. Các đỉnh tô đậm là các đỉnh giải được. Cây lời giải được phân biệt bởi các cung tô đậm.

Quá trình đưa các đỉnh vào tập MO và DONG như sau:

MO	DONG
1	
2,3	1
3;4,5	1,2
4,5;6,7	1,2,3
5;6,7;8,9	1,2,3,4
6,7;8,9;12,13	1,2,3,4,5
7;8,9;12,13;10,11	1,2,3,4,5,6 (đỉnh 2,3: "gđ")
	(Đỉnh gốc 1 : "gđ" → Kết thúc)



(Hình II.2.5.3)

* ***Thuật toán*** *tìm kiếm theo chiều sâu trên cây VÀ/HOẶC*

Chính là thuật toán *TìmKiếm_VÀ_HOẶC(x_o, DONG),* với cấu trúc tập *MO* là *ngăn xếp.*

* ***Thuật toán*** *tìm kiếm cực tiểu giá thành trên cây VÀ/HOẶC*

Cho cây VÀ/HOẶC $G = (S, A)$ với gốc x_0 và hàm giá c: $A \rightarrow R_+$. Để tìm cây lời giải có giá cực tiểu, ta cần đến khái niệm hàm giá tối ưu *h(n)* của cây lời giải có gốc là đỉnh *n* bất kỳ và hàm giá ước lượng $h^0(n)$ của nó.

Định nghĩa: hàm giá tối ưu h(n) của cây lời giải có gốc n được xây dựng như sau:

- Nếu n là đỉnh lá và

. giải được (đỉnh kết thúc) thì $h(n) = 0$.

. không giải được thì hàm h(n) không xác định

- Nếu n có các đỉnh con $n_1, ..., n_k$ là:

. đỉnh HOẶC thì: $h(n) = \min (c(n, n_i) + h(n_i))$.

. đỉnh VÀ thì:

$$h(n) = \sum_{i=1}^{k} (c(n, n_i) + h(n_i)), \text{ với kiểu giá tổng cộng}$$

$$h(n) = \max_{1 \le i \le n} (c(n, n_i) + h(n_i)), \text{ với kiểu giá cực đại}$$

(Lưu ý rằng h(n) không xác định đối với các đỉnh không giải được)

Mục đích của việc tìm kiếm là xây dựng cây lời giải có giá cực tiểu $h^0(n_o)$.

- Trên thực tế, trong thuật toán ta *cần xác định và sử dụng hàm ước lượng heuristics h^o đối với các đỉnh không phải là đỉnh không giải được*. *Cây lời giải được xây dựng dần* trong quá trình mở rộng cây lựa chọn, tại mỗi thời điểm các nút lá của nó thuộc một trong 3 dạng sau:

a. Các đỉnh lá giải được (đỉnh kết thúc).

b. Các đỉnh lá không giải được (đỉnh không kết thúc và không có con).

c. Các đỉnh chưa được tháo (chưa được xử lý).

Mỗi đỉnh này của cây lựa chọn gọi là <u>*đỉnh mút*</u>.

- Ta <u>*xây dựng ước lượng h^o*</u> của h như sau: (1)

1. n là đỉnh mút

- Nếu n là đỉnh lá giải được thì $h^o(n) = 0$.
- Nếu n là đỉnh lá không giải được thì $h^o(n)$ không xác định.
- Nếu n chưa được xử lý thì để làm giá trị $h^o(n)$ có thể lấy một ước lượng heuristic nào đó của h(n). Ước lượng này dựa trên các thông tin heuristics về bài toán.

2. n không là đỉnh mút: nếu n có các đỉnh con $n_1,..., n_k$ là:

- đỉnh HOẶC thì: $h^o(n) = \min (c(n, n_i) + h^o(n_i))$.
- đỉnh VÀ thì:

$$h^o(n) = \sum_{i=1}^{k} (c(n, n_i) + h^o(n_i)), \text{ với kiểu giá tổng cộng}$$

$$h^o(n) = \max_{1 \le i \le n} (c(n, n_i) + h^o(n_i)), \text{ với kiểu giá cực đại}$$

- Trong cây tìm kiếm, ở mỗi bước có thể chứa một tập các cây con có gốc x_o sao cho chúng trở thành phần trên của cây lời giải đầy đủ. Ta gọi các cây này là cây lời giải tiềm tàng gốc x_o. Dựa vào h^o có thể xây dựng một <u>*cây lời giải tiềm tàng T_o gốc x_o*</u> có nhiều triển vọng trở thành phần trên của cây lời giải thật sự gốc x_o như sau: (2)

1. Đỉnh xuất phát $x_o \in T_o$.

2. Nếu $n \in T_o$ có các đỉnh con $n_1,..., n_k$ là đỉnh :

a. HOẶC thì chọn một con n_{io} đưa vào T_o sao cho: $c(n,n_{io})+h^o(n_{io}) \rightarrow \min$ và *nhãn*$(n_{io}) \neq$ "*kgđ*".

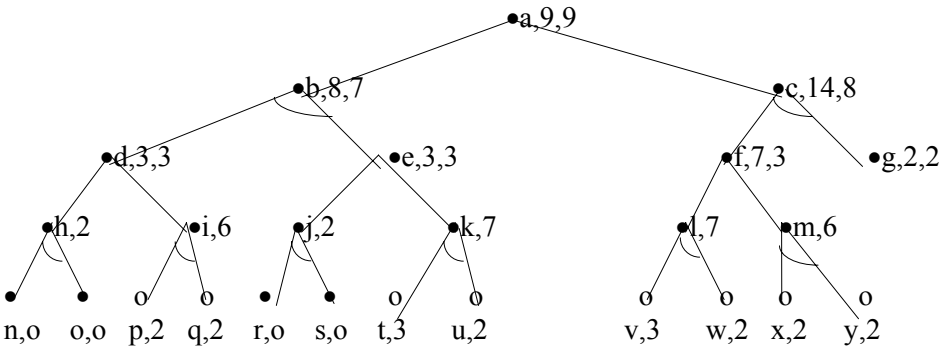
b. VÀ thì đưa tất cả $n_1,..., n_k$ vào T_o nếu *nhãn*$(n_i) \neq$ "*kgđ*" $\forall$ i = 1, .., k

TìmKiếm_VÀ_HOẶC_CựcTiểu

```
{1 MO = {xo}; To ← xo ;
  while (MO ≠∅) do
  {2 n ← Get(MO);                    // lấy n ra khỏi đầu danh sách MO
    DONG ← {n} U DONG;
   if (kt(n))                                          // n là đỉnh lá gđ
     then {      gđ(xo);
```

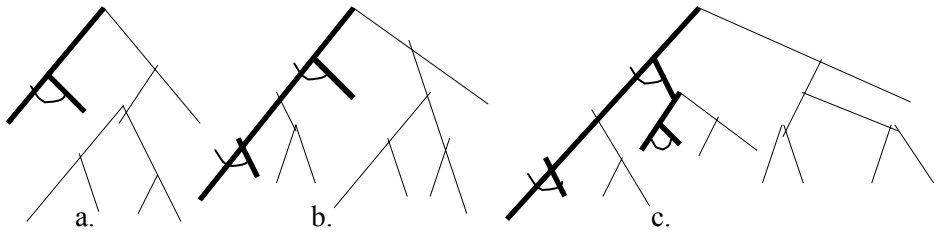
```
                if (nhãn(x_o) == "gđ")
                then {   Xuất cây lời giải tối ưu T_o; exit;
                     }
                else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
            }
    else        if (B(n) ≠∅)                                  // n không là đỉnh lá
                then { for each m ∈ B(n) do Tính h^o(m);       // theo (1)
                         for each m ∈ MO U DONG do Tính h^o(m);
                         MO←MO U B(n); // ChènTăngTập(B(n), MO) theo h^0(n)
                         xd(T_o);  // Xây dựng cây T_o dựa trên n và B(n) theo (2)
                     }
                else                                          // n là đỉnh lá Kgđ
                     { Kgđ(x_o);
                       if (nhãn(x_o) == "kgđ")
                       then exit("Không thành công")
                       else Loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được
                     }
    }2
}1
```

- *Ví dụ 21*: Xét cây VÀ/HOẶC như hình *II.2.5.4*.



(Hình II.2.5.4)

Các số bên cạnh đỉnh là giá trị h và h^o. Đối với các nút mút trong quá trình tìm kiếm các giá trị h^o là các đánh giá heuristics giá tổng cộng của bài toán tương ứng. Lấy $h^o = h$ đối với các nút ở độ sâu 3, 4 (các nút từ h đến y), $h^o(d) = 3$, $h^o(e) = 3$, $h^o(f) = 3$, $h^o(g) = 2$, $h^o(b) = 7$, $h^o(c) = 8$, $h^o(a) = 9$. Quá trình tìm kiếm được chỉ ra trên hình *II.2.5.5 và II.2.5.6*. Cây lời giải tiềm tàng chỉ ra bởi các cung tô đậm.



(Hình II.2.5.5)

Vấn đề cần đặt ra là chọn đỉnh nào trong các đỉnh VÀ của T_o để xử lý. Có thể chọn các đỉnh mút trong T_o có xác suất lớn nhất sao cho có thể dẫn đến bác bỏ giả thuyết rằng T_o sẽ là phần trên của cây lời giải có giá tối ưu. Với giá tổng cộng thì đỉnh có xác suất lớn nhất chứng tỏ sự không phù hợp của T_o là đỉnh có giá trị h^o lớn nhất. Với giá cực đại ta chọn đỉnh MO trong T_o như sau: Xuất phát từ x_o đi trong cây T_o, nếu gặp một đỉnh n nào đó có đỉnh con là đỉnh VÀ thì chọn cung từ n tới đỉnh con n_i sao cho $c(n,n_i) + h^o(n_i) \rightarrow \max$.

Quá trình đưa các đỉnh vào tập MO và DONG như sau:

T_o	MO		DONG
(a,9) /* (a,h^o) */ (a,9)		(a,9)	
(b,7)	(b,7),(c,8);	min	(b,7)
(d,3) ,(e,3)	(d,3) ,(e,3);(c,8);	∀	(d,3)
(h,2)	(h,2),(i,6);(e,3);(c,8);	min	(h,2)
(n,0),(o,0)	(n,0),(o,0);(i,6);(e,3);(c,8)	∀	(n,0),(o,0)
(e,3)	(e,3),(c,8)		(e,3)
(j,2)	(j,2),(k,7);(c,8)	min	(j,2)
(r,0),(s,0)	(r,0),(s,0);(k,7);(c,8)	∀	(r,0),(s,0)
	(→ Kết thúc)		

(Hình II.2.5.6)

II.2.6. Phương pháp GPS (General Problem Solving)

Phương pháp này còn gọi là phương pháp *mục đích - phương tiện* (Ends-Means). PP GPS là sự *tổng quát hóa* của những PP giải quyết vấn đề đối với bài toán biểu diễn vấn đề trong không gian trạng thái đã trình bày.

- *Ý tưởng:* Giả sử bài toán được biểu diễn trong không gian trạng thái S và cần phải đưa ra cách biến đổi từ trạng thái ban đầu $x_o \in S$ về trạng thái cuối $x^* \in S$. Trong các PP dựa trên lôgic hình thức quá trình mở rộng cây tìm kiếm dựa trên các thông tin định lượng: số phép biến đổi đã sử dụng và ước lượng heuristics h^o phản ánh mối liên hệ giữa x_o và x*. Trong khi đó , GPS quan tâm đến *sự khác biệt δ_i giữa x_o và x** (theo nghĩa *định tính lẫn định lượng*) và mối liên hệ giữa chúng với các phép biến đổi trạng thái o_j. Như vậy, giả sử xuất phát từ x_o ta đã đi tới trạng thái x. Xem xét một *khác biệt "quan trọng nhất" δ_i giữa x và x*,* dựa vào đó xác định phép biến đổi o_j nào đó *"hiệu qủa nhất" để biến đổi x sao cho giảm sự khác biệt δ_i nhiều nhất*. Kết qủa nhận được trạng thái mới $o_j(x)$ và cứ như vậy tiếp tục cho đến khi $x \equiv x^*$.

- *Ba yếu tố cơ bản trong PP GPS*:

a. Xác định *không gian trạng thái S, trạng thái đầu $x_o \in S$, trạng thái cuối $x^* \in S$ và tập các phép biến đổi* O từ trạng thái này sang trạng thái khác:

$$O = \{ o_j \mid o_j: S \rightarrow S, j = 1..n \}$$

b. Xác định *tập Δ các kiểu khác biệt* giữa các trạng thái trong không gian

$$\Delta = \{ \delta_i \mid \delta_i : S^2 \rightarrow R, i = 1..m \}$$

c. Xây dựng *ma trận M* với các cột ứng với các toán tử, các hàng ứng với *các sự khác biệt có thể có*

$M = (m_{ij})_{m \times n}$, trong đó:

$$m_{ij} = \begin{cases} 1, \text{ nếu phép biến đổi } o_j \text{ làm giảm sự khác biệt } \delta_i \\ 0, \text{ nếu ngược lại} \end{cases}$$

*(*Có thể cho m_{ij} *những trị khác để phản ánh tốt hơn tác dụng của toán tử* o_j *đến sự khác biệt* δ_i*).*

<u>*Bài toán*</u>

Input: - Tập S các trạng thái, trạng thái đầu x_o, trạng thái cuối x*
- Tập $\Delta = \{ \delta_1, ..., \delta_m \}$ các sự khác biệt
- Tập $O = \{ o_1, ..., o_n \}$ các toán tử
- Ma trận $M = (m_{ij})_{m \times n}$

Output: Ra thông báo "Thành công" nếu đưa được x_o về x*

Thuật giải GPS

```
{ x = x_o; OP = ∅;
  while (x ≠ x*) do
  { D = match(x, x*) ; // D = { δ_i1, ..., δ_ik} là tập sự khác biệt giữa x và x*
    δ ← get(D);          // chọn ra sự khác biệt quan trọng nhất δ = δ_i ∈ D
    for j=1 to n do
        if (m_ij = 1) then OP = OP U {o_j}; // OP ={o_j | m_ij=1}
    o ← get(OP); // chọn toán tử o = o_j ∈ OP làm giảm sự khác biệt δ_i nhiều nhất
    x = o(x);
  }
  if (x = x*) Xuất lời giải;
}
```

Ở đây thủ tục *match(x, x*) cho phép xác định những điểm khác biệt giữa x và x**. Chẳng hạn, sự khác biệt giữa đỉnh x và x* trong đồ thị biểu diễn không gian trạng thái S có thể là độ dài đường đi từ x tới x*.

- <u>*Ví dụ 22*</u>: Cần phải chứng minh sự tương đương logic giữa hai biểu thức mệnh đề:

$$R \wedge (\neg P \Rightarrow Q) \equiv (Q \vee P) \wedge R$$

Áp dụng phương pháp GPS để giải bài toán trên.

. Trong không gian biểu diễn các mệnh đề lôgic xét *4 kiểu khác biệt* sau:

ΔT: Có sự khác biệt về dấu $\neg$ (phủ định) trong biểu thức.

ΔC: Khác nhau về số các phép toán ($\wedge, \vee, \Rightarrow$).

ΔG: Khác nhau về phương pháp nhóm các mệnh đề.

ΔP: Khác nhau về vị trí các thành phần trong hai biểu thức.

. Trong bài toán này, ta lấy c*ác toán tử biến đổi trạng thái là 3 phép biến đổi tương đương* sau:

R1: $A \vee B \Leftrightarrow B \vee A$; $A \wedge B \Leftrightarrow B \wedge A$

R2: $A \rightarrow B \Leftrightarrow \neg A \vee B$

R3: $\neg\neg A \Leftrightarrow A$

. Xét ma trận M sao cho các cột tương ứng với các phép biến đổi : R1-R3.

	R1	R2	R3
ΔT		1	1
ΔC		1	
ΔG			
ΔP	1		

Đặt : $L1 \equiv R \wedge (\neg P \Rightarrow Q)$, $Lo \equiv (Q \vee P) \wedge R$

Đích 1: Biến đổi L1 về Lo

Sự khác biệt: D = {ΔT, ΔP, ΔC}. Với ΔP (ΔP = get(D)), OP = {R1}: chọn R1 áp dụng vào L1 ta được: $L2 = (\neg P \Rightarrow Q) \wedge R$.

Đích 2: Biến đổi L2 về Lo

Sự khác biệt: D = {ΔT, ΔC, ΔP}. Với ΔC (ΔC = get(D)), OP = {R2}: chọn R2 áp dụng vào L2 ta được: $L3 = (\neg\neg P \vee Q) \wedge R$.

Đích 3: Biến đổi L3 về Lo

Sự khác biệt: D = {ΔT, ΔP}. Với ΔP (ΔP = get(D)), OP = {R1}: chọn R1 áp dụng vào L3 ta được: $L4 = (Q \vee \neg\neg P) \wedge R$.

Đích 4: Biến đổi L4 về Lo

Sự khác biệt: D = {ΔT}. Với ΔT (ΔT = get(D)), OP = {R2, R3}: ta chọn R3 mới làm giảm sự khác biệt ΔT của L4, ta được: $L5 = (Q \vee P) \wedge R \equiv Lo$.

Bài toán được chứng minh.

- Ngoài ra, nếu qui bài toán về dạng:

$[R \wedge (\neg P \rightarrow Q)] \Rightarrow [(Q \vee P) \wedge R]$

và $[(Q \vee P) \wedge R] \Rightarrow [R \wedge (\neg P \Rightarrow Q)]$

thì ta có thể giải bài toán bằng: thủ tục tách Wong hay thủ tục hợp giải Robinson sẽ được trình bày trong chương 3.

II.3. Kỹ thuật Heuristic

Những bài toán, đặc biệt là khi chúng có kích thước lớn và có cấu trúc phức tạp, đã có thuật toán khả thi với độ phức tạp không quá đa thức, thường rất hẹp.

- <u>*Hạn chế của thuật toán*</u>: Trên thực tế, ta thường gặp nhiều bài toán:

. hoặc cho đến nay *chưa có thuật toán nào để giải*;

. hoặc đã *có thuật toán để giải nhưng không khả thi* về không gian nhớ và thời gian do độ phức tạp của nó có cấp vượt quá đa thức;

. hoặc *có các phương pháp giải mặc dù được thực tế chấp nhận nhưng lại vi phạm một số tính chất của thuật toán như tính xác định hay tính đúng*. Chẳng

hạn, để chuyển từ bước sơ cấp này đến bước sơ cấp sau đôi khi lại cần bổ sung thêm thông tin kinh nghiệm (rút ra trong quá trình tìm kiếm lời giải thực tế) đặc thù của bài toán mới quyết định được. Những bài toán liên quan đến số thực được giải trên máy tính, nếu chấp nhận lời giải gần đúng hoặc gần tối ưu thì có thể tồn tại nhiều cách giải đơn giản và hiệu quả hơn.

Để giải quyết khó khăn này người ta đã *mở rộng* tính *xác định, tính đúng* của thuật toán và bổ sung *thêm các tri thức kinh nghiệm* đặc thù hay *mẹo giải* để có được các *thuật giải heuristics*.

- *Các đặc trưng của thuật giải heuristic*: *thường* tìm được lời giải *tốt* (không chắc luôn luôn là tốt nhất) hay *gần tốt* một cách *đơn giản, nhanh chóng* và đôi khi độc đáo (so với những thuật toán tương ứng giải cùng một bài toán). Có *2 cách đưa các thông tin heuristics đặc tả vào các thủ tục tìm kiếm:*

. các tri thức được nạp ngay trong biểu diễn của bài toán;

. các hàm đánh giá heuristics nhằm lượng hoá cấp độ của khả năng lựa chọn việc thực hiện.

- *Các kỹ thuật heuristics liên hệ chặt chẽ với chiến lược điều khiển các hướng tìm kiếm lời giải và xử lý cạnh tranh*; chúng được chia thành 2 lớp chính:

. *Định tính*: dưới dạng các luật *nếu ... thì*

. *Định lượng*: dưới dạng các hàm ước lượng heuristics h^o. Các *hàm đánh giá được xem là công cụ có rất hiệu quả* khi sử dụng kỹ thuật heuristic. Hai vấn đề quan trọng trong lập trình heuristic có sử dụng các hàm đánh giá là: *tách những dấu hiệu có ích và tổ hợp các dấu hiệu này để thể hiện thành những hàm đánh giá tốt*. Một kỹ thuật heuristics được coi là hợp lý khi nó *cho phép tiến hành đánh giá các khả năng để làm rõ khả năng nào tốt hơn các khả năng còn lại*.

- *Hàm số heuristics*

Dạng tổng quát của các hàm đánh giá: $f(p) = f(t_1, ..., t_n)$, trong đó ứng với mỗi khả năng p, dạng giải tích của hàm đánh giá f phụ thuộc vào các tham số $t_1, ..., t_n$ và trả về một giá trị $f(t_1, ..., t_n)$ đặc trưng cho khả năng p.

. Trường hợp đơn giản nhất (*dạng afin*): $\boldsymbol{f(t_1, ..., t_n) = \sum a_i.t_i}$, với a_i là các hệ số đặc trưng cho trọng số của các tham số t_i.

. Phức tạp hơn là dạng toàn phương:

$$\mathbf{f(t_1, ..., t_n) = \sum_{i=1}^{n} a_i.t_i + \sum_{j=1}^{n} b_i.t_i^2 + \sum_{i \neq j=1}^{n} c_{ij}.t_i.t_j}$$

- *Quá trình xây dựng hàm heuristic gồm 3 vấn đề chính* sau:

. Xác định các dấu hiệu đặc tả s_i của bài toán;

. Với mỗi khả năng p, các dấu hiệu đặc tả s_i được lượng hoá bởi các giá trị t_i. Xác định dạng của hàm đánh giá $f(p) = f(t_1, ..., t_n)$ dựa trên các giá trị đặc trưng cho các thuộc tính t_i và các tham số;

. Xác định giá trị của các tham số cho từng bài toán cụ thể.

II.3.1. Các thuật giải tìm kiếm tối ưu trên cây và đồ thị với tri thức heuristic

Problem_3**(G = (S, A), x_0, DICH, c, h^0) ***(BT3**)***

a. <u>Thuật giải A^{KT}</u> (Algorithm for Knowledge Tree search) giải *bài toán* ***tối ưu*** *BT3** với tri thức bổ sung* ***heuristic*** *trên* ***cây***

Tương tự thuật toán A^T tìm kiếm cực tiểu trên cây G = (S, A) có gốc x_0 (phần II.2.4.c), nhưng thay hàm $g^0(x)$ bởi $f^0(x) = g^0(x) + h^0(x)$.

- <u>Ý nghĩa</u>:

. $g^0(x)$: *giá chi phí thật sự từ x_0 đến x,* cách tính $g^0(x)$ theo công thức đệ qui: $g^0(x_0) = 0$ & $g^0(con) = g^0(cha) + c(cha, con)$ $\forall$ (cha, con) $\in$ A.

. $h^0(x)$: *ước lượng chi phí từ x đến tập DICH*, hay là *hàm ước lượng khả năng dẫn đến lời giải.*

. Việc dùng hàm $f^0(x) = g^0(x) + h^0(x)$ *khi chọn lựa phương án kế tiếp, không chỉ quan tâm đến chi phí đã trả từ điểm xuất phát x_0 đến trạng thái hiện tại x, mà còn tham khảo thêm cả thông tin heuristic đặc trưng cho khả năng nhanh dẫn đến đích cuối cùng từ trạng thái hiện tại*. Có thể xem các phương pháp này là các *thuật giải vét cạn thông minh*. Đây là cách giải thông minh và độc đáo mà con người thường sử dụng khi gặp các bài toán khó trong thực tế như chứng minh định lý, giải các bài toán suy luận lôgic, chơi cờ, …

- <u>Nhận xét</u>: Có thể xem A^{KT} là tổng quát hoá của A^T khi xét $h^0 \equiv 0$. Để tìm được đường đi tối ưu, ta cần chọn hàm ước lượng $h^0(x) \le h^*(x)$ $\forall x \in S$, với $h^*(x)$ là hàm giá tối ưu thật sự từ *x* đến DICH.

b. <u>Thuật giải A*</u> giải *bài toán* ***tối ưu*** *BT3** với tri thức bổ sung* ***heuristic*** *trên* ***đồ thị*** *tổng quát*

Tập MO, DONG chứa các phần tử E có dạng (x, cha(x), $g^0(x)$, $f^0(x)$), với qui ước: E_x =x, E_{cha} = cha(x), $E_{g0} = g^0(x)$, $E_{f0} = f^0(x)$. ChènTăng(E, MO) là thao tác chèn E vào MO theo thứ tự tăng của $f^0(x) = g^0(x) + h^0(x)$ từ đầu đến cuối danh sách MO.

A*(x_o, DICH)

```
{1 g°(x_o) = 0; Tính f°(x_o) = h°(x_o); MO ← {(x_o, rỗng, 0, f°(x_o))};
  DONG ← ∅;
  while (MO ≠ ∅) do
  {2    E ← get(MO);                    // trong MO, E ở đỉnh và có f°(E_x) nhỏ nhất !
        Push(DONG, E);
        if (E_x ∈ DICH) then { XuấtLờiGiải(E, DONG); Dừng;}
        for each con ∈ B(E_x) do
        {3    g°(con) = g°(E_x) + c(E_x, con); f°(con) = g°(con) + h°(con);
              if (con ∉ (MO U DONG)_x)
              then ChènTăng((con, E_x, g°(con), f°(con)), MO);
              else {4 if (∃ y ∈ MO and con == y_x and g°(con) < g°(y_x))
                        { Loại(y, MO);              // loại y khỏi tập MO
                          ChènTăng((con, E_x, g°(con), f°(con)), MO);
          }
```

```
                        if (∃ y ∈ DONG  and  con == y_x and g°(con) < g°(y_x))
                          { Loại(y, DONG);      // loại y khỏi tập DONG
                            Push((con, E_x, g°(con), f°(con)), DONG);
                            Cập nhật lại g° và f° cho các hậu duệ của con đã lưu
                          }
                      }4
                }3
        }2
        write("Không thành công");
}1
```

c. <u>Các ví dụ</u>

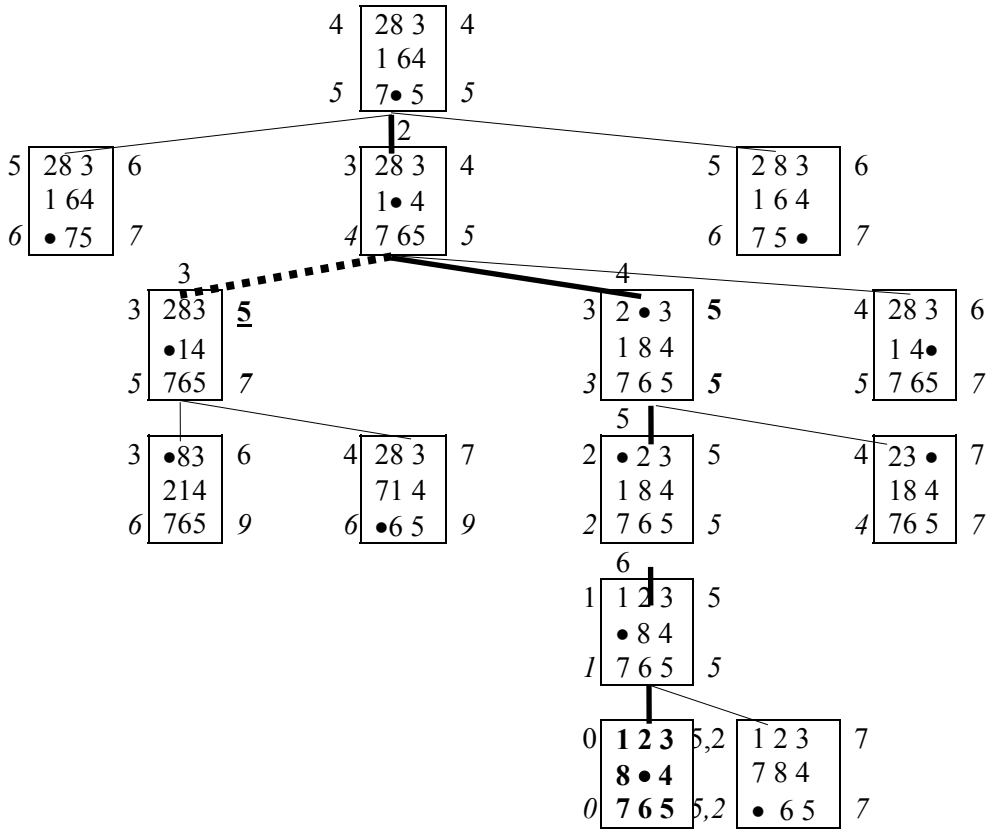
- *<u>Ví dụ 23</u>*: Xét bài toán trò chơi 8 số như trong ví dụ 1 (*hình II.3.1*).

Ta xác định hàm $g^o(x)$ là độ dài đường đi hiện tại từ x_o đến x (số phép dịch chuyển vị trí trống để đưa trạng thái x_o về trạng thái x), còn đối với $h^o(x)$, ta có thể tính ít nhất theo hai cách sau đây.

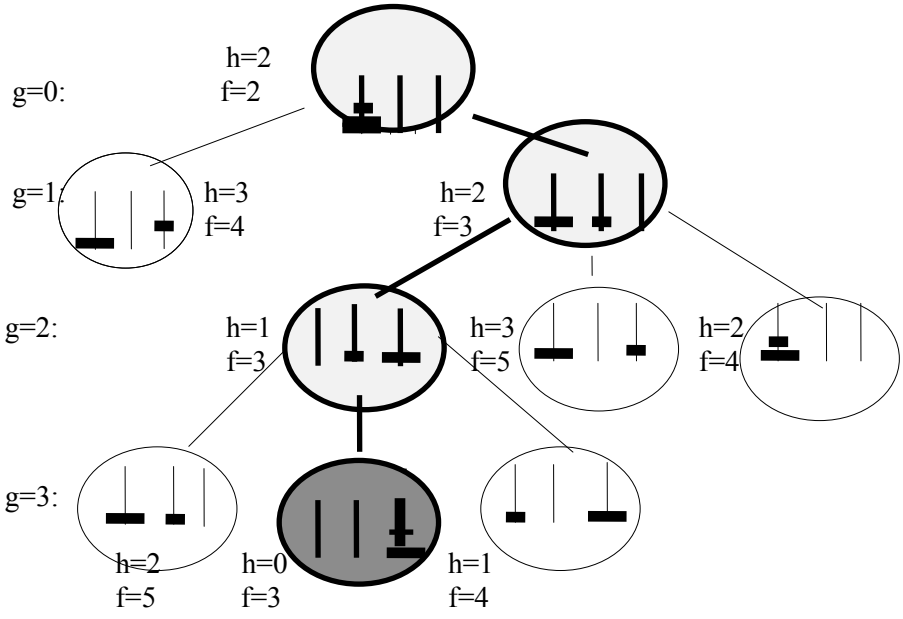
. *Cách 1*: $h^o_1(x)$ là số lượng chữ số (khác trống) trong trạng thái x không nằm ở đúng vị trí của chúng trong trạng thái đích. Chẳng hạn, với đỉnh gốc x_o, $g^o(x_o) = 0$, $h^o_1(x_o) = 4$, do đó $f^o_1(x_o) = g^o(x_o) + h^o_1(x_o) = 4$. Trên *hình II.3.1,* ta có kết qủa áp dụng thủ tục *A** với tri thức bổ sung $h^o_1(n)$ được cho ở góc trên bên trái mỗi trạng thái, giá trị của f^o_1 đối với mỗi nút được cho ở góc trên bên phải của nó. Các số nằm ở đỉnh trên mỗi trạng thái là thứ tự xét chúng trong thủ tục. Với cách chọn hàm ước lượng heuristic không tốt như h^o_1, ta có thể đi nhầm một bước (bước 3).

. *Cách 2*: $h^o_2(x)$ là tổng số các phép di chuyển ít nhất tương ứng với mỗi chữ số (khác trống) trong trạng thái x cần di chuyển đến đúng vị trí của nó trong trạng thái đích. Chẳng hạn, với đỉnh gốc x_o, $g^o(x_o) = 0$, $h^o_2(x_o) = 5$, do đó $f^o_2(x_o) = g^o(x_o) + h^o_2(x_o) = 5$. Trên *hình II.3.1,* ta có kết qủa áp dụng thủ tục *A** với tri thức bổ sung $h^o_2(n)$ được cho ở góc dưới bên trái mỗi trạng thái, giá trị của f^o_2 đối với mỗi nút được cho ở góc dưới bên phải của nó. Với cách chọn hàm ước lượng heuristic h^o_2 tốt hơn h^o_1, ta đi không nhầm bước nào, do h^o_2 *phản ánh sự khác nhau giữa hai trạng thái tốt hơn.* Ta nói *heuristic* h^o_2 *chứa nhiều thông tin hơn heuristic* h^o_1 *nếu:* $h_2^o(n) \geq h_1^o(n)$.

Qua đó, ta thấy *việc sử dụng tri thức bổ sung thông qua hàm* h^o *đóng vai trò rất quan trọng. Nó cho phép gạt bỏ những thông tin không có triển vọng, nhưng nếu* h^o *vượt quá giới hạn h thật sự* thì *có thể thuật toán nhanh dừng song không cho kết quả mong muốn.* Hàm h^o và g^o có ảnh hưởng khá lớn đến sức mạnh heuristics của thuật giải *A**.



(Hình II.3.1)



(Hình II.3.2)

- *Ví dụ 24*: Xét bài toán tháp Hà nội với n = 2 (*hình II.3.2*). Hàm $f^o(n) = g^o(n) + h^o(n)$, với $g^o(n)$ là số đĩa đã di chuyển, $h^o(n)$ là thông tin về mối liên hệ giữa n và trạng thái đích, chẳng hạn:

- Nếu ở cọc C có 2 đĩa, đĩa nhỏ trên đĩa to thì $h^o = 0$.
- Nếu ở cọc C có 1 đĩa to thì $h^o = 1$.
- Nếu ở cọc C chưa có đĩa nào thì $h^o = 2$.
- Nếu ở cọc C có 1 đĩa nhỏ thì $h^o = 3$.

Lời giải tối ưu cho bài toán tháp Hà Nội được trình bày qua cây lời giải in đậm trong *hình II.3.2.*

Một cách *tổng quát*, ta có thể lấy *$f^o(n) = g^o(n) + h^o(n)$, với $h^o(n)$ là tri thức bổ sung chỉ ra triển vọng của đỉnh nằm trên đường tối ưu. Do đó có thể xem $f^o(n)$ là ước lượng của hàm $f(n) = g(n) + h(n)$ với $g(n)$ là giá đường đi tối ưu từ n_o tới n, $h(n)$ là giá đường đi tối ưu từ n tới tập DICH. Nói cách khác $f^o(n)$ là xấp xỉ giá đường đi tối ưu từ n_o tới tập DICH và đi qua đỉnh n. Giá trị $g^o(n)$ đối với n có thể tính được dựa trên ý tưởng thuật giải A* (giá đường đi "gần cực tiểu"từ n_o tới n), còn ước lượng $h^o(n)$ dựa trên thông tin heuristics gắn liền với bài toán*.

Những kết qủa sau bảo đảm tính đúng đắn và tính tối ưu của giải thuật A*.

- *(Tính đúng đắn)*: *Nếu đối với mỗi đỉnh $n \in N$ ta có $0 \le h^o(n) \le h(n)$ và tồn tại số $\Delta > 0$ sao cho đối với mọi cung $a \in A$, $c(a) \ge \Delta$ thì thủ tục A* dừng và cho kết qủa là đường đi p từ n_o tới $n^* \in DICH$ sao cho $g(n^*) \to min$.*

- Giả sử các thủ tục A_i* sử dụng hàm đánh giá:

$$f_i^o(n) = g_i^o(n) + h_i^o(n), \quad i = 1, 2$$

Thêm vào đó, h_2^o thỏa mãn điều kiện $h_2^o(m) - h_2^o(n) \le h(m, n)$, trong đó h(m,n) là độ dài đường đi ngắn nhất từ m tới n và với mọi $n \in N : 0 \le h_1^o(n) \le h_2^o(n) \le h(n)$. Khi đó: số nút đưa vào tập DONG của thủ tục A_2* bao giờ cũng nhỏ hơn số nút đối với thủ tục A_1*.

- Ba nhân tố ảnh hưởng đến sức mạnh heuristics là: giá của đường đi nhận được, số đỉnh phải tháo trong quá trình xử lý, chi phí tính toán giá trị hàm h^o.

- <u>*Nhận xét*</u>:

. Trong số các thủ tục tìm kiếm sử dụng tri thức bổ sung heuristic *h^o, thủ tục sử dụng triệt để các thông tin về các đỉnh ($h^o = h$ và $f^o = g + h$) là tốt nhất và thủ tục A^T ($f^o = g^o$, $h^o = 0$) là tồi nhất*.

. Nếu *chỉ cần tìm một đường đi nào đó tới đỉnh đích* thì có thể không cần tính đến g^o, tức là $f^o = h^o$. Trong một số trường hợp, nếu không sử dụng g^o thì số nút phải tháo có thể tăng đáng kể so với khi sử dụng g^o.

- <u>*Một số heuristics*</u>

. *Lựa chọn theo giai đoạn*: Quá trình tìm kiếm có thể tiến hành theo từng giai đoạn, ở mỗi giai đoạn chỉ giữ lại một tập con nào đó các đỉnh trong tập MO, chẳng hạn các đỉnh với giá trị f^o gần nhỏ nhất.

. *Hạn chế số đỉnh con*: Mỗi lần tháo một đỉnh n, chỉ giữ lại những đỉnh $m \in B(n)$ sao cho $f^o(m)$ thuộc lân cận của f^o nhỏ nhất.

. *Xây dựng dần các đỉnh con*: Mỗi lần tháo đỉnh con chỉ xét tới một vài đỉnh có triển vọng nhất, đầu tiên chỉ chú ý đến đỉnh con quan trọng nhất, còn các đỉnh khác khi nào cần mới đưa ra.

II.3.2. Nguyên lý tham lam

Với thuật giải sử dụng *nguyên lý tham lam* giải *bài toán tối ưu BT3**: ta sử dụng các thuật toán *TìmKiếm* với một ít điều chỉnh: *chỉ chọn lời giải tốt nhất trong một bước kế tiếp* (chứ chưa chắc tốt nhất trong cả quá trình), do đó *không cài đặt cơ chế quay lui. Ưu* điểm của nguyên lý này là trong một số trường hợp, có thể *nhanh thấy lời giải*. Nhưng *nhược* điểm của nó là trong các trường hợp khác, *có thể tồn tại lời giải tối ưu nhưng thuật toán này không tìm ra*. Khi đó, do không dự phòng cơ chế quay lui nên có thể không tìm ra lời giải, mặc dù nó vốn tồn tại ! (*Tham cái lợi trước mắt* mà không để ý đến cái lợi toàn cục và *không dự phòng lưu lại đường đi đã qua để quay lui* khi lạc hướng!)

```
ThamLam(x0, DICH)
{  DONG = ∅;                          // có thể chọn DONG có cấu trúc stack
   MO = {x0};
   if (x0 ∈ DICH) {XuấtLờiGiải(x0, ∅); Dừng;}
   while (MO ≠ ∅)
   {  x = get(MO);                    // lấy một phần tử  x ra khỏi tập MO
      Push(x, DONG);                  // đưa x vào tập DONG
      if (∃phần tử y "tốt nhất" trong B(x))     // luôn tồn tại nếu B(x) ≠ ∅ !
      {  if (y ∈ DICH)
                {XuấtLờiGiải(y, DONG); Dừng;}
         MO = MO U{y};                // đưa y vào tập MO
      }
   }
   write ("Không tìm thấy lời giải");
}
```

Thật ra MO chỉ gồm không quá 1 phần tử ! Vì vậy, có thể thay thuật toán trên bởi:

```
ThamLam2(x0, DICH)
{  DONG = ∅;      // có thể chọn DONG có cấu trúc stack
   if (P(x0))                                  // hay if (x0 ∈ DICH)
             {XuấtLờiGiải(x0, ∅); Dừng;}
   x = x0;
   while (∃phần tử y "tốt nhất" trong B(x))  // luôn tồn tại nếu B(x) ≠ ∅ !
   {  Push (x, DONG);                          // đưa x vào tập DONG
      if (P(y))                                // hay if (y ∈ DICH)
```

```
            {XuấtLờiGiải(y, DONG); Dừng;}
      x = y;
    }
    write ("Không tìm thấy lời giải");
}
```

- <u>*Ví dụ 25*</u>: (*Traveling Saleman)*

Xây dựng một hành trình TOUR có chi phí nhỏ nhất COST cho bài toán đi qua *n* thành phố, với ma trận chi phí C, sao cho bắt đầu từ thành phố *u* và đi qua mỗi thành phố đúng một lần.

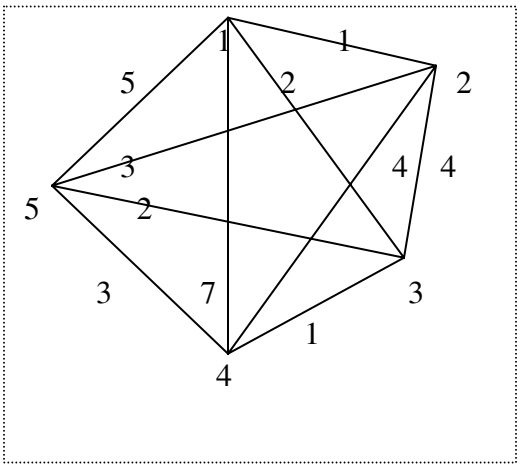
Biến thể của thuật giải *ThamLam2* là *thuật giải GTS* giải bài toán người bán hàng du lịch trên đây.

Thuật giải GTS

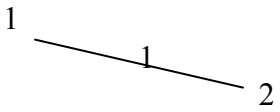
```
{  // Khởi tạo
     TOUR  = ∅; COST = 0;
     v = u;
  // thăm tất cả các thành phố
     for  k = 2 to n do
     {  // Chọn cạnh kế tiếp <v, w> là cạnh có chi phí nhỏ nhất từ v đến
        // các đỉnh chưa  được sử dụng w:
          TOUR = TOUR + <v, w>;
          COST  = COST + C(v, w);
          v = w;                              // khi đó w đã được sử dụng
     }
  // Chuyến đi hoàn thành
          TOUR = TOUR + <v, u>;
          COST = COST + C(v, u);
}
```

Minh họa việc thực hiện:

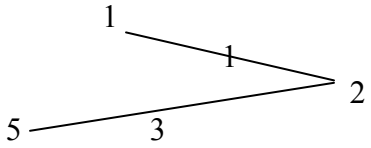
$$C = \begin{bmatrix} \infty & 1 & 2 & 7 & 5 \\ 1 & \infty & 4 & 4 & 3 \\ 2 & 4 & \infty & 1 & 2 \\ 7 & 4 & 1 & \infty & 3 \\ 5 & 3 & 2 & 3 & \infty \end{bmatrix}$$



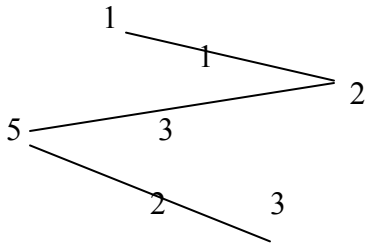
1. TOUR = ∅; COST = 0; u = 1;
   ⇒ w = 2;



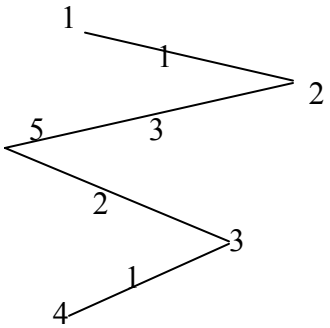
2. TOUR = <1,2>; COST = 1; u = 2;
   ⇒ w = 5;



3. TOUR = {<1,2>, <2,5>}; COST = 4; u = 5;
   ⇒ w = 3;



4. TOUR = {<1,2>, <2,5>, <5,3>}
   COST = 6; u = 3;
   ⇒ w = 4;



5. TOUR = {<1,2>, <2,5>, <5,3>, <3,4>}; COST = 7;
   u = 1;
   TOUR = {<1,2>, <2,5>, <5,3>, <3,4>, <4,1>}; COST = 14;

Tìm min trong các phần tử còn lại của hàng. Xuất phát ở cột nào thì bỏ cột đó

1	*2*	*4*	*5*	*3*
∞	1	2	7	5
1	∞	4	4	3
2	4	∞	1	2
7	4	1	∞	3
5	3	2	3	∞

Do không quay lui nên lời giải vừa tìm chưa chắc là đường đi ngắn nhất. Thật vậy, lời giải vừa tìm có chiều dài lớn hơn đường đi:

TOUR2 = {<1, 3>, <3, 4>, <4, 5>, <5, 2>, <2, 1>} với COST2 = 10.

II.3.3. Nguyên lý hướng đích, phương pháp leo núi (Hill-Climbing)

- Với thuật giải sử dụng *nguyên lý hướng đích* hay *leo đồi (Hill climbing)* giải *bài toán tối ưu BT3*,* ta sử dụng các thuật toán *TìmKiếm* với một ít điều chỉnh, trong đó *không cài đặt cơ chế quay lui ! Ưu* điểm của nguyên lý này là trong một số trường hợp, có thể *tìm thấy rất nhanh lời giải*. Nhưng *nhược* điểm của nó là trong nhiều trường hợp khác, *có thể tồn tại lời giải tối ưu nhưng thuật giải này không tìm ra,* hoặc nó *chỉ tìm thấy lời giải tối ưu địa phương! (Chỉ chú ý đến tối ưu địa phương trước mắt, không dự phòng lưu lại đường đã đi qua* để quay lui khi lạc hướng*, không biết lùi một bước để tiến nhiều bước hơn!)*

```
LeoĐồi(x0, DICH)
{   DONG = ∅;                          // tập DONG có cấu trúc stack
    MO = {x0};
    if (P(x0))                                  // hay if (x0 ∈ DICH)
              {XuấtLờiGiải(x0, ∅); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                     // lấy một phần tử  x ra khỏi tập MO
      Push(x, DONG);                   // đưa x vào tập DONG
      if (∃phần tử  tốt nhất y ∈ B(x) mà "tốt hơn" x)
                                       // chưa chắc tồn tại dù B(x) ≠∅ !
      { if (P(y) đúng)                 // hay if (y ∈ DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
        MO = MO U{y};                           // đưa y vào tập MO
      }
    }
    write ("Không tìm thấy lời giải");
}
```

Thật ra MO chỉ gồm không quá 1 phần tử ! Vì vậy, có thể thay thuật toán trên bởi:

```
LeoĐồi2(x0, DICH)
{   DONG = ∅;                                   // tập DONG có cấu trúc stack
    x = x0; Push(x, DONG);
    if (P(x0))  {XuấtLờiGiải(DONG); Dừng;}
    while (B(x) ≠∅  and ∃phần tử  tốt nhất y ∈ B(x) mà "tốt hơn" x)
                              //chưa chắc tồn tại dù B(x) ≠∅ !
```

```
    { Push(y, DONG);                    // đưa x vào tập DONG
      if (P(y))                         // hay if (y ∈ DICH)
              { XuấtLờiGiải(DONG); Dừng; }
      x = y;
    }
    write ("Không tìm thấy lời giải");
}
```

- Một *biến thể của thuật giải LeoĐồi_2* là thủ tục sau:

Hill_Climbing

{
- Bước 1: n = Đỉnh xuất phát;
- Bước 2: if (n ∈ DICH) then Dừng;
- Bước 3: else *Chọn* $h^0(n_i)$ *bé nhất* trong số mọi đỉnh con $n_i \in B(n)$
  *(*gọi đỉnh này là *Next(n));*
- Bước 4: if $(h^0(n) < h^0(Next(n))$ then Dừng;
  else n = Next(n);
- Bước 5: Goto 2;

}

- <u>Nhận xét</u>: Để *khắc phục nhược điểm* của thuật toán: *nhanh rơi vào lân cận của tối ưu địa phương* mà không thoát ra được, ta có thể kết hợp thêm *thuật giải di truyền GA.*

II.3.4. Nguyên lý sắp thứ tự, nguyên lý trùng khớp nhất

Ta minh họa các nguyên lý trên qua bài toán phân công công việc trên các máy và bài toán xếp hàng vào container.

- <u>*Ví dụ 26*</u>: (Bài toán *phân công công việc*)

Giả sử có n máy: $P_1, \ldots, P_n$ và m công việc: $J_1, \ldots, J_m$. Các công việc được tiến hành đồng thời và có thể được thực hiện trên bất kỳ máy nào. Mỗi lần một công việc được đưa vào máy, máy chỉ dừng khi công việc hoàn tất. Thời gian hoàn tất công việc J_i là t_i. Thời gian nạp công việc vào máy là 0.

Vấn đề đặt ra là: Bố trí công việc vào các máy sao cho tổng thời gian xử lý hết mọi công việc là bé nhất.

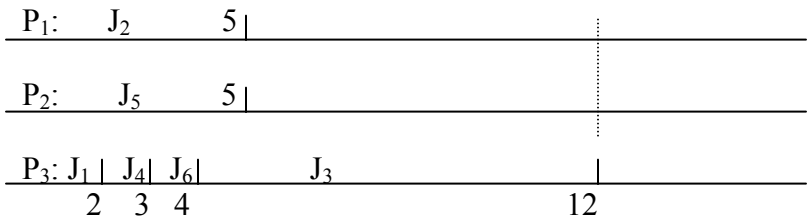
<u>*Mô hình 1*</u>:

. 3 máy: P_1, P_2, P_3.

. 6 công việc

J_i	1	2	3	4	5	6
t_i	2	5	8	1	5	1

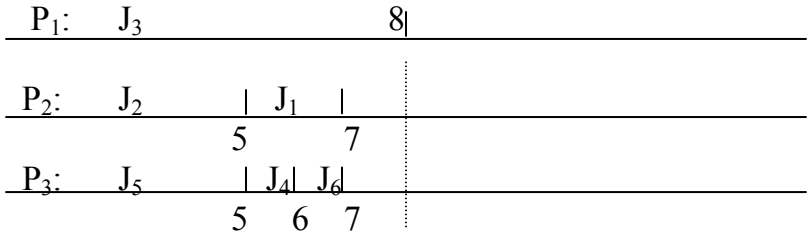
Chọn phương án: $(J_2, J_5, J_1, J_4, J_6, J_3)$; T = 12.



**** Nguyên lý sắp thứ tự:*** *Sắp xếp công việc theo thứ tự thời gian giảm dần.*

Bố trí công việc theo thứ tự trên: $L^* = (J_3, J_2, J_5, J_1, J_4, J_6)$; T = 8.

Đây là lời giải tối ưu.



Mô hình 2(*Phản ví dụ về nguyên lý sắp thứ tự không cho lời giải tối ưu*)

. 2 máy: P_1, P_2.

. 5 công việc

J_i	1	2	3	4	5
t_i	3	3	2	2	2

Phương án:$(J_1, J_2, J_3, J_4, J_5)$ không tối ưu (sắp giảm dần theo thời gian): T=7.

Phương án tối ưu là: $L^* = (J_1, J_3, J_4, J_2, J_5)$; T = 6.

- *Ví dụ 27.1*: Bài toán đóng gói (*xếp hàng vào container)*

Cho dãy các gói hàng:1, 2, 3, 4, 6, 7, 7, 9; kích thước container: 13.

Vấn đề đặt ra: cần xác định số container ít nhất để chứa hết hàng.

- *Các cách (heuristics) xếp* thùng *khác nhau* vào container:

H1: - Thứ tự: tùy ý
- Cách xếp tùy ý.

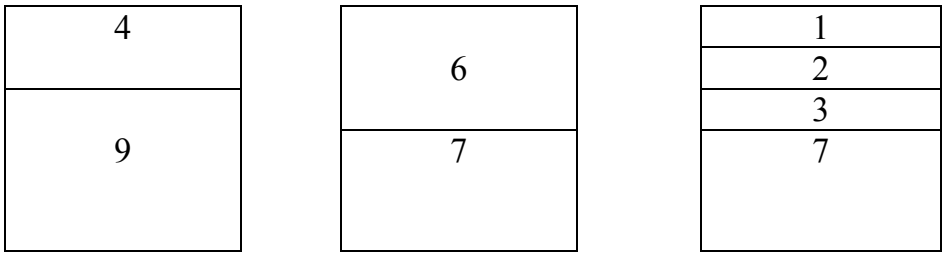
H2: - Thứ tự: Giảm dần theo kích thước
- Cách xếp: Lớn trước, nhỏ sau.

Nguyên lý trùng khớp nhất

H3: - Thứ tự: tùy ý
- Ưu tiên xếp trùng khít.

H4: - *Ưu tiên xếp trùng khít.*
- Sau đó xếp theo *Thứ tự: Giảm dần của kích thước*

Áp dụng nguyên lý này vào ví dụ trên, ta được phương án tối ưu: (9, 7, 7; 4, 6, 3; , , 2; , , 1;) chứa trong 3 container.



- *Ví dụ 27.2*: (*Phương án thu được từ nguyên lý trùng khít nhất không là phương án tối ưu !*)

Cho dãy các gói hàng:2, 2, 5, 6, 8, 9; kích thước container: 16.

. Phương án thu được từ nguyên lý trùng khít nhất: cần 3 thùng

. Phương án tối ưu chỉ cần 2 thùng ! (*Bài tập*)

- *Nhận xét*: Đối với các bài toán phức tạp và đa dạng trong thực tế, nếu *chỉ áp dụng riêng một* phương pháp, ý tưởng hay một nguyên lý heuristic riêng rẽ thì *không thể giải quyết nổi*. Khi đó ta cần áp dụng *chiến lược lai nhiều phương pháp, ý tưởng và nguyên lý giải một cách phù hợp*.

Bài tập

1. *Tìm biểu thức giải tích tường minh từ các hệ thức truy hồi* sau:
   a. $\begin{cases} {}^{+}a_n = a_{n-1} + 2^n, & \forall n \geq 1 \\ a_0 = 1 \end{cases}$
   b. $\begin{cases} {}^{+}a_n = 2a_{n-1} + 1, & \forall n \geq 1 \\ a_0 = 1 \end{cases}$
   c. $*F_n = b_1 F_{n-1} + b_2 F_{n-2}, \quad \forall n \geq 2$ với $\{F_0, F_1\}$, b_1, b_2 cho trước (*trường hợp tổng quát*)
   d. $^{+}$Dãy số *Fibonacci*:
   $\begin{cases} F_0 = 0, \ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2}, & \forall n \geq 2 \end{cases}$
   e. $\begin{cases} *u_n - 4u_{n-1} + 4u_{n-2} = 2n, & \forall n \geq 2 \\ u_0 = 0, \ u_1 = 1 \end{cases}$
   f. $\begin{cases} *T(n) = 2T(n/2) + n - 1, & \forall n \geq 2 \\ T(1) = 1 \end{cases}$
   g. $\begin{cases} u_n = u_{n-1} - u_n u_{n-1}, & \forall n \geq 1 \\ u_0 = 1 \end{cases}$
   h. $\begin{cases} **c_0 = 0, \ c_1 = 0 \\ c_n = n + 1 + \frac{2}{n}\sum_{k=1}^{n-1} c_k, & \forall n \geq 2 \end{cases}$

2. Thể hiện phương pháp *"chia để trị"bằng các thuật toán đệ qui* để giải các bài toán sau:
   a. $^{+}$Tìm min và max của một dãy số.
   b. *Tráo đổi hai phần của mảng (không nhất thiết có độ dài bằng nhau) mà không dùng mảng phụ.

Viết *thuật toán* (bằng *nhiều phương pháp khác nhau*, nếu có thể) *và cài đặt chương trình* để giải các bài toán sau (trong đó có các ví dụ đã được giới thiệu trong lý thuyết):

(Nguyên lý qui hoạch động)

3. $^{+}$Ví dụ 2: bài toán người giao hàng.
4. Ví dụ 3 (bài toán *sắp ba lô có giá trị lớn nhất*): Có *n* loại đồ vật có khối lượng $\{a_i\}$ và giá trị $\{c_i\}$ vào một ba lô có khối lượng w (nguyên dương) . Giả sử mỗi loại đồ vật có đủ nhiều để xếp. Hãy tìm cách xếp sao cho đạt giá trị cao nhất.
5. Tìm dãy các hệ số của nhị thức.
6. Bài toán đổi tiền: Có n loại tiền A_1, A_2, …, A_n (nguyên dương). Hãy tìm cách dùng các loại tiền này để có được số tiền L cho trước (nguyên dương và không bé hơn loại tiền nhỏ nhất) sao cho tổng số tờ là ít nhất.

(Phương pháp vét cạn đơn giản hoặc vét cạn có quay lui)

7. [+]Ví dụ 4.
8. Bài toán 8 quân hậu.
9. [+]Bài toán mã đi tuần (chú ý có thể sử dụng *phần tử cầm canh*).
10. *Tìm các tập con của dãy A các số nguyên dương sao cho có tổng bằng một số nguyên dương M cho trước (*sắp vừa khít* các đồ vật vào *ba lô*).

(Nguyên lý mắt lưới)

11. [+]Ví dụ 5: tìm nghiệm gần đúng của phương trình f(x) = 0 trên [a, b], với f(x) liên tục trên [a,b].

(Phương pháp sinh và thử)

12. [+]Ví dụ 6: bài toán tìm các bộ chỉnh hợp lặp.
13. Tìm các bộ tổ hợp, chỉnh hợp (không lặp).

(Phương pháp nhánh cận)

14. [+]Ví dụ 7: bài toán người du lịch.

(Phương pháp Monte-Carlo)

15. [+]Ví dụ 8: *tìm số* π từ công thức tính diện tích hình tròn hoặc thể tích hình cầu.
16. [+]Ví dụ 9: Kiểm tra *giả thuyết Fermat* bằng thực nghiệm.
17. [+]*Tính gần đúng diện tích hình phẳng* giới hạn giữa trục hoành và đồ thị hàm số liên tục không âm trên đoạn [a, b].

(Thuật giải di truyền - GA)

18. *Ví dụ 10: giải phương trình $x^2 = 25$.

(Nguyên lý mê cung)

19. [+]Ví dụ 11: bài toán mê cung.

20. [+]Dựa trên thuật toán *TìmKiếm* nhằm tìm lời giải đầu tiên cho bài toán *BT3*, ta có thể cải biên để thu được thuật toán vét cạn dưới dạng lặp *VétCạn để tìm mọi lời giải của bài toán sau (bài tập):*

> - <u>Bài toán 4</u>: *Problem_4(G = (S, A), x_0, DICH)* *(BT4)*
> *Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$.*
> *Hãy tìm <u>mọi</u> đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

(Các phương pháp tìm kiếm trong không gian trạng thái theo chiều rộng, chiều sâu, sâu dần, cực tiểu)

21. [+]Ví dụ 12: bài toán *Toci* hay *$n^2 - 1$ số.*

(Các phương pháp tìm kiếm trên đồ thị VÀ/HOẶC theo: chiều rộng, chiều sâu, sâu dần, cực tiểu)

22. $^{+}$Ví dụ 13: bài toán tháp Hà Nội với n = 2, 3.

(Phương pháp GPS)

23. Bài toán chứng minh tính hằng đúng của các biểu thức lôgic sau:
    a. $^{+}$ $[a \rightarrow (b \rightarrow c)] \Leftrightarrow [c \vee \neg b \vee \neg a]$
    b. *$[a \wedge (a \vee c \rightarrow b) \wedge (a \wedge b \rightarrow c \wedge d)] \Rightarrow d$

(Phương pháp heuristic)

24. *Tìm thuật giải heuristics* để giải bài toán *tháp Hà Nội* với:
    a. $^{+}n = 3$
    b. *$n \geq 2$ bất kỳ.

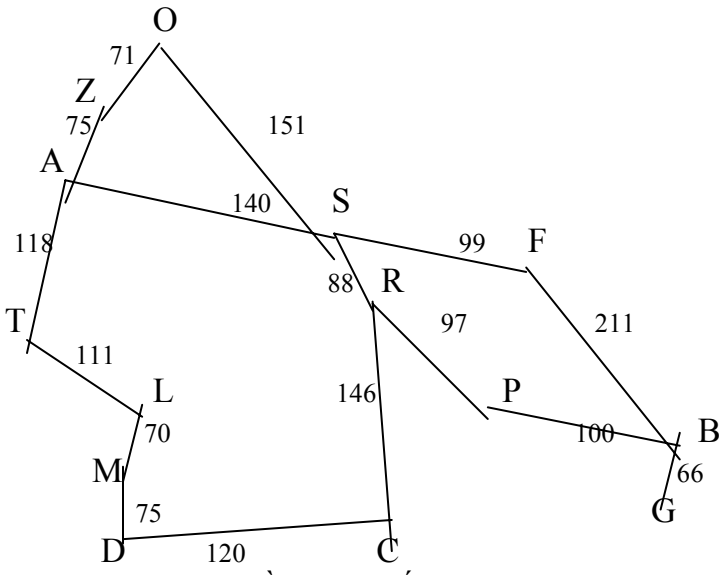
25. $^{+}$Bài toán *Toci* hay *$n^2 - 1$ số*

.

26. $^{+}$*Tìm đường đi ngắn nhất từ* thành phố *A đến* thành phố *B* dựa trên bản đồ các thành phố sau đây, trong đó ngoài các khoảng cách thật sự giữa hai thành phố được chỉ ra bởi các số trên các cung, còn có dãy h^0 chứa chiều dài ước lượng từ một thành phố bất kỳ đến B là khoảng cách đường chim bay từ thành phố đó đến thành phố B.

A	B	C	D	F	G	L	M	O	P	R	S	T	Z
366	0	160	242	178	77	244	241	380	98	193	253	329	374

Dãy h^0 chứa khoảng cách ước lượng từ các đỉnh đến B



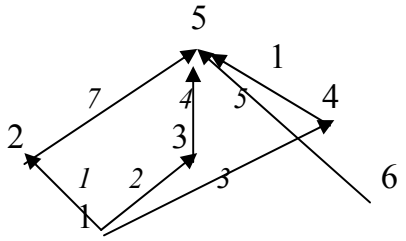
(Bản đồ thành phố)

(Nguyên lý tham lam)

27. +Ví dụ 25: bài toán người du lịch.
28. *Bài toán tô màu (trên đồ thị).

(Nguyên lý leo đồi)

29. Tìm đường đi từ đỉnh 1 đến một trong các đỉnh thuộc tập DICH = {5} trên đồ thị sau với giá thành lớn nhất:



(Nguyên lý sắp thứ tự và nguyên lý trùng khớp nhất)

30. +Ví dụ 26: bài toán phân công công việc.
31. +Ví dụ 27: bài toán sắp xếp vào container.

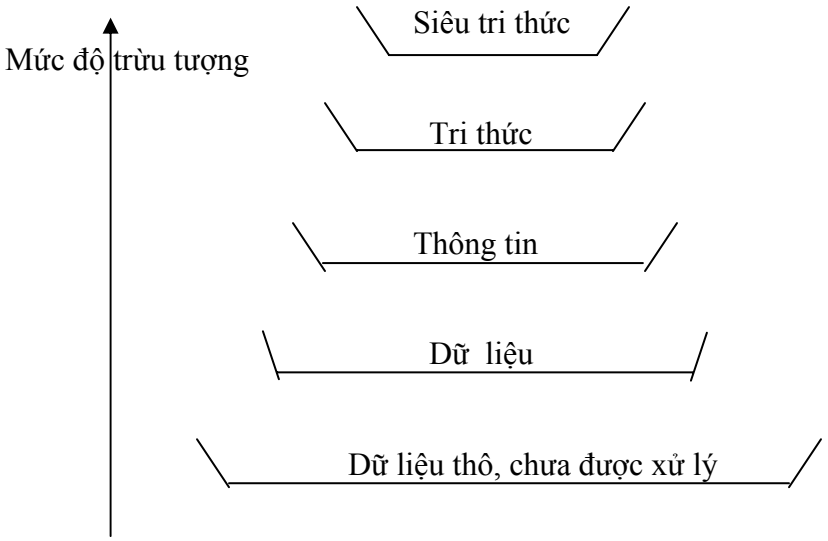
Chương III

BIỂU DIỄN VÀ XỬ LÝ TRI THỨC

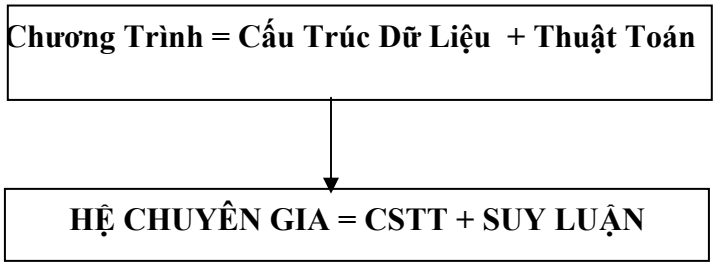
III.1. Khái niệm về biểu diễn và xử lý tri thức

III.1.1. *Từ dữ liệu đến tri thức*

Dữ liệu → Thông tin → Tri thức → Siêu tri thức và Cơ sở tri thức



- *Dữ liệu (Data)*: được biểu diễn dưới dạng chuỗi số, ký tự hay hỗn hợp cả hai loại mà tự bản thân nó không có ý nghĩa độc lập.
- *Thông tin (Information)*: là dữ liệu được tổ chức có đầy đủ ý nghĩa đối với người nhận nó.
- *Tri thức (Knowledge)*: là những thông tin liên quan đến một vấn đề nào đó được tổ chức để có thể giải quyết được vấn đề. *Siêu tri thức (Meta Knowledge)* là loại tri thức ở mức cao, nó có tính khái quát và trừu tượng hơn tri thức.
- *Cơ sở tri thức (CSTT, Knowledge Base)*: là tập hợp các tri thức liên quan đến một vấn đề được sử dụng trong một hệ thống trí tuệ nhân tạo. Nó không chỉ bao gồm các *sự kiện* mà còn chứa các *luật suy diễn* và nó có thể *được sắp xếp lại* khi có thêm một tri thức mới làm thay đổi mối liên hệ giữa chúng. *Hệ chuyên gia (Expert system)* là một hệ cơ sở tri thức (*Knowledge-based system*) được xây dựng từ tri thức của các chuyên gia trong một lĩnh vực nào đó.



III.1.2. ***Một số đặc trưng của tri thức****:* tự giải thích nội dung, có cấu trúc (sự phân cấp giữa các khái niệm và mối quan hệ giữa chúng), có tính liên hệ giữa các tri thức, tính chủ động.

III.1.3. ***Phân loại tri thức***

Có thể phân loại tri thức theo các quan niệm sau:

* *Tri thức tất định và bất định*:

- Tri thức *tiền định (*hay *tất định*) : độc lập với cảm giác, có tính phổ quát và phi mâu thuẫn
- Tri thức *tất yếu (*hay *bất định)*: thu được từ kinh nghiệm mà tính đúng – sai của nó chỉ có thể kiểm tra bằng cảm giác hay kinh nghiệm với độ tin cậy nào đó và có thể có một "mức độ mâu thuẫn" nào đó.

* *Tri thức tồn tại dưới hai dạng cơ bản*:

- *Tri thức định lượng*: thường gắn với các loại heuristics khác nhau, nó phụ thuộc vào chất lượng các hàm đánh giá heuristic và có thể được dùng làm cơ sở để chọn các chiến lược điều khiển.
- *Tri thức định tính gồm 3 dạng chính*:

. *Tri thức mô tả (*hay *khai báo):* cho những thông tin về một sự kiện, hiện tượng hay quá trình mà *không đưa ra thông tin về cấu trúc bên trong cũng như phương pháp sử dụng bên trong tri thức* đó. Nó cho phép *mô tả các mối liên hệ, ràng buộc* giữa các đối tượng.

. *Tri thức thủ tục:* cung cấp các phương pháp *cấu trúc tri thức, ghép nối, suy diễn và tổng hợp các tri thức mới từ các tri thức đã có*. Nó tạo cơ sở cho công nghệ xử lý tri thức: suy diễn, qui diễn, qui nạp, học tri thức.

. *Tri thức điều khiển:* dùng để *điều khiển, phối hợp các nguồn tri thức thủ tục và tri thức mô tả* khác nhau.

III.1.4. ***Các phương pháp biểu diễn tri thức****:* thông qua *lôgic* (lôgic cổ điển hay tất định: lôgic mệnh đề hay vị từ; lôgic bất định: lôgic tình huống, lôgic xác suất, lôgic khả xuất, lôgic mờ), *luật sinh, mạng ngữ nghĩa, khung (frame)*, ...

III.1.5. ***Các phương pháp xử lý tri thức***: *chuyển đổi, suy luận (*chứng minh tự động, các phương pháp giải quyết vấn đề, …), *tổng hợp* tri thức (liên kết, sắp xếp, khái quát hoá, …).

Trong bất kỳ một *hệ thống biểu diễn tri thức đều có chứa 3 yếu tố: ngôn ngữ biểu diễn, cơ chế suy dẫn và công cụ tạo lập cơ sở tri thức cho từng lĩnh vực cụ thể.*

III.2. Một số phương pháp biểu diễn tri thức

III.2.1. Biểu diễn tri thức nhờ lôgic

a. <u>Lôgic cổ điển (hay tất định)</u> gồm : lôgic mệnh đề, lôgic vị từ

* <u>*Lôgic mệnh đề*</u>:

- Xét các mệnh đề p chỉ có thể *nhận một trong hai giá trị lôgic*: đúng (True, 1) hoặc sai (False, 0).
- *Các phép toán lôgic* nối kết các mệnh đề: phép hội (and, $\wedge$, và), phép tuyển (or, $\vee$, hoặc), phủ định (not, $\neg$, không), kéo theo ($\Rightarrow$, $\rightarrow$), tương đương ($\Leftrightarrow$, $\leftrightarrow$, $\equiv$), …

- Vài *phép biến đổi tương đương*:

. $(a \leftrightarrow b) \Leftrightarrow ((a \rightarrow b) \wedge (b \rightarrow a))$

. $(a \rightarrow b) \Leftrightarrow (\neg a \vee b) \Leftrightarrow (\neg b \Rightarrow \neg a)$

$\neg(\underset{1\le i\le n}{\Lambda} a_i) = \underset{1\le i\le n}{V}(\neg a_i); \neg(\underset{1\le i\le n}{V} a_i) = \underset{1\le i\le n}{\Lambda}(\neg a_i) : (DeMorgan)$

. $a \wedge (b \vee c) \Leftrightarrow (a \wedge b) \vee (a \wedge c)$; $a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$: *phân phối*

. Các phép toán $\wedge$, $\vee$ có tính *giao hoán , kết hợp, lũy đẳng.*

- Các phép toán lôgic được thực hiện theo *thứ tự ưu tiên giảm dần* sau: phủ định, hội, tuyển, kéo theo, tương đương.

- Mọi biểu thức logic mệnh đề đều có thể *đưa về dạng biểu thức tương đương chỉ chứa các phép $\wedge$, $\vee$* và *$\neg$ dưới dạng:*

. *chuẩn hội:*

$$\underset{1\le i\le n}{\Lambda}\left(\underset{1\le ki\le m_i}{V} a_{ki}\right)$$

. *chuẩn tuyển:*

$$\underset{1\le i\le n}{V}\left(\underset{1\le ki\le m_i}{\Lambda} a_{ki}\right)$$

trong đó: $a_{ki} = p_{ki}$ hoặc $a_{ki} = \neg p_{ki}$, với p_{ki} là mệnh đề đơn.

- *Giá trị chân lý* của một biểu thức có thể được tính dựa trên bảng chân lý. Nếu biểu thức có *n* biến mệnh đề khác nhau thì để xác định giá trị chân lý của nó ta phải xét đến 2^n bộ giá trị của các biến mệnh đề (hạn chế là sẽ bị bùng nổ tổ hợp).

- Để *chứng minh một biểu thức lôgic mệnh đề là đồng nhất đúng*, ta có thể dùng: bảng chân lý, các phép biến đổi tương đương, số lôgic, phương pháp GPS, các thuật toán: Robinson, Wong Havard (Vương Hạo), suy diễn tiến, suy diễn lùi mà ta sẽ xét trong phần sau.

- Một *luật suy diễn ở* ***dạng chuẩn Horn*** được biểu diễn như sau:

$$p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q_1 \vee .... \vee q_m \qquad (1)$$

Ta thường gặp các trường hợp đặc biệt sau:

. n = 0, m = 1: biểu diễn các *sự kiện (facts*)

$\square \rightarrow q = F(t_1, ..., t_k)$;

. n ≥ 1, m = 1: ta có ***dạng chuẩn*** *sơ cấp dùng để biểu diễn các luật (rules):*

$$p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q \qquad (2)$$

Mọi luật (1) đều có thể biểu diễn dưới dạng (2):

$(1) \quad \Leftrightarrow \quad (p_1 \wedge .... \wedge p_n \wedge \neg q_2 \wedge .... \wedge \neg q_m \rightarrow q_1\})$

- Quá trình suy diễn có thể dựa trên 2 nguyên lý:

$$\text{Modus Ponens}: \frac{A, A \rightarrow B}{B}$$

$$\text{Modus Tollens}: \frac{\neg B, A \rightarrow B}{\neg A}$$

- *Cơ sở tri thức biểu diễn bằng logic mệnh đề*: dựa trên tập các sự kiện *Facts* và tập các luật suy diễn *Rules* (tương tự cách xây dựng một chương trình trong ngôn ngữ lập trình PROLOG).

* <u>*Lôgic vị từ*</u>:

- *Vị từ p(*$x_1, x_2, ... , x_n$*)* là một phát biểu chứa các biến mệnh đề $x_1, x_2, ... , x_n$ sao cho khi chúng nhận các giá trị lôgic cụ thể thì nó cũng nhận một trong hai giá trị lôgic đúng hoặc sai.

- Trong logic vị từ, *ngoài các phép toán trong logic mệnh đề*, người ta còn sử dụng *lượng từ tồn tại (*ký hiệu là *∃)* và *lượng từ với mọi (*ký hiệu là *∀)*.

- <u>*Ví dụ 1*</u>: Vị từ p(x, y, z) biểu diễn đẳng thức: x.y = z, cho ta một vị từ của 3 biến thực x, y, z. Tính chất giao hoán của phép nhân được diễn tả:

$$\forall x, y \;\; p(x,y,z) \Rightarrow p(y,x,z)$$

- Logic vị từ cho phép *diễn đạt hầu hết các khái niệm và nguyên lý khoa học* cơ bản, nhất là toán học và vật lý. Có 2 phạm vi ứng dụng cơ bản của lôgic hình thức là: giải quyết vấn đề và chứng minh định lý tự động.

- Ngoài việc dùng lôgic vị từ thông thường (lôgic vị từ bậc 1), người ta còn nghiên cứu thêm lôgic vị từ bậc cao. Lôgic vị từ bậc n là lôgic vị từ mà các biến vị từ của nó thuộc lôgic vị từ bậc n-1.

* *Ưu điểm* của phương pháp lôgic:

- Là *ngôn ngữ biểu diễn kiểu mô tả*;
- Có *khả năng suy diễn với các cơ chế* quen thuộc: *modus ponens, modus tollens*;
- Khá *trực quan* đối với người sử dụng;
- Khá *gần gũi về mặt cú pháp* với các câu lệnh của *ngôn ngữ lập trình logic* như PROLOG;
- Có thể *dùng để mô tả cấu trúc mô hình và xử lý động mô hình*;
- Có thể *kiểm tra tính mâu thuẫn* trong cơ sở tri thức;
- Tính *môđun hoá cao*, do đó có thể thêm, bớt, sửa các tri thức khá độc lập với nhau và có cả cơ chế suy diễn.

* *Nhược điểm* của phương pháp lôgic:

- *Mức độ hình thức hoá quá cao*, dẫn tới *khó hiểu ngữ nghĩa* của các vị từ khi xem xét chương trình.
- *Năng suất xử lý thấp* (khó khăn cơ bản trong quá trình suy diễn là cơ chế hợp và suy diễn vét cạn).
- Do tri thức được biểu diễn bằng logic vị từ nên các ưu thế sử dụng cấu trúc dữ liệu không được khai thác triệt để.

b. Lôgic bất định gồm: lôgic *tình huống* (các kết xuất của nó có thể rút ra từ các tình huống không gian và thời gian xác định có liên quan với nhau), lôgic *xác suất*, lôgic *khả xuất* và lôgic *mờ*.

III.2.2. Biểu diễn tri thức nhờ luật sinh: thông qua các luật nếu ... thì:

Nếu P_1, P_2, … và P_n thì Q , *hay:*

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow Q$$

Cách biểu diễn này được áp dụng trong:

- Lôgic mệnh đề và vị từ: với $P_1, P_2, \ldots, P_n$ là các biểu thức lôgic và $\rightarrow$ là phép kéo theo
- Ngôn ngữ lập trình: luật sinh là các câu lệnh

  if (P_1 and P_2 and … and P_n) then Q ;
- Ngôn ngữ tự nhiên: luật sinh là phép dịch.
- Hệ chuyên gia: là một hệ cơ sở tri thức, gồm:

  . CSDL các sự kiện F (Facts) : $\{f_1, ..., f_n\}$

  . Tập luật R (Rules) gồm các luật có dạng:

$$f_{i_1} \wedge f_{i_2} \ldots f_{i_k} \rightarrow q$$

. Phương pháp suy diễn tiến và lùi, chẳng hạn:

$$f_{i_1}, f_{i_2}, \ldots, f_{i_k} \in F \Rightarrow q \in F$$

III.2.3. Biểu diễn tri thức nhờ mạng ngữ nghĩa (Semantic Network)

- *Ví dụ 2*: Ta có thể biểu diễn CSTT:

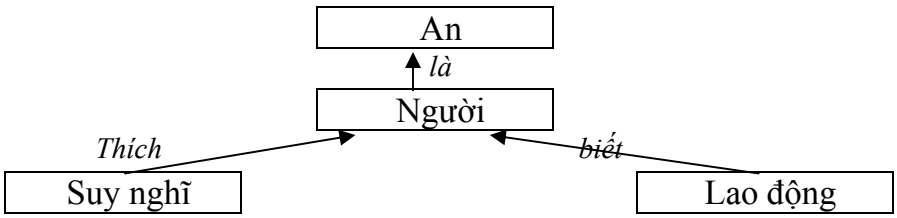
. An là người

. Người thích suy nghĩ

. Người biết lao động

thông qua sơ đồ *III.2.3.1*.

Theo cơ chế lan truyền thông tin theo các cung nối được thể hiện bằng các mũi tên, ta thu được: An thích suy nghĩ và biết lao động.



(*Hình III.2.3.1*)

- *Biểu diễn mạng ngữ nghĩa bằng đồ thị*:

Mạng ngữ nghĩa là đồ thị có hướng mở rộng: G = (V, R), trong đó: $V = \{x_1, x_2, \ldots, x_n\}$ là tập các đỉnh, $R = \{r_1, r_2, \ldots, r_m\}$ là tập các quan hệ nhiều ngôi trên V.

- *Đặc điểm của mạng ngữ nghĩa*: trực quan, có tính kế thừa và lan truyền thông tin; khó kiểm tra tính phi mâu thuẫn, …

- *Ứng dụng của mạng ngữ nghĩa*: trong xử lý ngôn ngữ tự nhiên (dễ dàng hơn trong việc hiểu ngữ nghĩa của câu, do đó có thể tiến hành việc đọc và dịch các bài text), hệ chuyên gia, giải các bài toán thông minh (chẳng hạn, giải bài toán hệ thức lượng trong tam giác), …

III.2.4. Biểu diễn tri thức bằng Frame

- Tương tự như bản ghi (*record*), nhưng *ngoài các trường dữ liệu*, *khung (frame)* còn có thể có các trường là các *thao tác,* thủ tục hay luật suy diễn nào đó, các trường này có thể *kế thừa các tính chất từ các khung khác* (tương tự như khái niệm lớp trong lập trình hướng đối tượng *OOP*). Frame có *tính cấu trúc* rất cao.

- *Ví dụ*:
Frame MáyTínhCáNhân;
{ Là một dạng riêng của MáyTínhĐiệnTử;
  KiếnTrúc {8 bits, 16 bits, 32 bits};
  SốỔĐĩa {1, .., 5};
  MànHình Thủ tục xác định tính năng của màn hình;
}

Ngoài ra, người ta còn có thể biểu diễn tri thức bằng *bộ ba liên hợp OAV* (Object - Attribute – Value), *mạng nơron (ANN* - Artificial Neural Network), …

III.3. Xử lý tri thức tất định bằng phương pháp suy diễn logic

III.3.1. Các cơ chế lập luận với tri thức tất định

- *Suy diễn* (Deduction, diễn dịch): modus ponens, modus tollens, các kiểu suy diễn: tiến và lùi.
- *Qui diễn* (Abduction, tương tự): Cho trước: $A \to B$.
  . $A \sim A' \Rightarrow A' \to B$
  . $B \sim B' \Rightarrow A \to B'$
  . $A \sim A'$ & $B \sim B' \Rightarrow A' \to B'$
- *Qui nạp* (Induction): hoàn toàn và không hoàn toàn

Sau đây ta sẽ xét các thuật toán và thuật giải nhằm chứng minh tự động các biểu thức lôgic mệnh đề.

Bài toán A (BTA):Xét bài toán lôgic mệnh đề sau có hằng đúng hay không:

$$\bigwedge_{1 \le i \le m} GT_i \Rightarrow \bigvee_{1 \le j \le m} KL_j$$

với $\{GT_i , KL_j\}$ là các biểu thức lôgic mệnh đề bất kỳ.

Tại sao ta chỉ xét bài toán dạng trên đây (*bài tập)* ?

III.3.2. Thuật toán Vương Hạo (Wong Havard) giải BTA

* *Ý tưởng*: Áp dụng chiến lược "*Chia để trị*" nhằm tách bài toán xuất phát thành các bài toán con dạng "VÀ" đơn giản hơn. Bài toán ban đầu sẽ được giải khi và chỉ khi mọi bài toán con sơ cấp giải được.

* Trước tiên, ta đưa vế *trái VT* (hay vế *phải VP*) về dạng *chuẩn hội (*hay *chuẩn tuyển* tương ứng) bằng cách:

. Thay các phép toán tương đương $\leftrightarrow$ (nếu có) bởi các phép toán kéo theo $\rightarrow$

. Thay các phép toán $\rightarrow$ bởi các phép toán $\neg, \vee$

. Dùng các luật *De Morgan* để bỏ các dấu $\neg$ của nguyên một nhóm mệnh đề

. Dùng các *luật phân phối* (nếu chưa gặp dạng chuẩn cần tìm) của *phép tuyển đối với phép hội* (hay của *phép hội đối với phép tuyển* tương ứng).

Trong thuật toán Wong, sau khi đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển, ta sẽ sử dụng các qui tắc sau (hãy chứng minh tại sao ta có quyền sử dụng chúng ? *Bài tập*).

- Thủ tục *Chuyển*(VT,VP): thay các dấu $\wedge$ các nhóm chính bên VT và các dấu $\vee$ các nhóm chính bên VP bởi dấu phẩy; chuyển vế các mệnh đề chính ở dạng phủ định $\neg$p từ vế này sang vế kia và bỏ đi dấu $\neg$ (chỉ còn p). Nói cách khác, ta có thể xem dấu phẩy (,) bên VT là phép hội và dấu phẩy (,) bên VP là phép tuyển.

. <u>*Ví dụ 3*</u>: Kiểm tra tính hằng đúng của biểu thức lôgic mệnh đề sau:

$VT \equiv (\neg a) \wedge (\neg b \vee c) \quad \Rightarrow \quad (a \wedge b) \vee (\neg b) \vee c \equiv VP$

Các *nhóm chính* trong: vế trái VT là $(\neg a)$ và $(\neg b \vee c)$, vế phải VP là $(a \wedge b)$, $(\neg b)$ và (c).

$VT \equiv (\neg a) \wedge (\neg b \vee c) \quad \Rightarrow \quad (a \wedge b) \vee (\neg b) \vee c \equiv VP$

Chuyển ↓

$\neg a, \neg b \vee c \quad \Rightarrow \quad a \wedge b, \neg b, c$

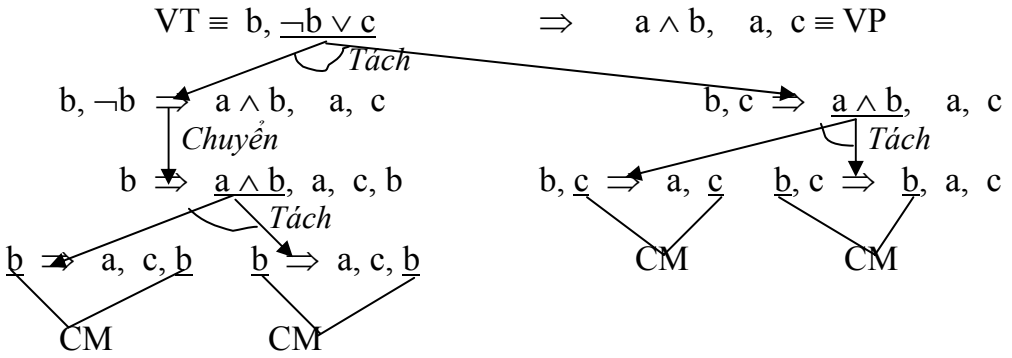
Chuyển ↓

$VT \equiv b, \neg b \vee c \quad \Rightarrow \quad a \wedge b, \quad a, \ c \equiv VP$

- *Hàm lôgic Tách*(VT,VP): tách mỗi *nhóm tuyển* $\vee$ *chính trong VT* (hay *mỗi nhóm hội* $\wedge$ *chính trong VP)* thành *nhiều bài toán con dạng VÀ*. Mỗi bài toán con gồm một mệnh đề trong nhóm chính và giữ nguyên các nhóm chính khác. Nếu tách được thì thủ tục Tách(VT,VP) nhận giá trị *true*; nếu ngược lại, nó nhận trị *false*. Với ví dụ 3 trên đây, ta có sơ đồ biến đổi như *hình III.3.2.1* sau đây.

- Với mỗi bài toán con, nếu mỗi vế của nó *có một mệnh đề đơn đứng độc lập giống nhau thì nó được chứng minh (hằng đúng)*. *Bài toán xuất phát chỉ <u>được chứng minh</u> (hằng đúng) khi <u>mọi</u> bài toán con của nó được chứng minh*. Nói cách khác, *bài toán xuất phát <u>không hằng đúng</u> nếu có <u>tồn tại</u> một bài toán con của nó*

không hằng đúng (không thể áp dụng qui tắc tách cũng như chuyển được nữa và hai vế không có chung một mệnh đề đơn đứng độc lập nào cả).



(Hình III.3.2.1)

Vậy bài toán nêu ra trong ví dụ 3 là hằng đúng.

* ***Thuật toán Vương Hạo***

```
{       VT = VP = ∅ ;
        for  i = 1 to m  do { ChuẩnHội(GTi);      VT ← VT U {GTi }; }
        for  i = 1 to n  do { ChuẩnTuyển(KLi);  VP ← VP U {KLi }; }
        P ← {(VT,VP)};
        while  (P≠∅)  do
        {   (VT,VP) ← get(P);
            if  (VT ∩ VP = ∅) then
            {   Chuyển(VT,VP);
                if  (VT ∩ VP = ∅)  then
                    if  (not(Tách(VT,VP)))
                        then  exit("Không thành công"); // BT không hằng đúng
            }
        }
        write("Thành công");                          // BT hằng đúng
}
```

* *Nhận xét*:

. Thuật toán Wong *dừng sau một số hữu hạn bước* và cho ra thông báo "Thành công" nếu và chỉ nếu từ các giả thiết GT_1, ..., GT_m có thể suy ra một trong các kết luận KL_1, ..., KL_n.

. Nếu số các phép toán liên kết $\vee$ trong GT_i và $\wedge$ trong KL_j là M thì thuật toán sẽ sinh ra từ M đến 2^M dòng (VT,VP) $\in$ P (bùng nổ tổ hợp nếu M khá lớn !).

* *Ví dụ 4 (biểu thức hằng đúng)*: Có thể chứng minh rằng:

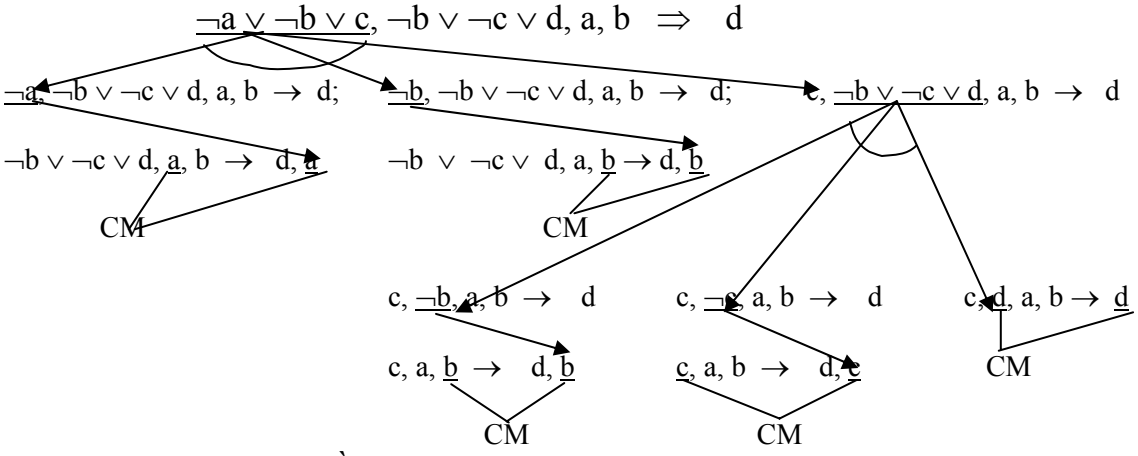
$$[(a \wedge b \rightarrow c) \wedge (b \wedge c \rightarrow d) \wedge a \wedge b] \Rightarrow d \ ?$$

Đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển:

VT ≡ {¬a ∨ ¬b ∨ c, ¬b ∨ ¬c ∨ d, a, b}: dạng chuẩn hội

VP ≡ {d}: dạng chuẩn tuyển

Ta có thể *biểu diễn quá trình giải* của thuật toán Wong thông qua *đồ thị suy diễn* hay *đồ thị lời giải* như sau*:*



(*Đồ thị lời giải trong thuật toán Wong)*

* *Ví dụ 5 (biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

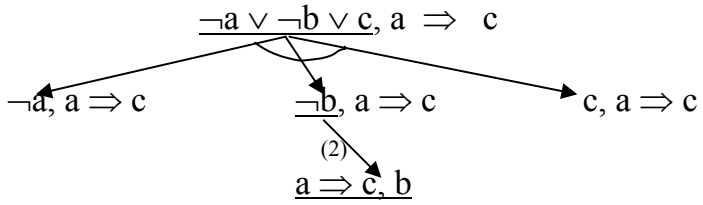
$$[(a \wedge b \rightarrow c) \wedge a] \Rightarrow c \ ? \qquad (1)$$

Đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển:

VT ≡ {¬a ∨ ¬b ∨ c, a}: dạng chuẩn hội

VP ≡ {c}: dạng chuẩn tuyển

Đồ thị lời giải:



BT con *(2)* sai vì không thể tách, chuyển và không có mệnh đề chung. Do đó bài toán *(1)* xuất phát cũng sai.

III.3.3. Thuật toán Robinson giải BTA

* *Ý tưởng*: Sử dụng:

. phương pháp *chứng minh phản chứng*:

$[a \rightarrow b$ *đúng*$] \equiv [a \wedge \neg b$ *sai* hay *mâu thuẫn*$]$

. *nguyên lý hợp giải*:

$(\neg a \vee b) \wedge (a \vee c) \Rightarrow b \vee c.$

(V*ì sao? Hãy chứng minh*).

- Ta thường gặp các trường hợp riêng của nguyên lý này:

. nguyên lý *modus ponens*:

$a \wedge (\neg a \vee b) \Rightarrow b$

. nguyên lý *modus tollens:*

$(\neg b \wedge (a \rightarrow b)) \Rightarrow \neg a$

- Để chứng minh từ các giả thiết GT_1, ..., GT_m suy ra một trong các kết luận KL_1, ..., KL_n, ta chỉ cần *lấy phủ định của KL_1, ..., KL_n đưa về cùng với các giả thiết*:

$$\mathbf{P \equiv (\bigwedge_{1 \le i \le m} GT_i) \wedge (\bigwedge_{1 \le j \le m} \neg KL_j)}$$

Nếu *suy ra được mâu thuẫn* từ tập các mệnh đề P thì quá trình chứng minh kết thúc và kết luận *BTA là hằng đúng*.

- Để suy ra được mâu thuẫn, Robinson đã đưa ra nguyên lý hợp giải trên đây để *bổ sung thêm càng ngày càng nhiều các biểu thức mệnh đề trung gian mới* cho đến khi nào trong P *có 2 mệnh đề đơn đứng độc lập là phủ định của nhau thì mâu thuẫn xảy ra*.

* ***Thuật toán Robinson*** *(giải BTA)*

```
{       P = ∅;
        for  i = 1 to m  do { GTi ← ChuẩnHội(GTi);     P  ← P U GTi; }
        for  i = 1 to n do {NotKLi ← ChuẩnHội(¬KLi); P ← P U NotKLi;}
        if  (MâuThuẫn(P)) then exit("Thành công");
        P1 = ∅;
        while  (P≠ P1  and not(MâuThuẫn(P)))  do {P1 =  P; HợpGiải(P);}
        if  (MâuThuẫn(P))
        then exit("Thành công");
        else  exit("Không thành công");
```

}

trong đó, *hàm lôgic MâuThuẫn*(P) và thủ tục *HợpGiải*(P) có nội dung như sau:

Function MâuThuẫn*(P): boolean*

{ **for each** p ∈ P **do**
  for each q ∈ P and q ≠ p **do**
    if (p = ¬ q or q = ¬ p) **then** return(*True*);
  return(*False*);
}

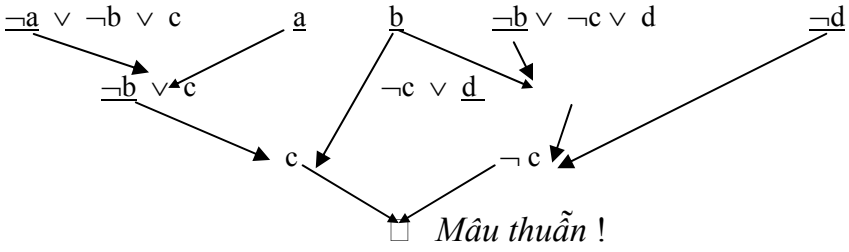
Procedure HợpGiải*(P)*

{ **for each** p ∈ P **do**
  for each q ∈ P and q ≠ p **do**
    if (p = ¬ a ∨ b and q = a ∨ c) **then** P ← P U {b ∨ c};
}

* *Ví dụ 6 (biểu thức hằng đúng)*: Xét bài toán trong *ví dụ 4*, ta có:

P ≡ {¬a ∨ ¬b ∨ c, ¬b ∨ ¬c ∨ d, a, b, ¬d}

Để tiện theo dõi, ta biểu diễn quá trình thực hiện thuật toán qua *đồ thị hợp giải* như sau:



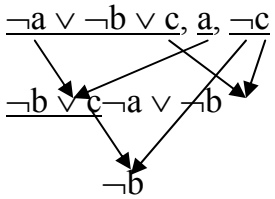
* *Ví dụ 7 (biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

[(a ∧ b → c) ∧ a] ⇒ c ?

Lấy phủ định của kết luận và đưa về dạng chuẩn hội, ta có:

P ≡ {¬a ∨ ¬b ∨ c, a, ¬c}

Đồ thị hợp giải:



Không thấy mâu thuẫn cũng không còn cách nào hợp giải mà có thể sinh ra mệnh đề mới nữa. Vậy bài toán xuất phát sai.

* <u>*Nhận xét*</u>:

- Thủ tục Robinson sẽ *dừng sau hữu hạn bước* và cho ra thông báo "Thành công" nếu và chỉ nếu $GT_1 \wedge ... \wedge GT_m \Rightarrow KL_1 \vee ... \vee KL_n$.
- Để *chứng minh BTA không hằng đúng*, ta phải *hợp giải hết tất cả các khả năng cho đến khi không thể sinh ra thêm biểu thức mệnh đề nào mới* trong P.
- Cũng như thủ tục Wong H., thủ tục Robinson có *nhược điểm* là *tùy theo thứ tự lấy các cặp mệnh đề để hợp giải* có thể *xảy ra hiện tượng tràn bộ nhớ (do bùng nổ tổ hợp)* đối với các bài toán có kích thước lớn.
- Có thể *áp dụng một số heuristic* cho thuật toán trên để thu được *"thuật giải Robinson"*, chẳng hạn: khi áp dụng nguyên lý hợp giải, sau khi thêm vào tập P mệnh đề $(b \vee c)$, có thể bỏ hai mệnh đề $(\neg a \vee b)$ và $(a \vee c)$ ra khỏi P.

* <u>*Ví dụ 8*</u> *(biểu thức hằng đúng nhưng có thể kết luận sai nếu áp dụng heuristic không đúng)*: Kiểm tra tính hằng đúng của biểu thức:

$$[a \wedge (a \rightarrow b) \wedge (a \rightarrow c)] \Rightarrow c \qquad (3)$$

Lấy phủ định của kết luận và đưa về dạng chuẩn hội, ta có:

$$P \equiv \{\neg a \vee b, \neg a \vee c, a, \neg c\}$$

. Nếu hợp giải trước hai mệnh đề $\neg a \vee b$ và a để được b rồi loại chúng thì trong P còn lại:

$$P \equiv \{b, \neg a \vee c, \neg c\}$$

Hợp giải tiếp tục thì: $P \equiv \{b, \neg a\}$: không có mâu thuẫn cũng không thể hợp giải được nữa: bài toán sai chăng ? Thật ra, bài toán *(3)* vẫn đúng !

. Nếu hợp giải trước hai mệnh đề $\neg a \vee c$ và a để được c rồi loại chúng thì trong P còn lại:

$$P \equiv \{c, \neg a \vee b, \neg c\}$$

Khi đó mâu thuẫn xảy ra giữa c và $\neg c$. Vậy bài toán xuất phát đúng !

* <u>*Chú ý*</u>: Để tránh tình trạng kết luận sai như trên, khi *thay thuật toán Robinson thành "thuật giải Robinson*", ta nên chú ý rằng ngoài việc *thêm heuristic trên vào thủ tục hợp giải, ta phải bỏ đi thông báo exit("Không thành công")*. Nếu xuất hiện mâu thuẫn thì kết luận "Thành công" và dừng. Nếu *không xuất hiện mâu thuẫn* thì không được thông báo exit("Không thành công"), nghĩa là *không được quyền kết luận gì* khi tình trạng này xảy ra !

* ***<u>Bài toán B (BTB)</u>***

Input: - Tập các mệnh đề giả thiết: $GT = \{g_1, ... , g_n\} = \wedge_{1 \le i \le n} g_i$

- Tập RULE gồm *m* luật có dạng chuẩn r : $p_1 \wedge ... \wedge p_n \rightarrow q$
- Tập các mệnh đề kết luận: KL = $\{q_1, ..., q_k\} = \wedge_{1 \le i \le k} q_i$

Output: Thông báo "Thành công" nếu *mọi* $q_i \in KL$ có thể suy diễn từ GT nhờ sử dụng tập luật RULE.

- Để chứng minh các bài toán lôgic mệnh đề, người ta còn dùng phương pháp *suy diễn tiến* (hay *lùi*), bằng cách *xuất phát từ giả thiết* (hay *kết luận* tương ứng), dựa trên các nguyên lý suy diễn:

. *Modus ponens:* $\dfrac{A,\ A \rightarrow B}{B}$

. *Modus tollens:* $\dfrac{\neg B,\ A \rightarrow B}{\neg A}$

ta *thêm vào các mệnh đề mới được chứng minh đúng* (hay các mệnh đề *cần phải chứng minh thêm* tương ứng) cho đến khi thu được *kết luận* (hay *giả thiết* tương ứng) thì dừng.

III.3.4. Thuật toán suy diễn tiến giải BTB

* *Ý tưởng*: Quá trình *suy diễn tiến bắt đầu từ tập TrungGian các mệnh đề đúng xuất phát* (tập giả thiết*)* và nó *sẽ được "làm nở" dần bằng cách thêm vào các sự kiện mới đúng* nhờ các luật suy diễn trong RULE*, cho đến khi các kết luận cần chứng minh được phát hiện.*

*** *Thuật toán suy diễn tiến SDT***

```
{       TrungGian = GT;
        THỎA   = Lọc(RULE, TrungGian);
      // THỎA là tập các luật r có dạng p1∧ ... ∧pn → q mà pi∈TrungGian, ∀i=1..n
        while (KL ⊄ TrungGian and THỎA ≠ ∅) do
          {     r ← get(THỎA);      // r ∈ THỎA có dạng r: p1∧ ... ∧pn → q
                TrungGian ← TrungGian U {q};
                RULE ← RULE \ {r};
                THỎA = Lọc(RULE, TrungGian)
          }
        if (KL ⊂ TrungGian)
        then write("Thành công")
        else write("Không thành công");
}
```

* *Ví dụ 9*: Cho trước tập các sự kiện giả thiết: GT = {a, b}. Sử dụng tập RULE các luật:

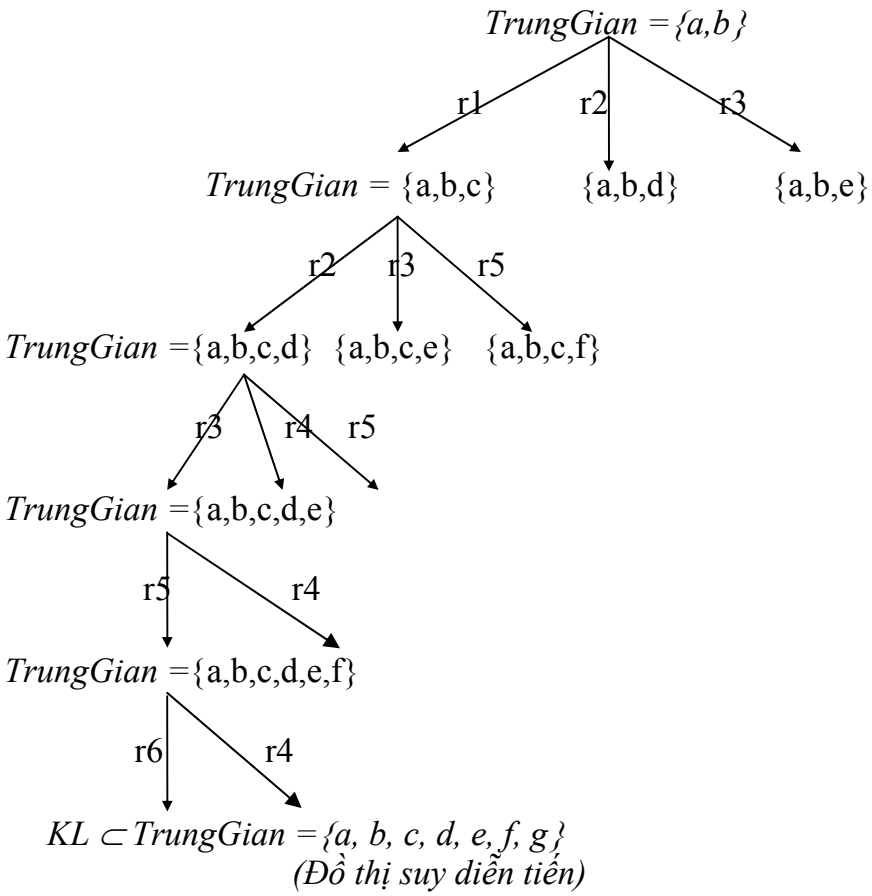
r1: $a \rightarrow c$

r2: $b \rightarrow d$

r3: $a \rightarrow e$
r4: $a \wedge d \rightarrow e$
r5: $b \wedge c \rightarrow f$
r6: $e \wedge f \rightarrow g$
Cần suy ra kết luận: KL = {g}.

. Ta có: TrungGian = {a, b}; RULE = {r1-r6}. Do đó THỎA = {r1,r2,r3}.
. Áp dụng luật r1: $a \rightarrow c$, ta được TrungGian = {a,b,c} và RULE={r2-r6}.
Do đó: THỎA = {r2, r3, r5}.
. Lúc này, KL = {g} $\not\subset$ {a, b, c} = TrungGian và THỎA $\neq \varnothing$.
Ta lại tiếp tục quá trình, bằng cách chọn r2 để thực hiện, ...

Ta minh họa quá trình trên bằng *đồ thị suy diễn tiến*, trong đó mỗi đỉnh ứng với tập TrungGian tại thời điểm đang xét và sẽ có một cung đi từ đỉnh *TrungGian* đến đỉnh *TrungGian* $\boldsymbol{U}$ *{q}* tương ứng với luật r đã được lọc và có dạng r : $p_1 \wedge ... \wedge p_n \rightarrow q$ mà $p_i \in$ TrungGian, $\forall i=1,.., n$.



(Đồ thị suy diễn tiến)

* <u>*Ví dụ 10*</u> *(biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

$[(a \wedge b \rightarrow c) \wedge a] \Rightarrow c$?

TrungGian = GT = {a}, KL = {c}, RULE = {r1: $a \wedge b \rightarrow c$ }.

Khi đó: THỎA = $\varnothing$ và KL $\not\subset$ TrungGian. Vậy bài toán trên không hằng đúng.

* <u>*Nhận xét*</u>:

.Với các thứ tự khác nhau để chọn r từ tập THỎA, ta sẽ có các cơ chế duyệt theo chiều sâu, theo chiều rộng, ... trên đồ thị.

. Nếu trong *BTB* thay tập kết luận KL = $\wedge_{1\le i\le k}\, q_i$ bởi: KL = $V_{1\le i\le k}\, q_i$ thì cần hiệu chỉnh thuật toán SDT ra sao ? (*Bài tập*)

. Để tránh các kết luận sai, với *mỗi biến mệnh đề p* xuất hiện trong biểu thức cần chứng minh, bằng các qui tắc biến đổi tương đương lôgic, chuyển về *cùng dạng p* hoặc *cùng dạng* $\neg p$.

III.3.5. Thuật toán suy diễn lùi giải BTB

* Với suy diễn lùi, để đưa ra kết luận B ta thử tìm tất cả các luật có dạng:

$A_1 \wedge ... \wedge A_n \rightarrow B$

Để có B, ta cần chứng minh A_1 ,... , A_n (các kết luận mới được thêm vào tập kết luận). Quá trình xác định A_i cũng diễn ra tương tự như đối với B. Nếu đến một lúc nào đó tìm thấy một A_{io} nào đó không thể dẫn xuất được từ các giả thiết thì ta quay lui sang luật khác sinh ra B và lại tiếp tục quá trình trên. Nếu không tìm được A_{io} như vậy (nghĩa là mọi A_i đều được dẫn xuất từ giả thiết) thì quá trình dẫn xuất ra B thành công.

Để thực hiện quá trình quay lui, ta sử dụng hai tập có cấu trúc ngăn xếp GOAL và VET: GOAL là tập lưu các mệnh đề cần phải chứng minh đến thời điểm đang xét và VET là tập lưu các luật đã sử dụng để chứng minh các đích (kể cả đích trung gian).

* ***<u>Thuật toán suy diễn lùi SDL</u>***

```
{1 if (KL ⊂ GT)
  then exit("Thành công");
  else {2 GOAL = ∅;
        VET = ∅;
        CMĐược = True;
        for each q ∈ KL do
                    GOAL = GOAL U {(q,0)};
        repeat
```

```
{3  (f,i) ← get(GOAL);
    if (f ∉ GT) then
    {4 Tìm_Luật(f, i, RULE, j);          // Tìm luật r_j : left_j →f
      if (j ≤ m)
      then { VET = VET U {(f,j)};
             for each t ∈left_j\GT do GOAL=GOAL U {(t,0)};
           }
      else {5 back = true;               // dùng biến này để quay lui
             while (f ∉ KL and back) do
             {6 repeat {(g,k)←get(VET);
                  // Lấy luật r_k:left_k→g ra khỏi VET
                  // để quay lui đến luật khác mà cũng →g
                        GOAL = GOAL \ left_k;
                        }
               until (f ∈ left_k);
                  // Bỏ hậu quả của việc chọn sai luật r và dẫn đến việc
                  // không CM được mệnh đề f được giả định cần CM
               Tìm_Luật(g, k, RULE, s); // Tìm luật r_s:left_s →g
               if (s ≤ m)
               then { for each t ∈ left_s \ GT do
                                GOAL = GOAL U {(t,0)};
                      VET = VET U {(g,s)}; back = false;
                    }
               else f = g;
             }6
             if (f ∈ KL and back) then CMĐược = False;
           }5
    }4
 }3
until (GOAL = ∅ or not(CMĐược));
if (not(CMĐược)) then exit("Không Thành công")
else exit("Thành công");
}2
}1
```

Trong thủ tục trên ta sử dụng thủ tục: *Tìm_Luật*(f, i, RULE, k) để tìm xem có luật r_k nào kể từ luật thứ i+1 trở đi mà cũng suy ra được f (r_k:$left_k$ →f). Nếu không có luật nào như thế thì qui ước lấy k = m+1.

* <u>*Ví dụ 11*</u>: Cho trước tập các sự kiện giả thiết GT = {a, b}. Sử dụng tập RULE các luật:

r1. $a \wedge b \rightarrow c$
r2. $a \wedge h \rightarrow d$
r3. $b \wedge c \rightarrow e$
r4. $a \wedge d \rightarrow m$
r5. $a \wedge b \rightarrow o$
r6. $o \wedge e \rightarrow m$

Cần suy ra KL = {m}.

Ban đầu GOAL = VET = ∅;

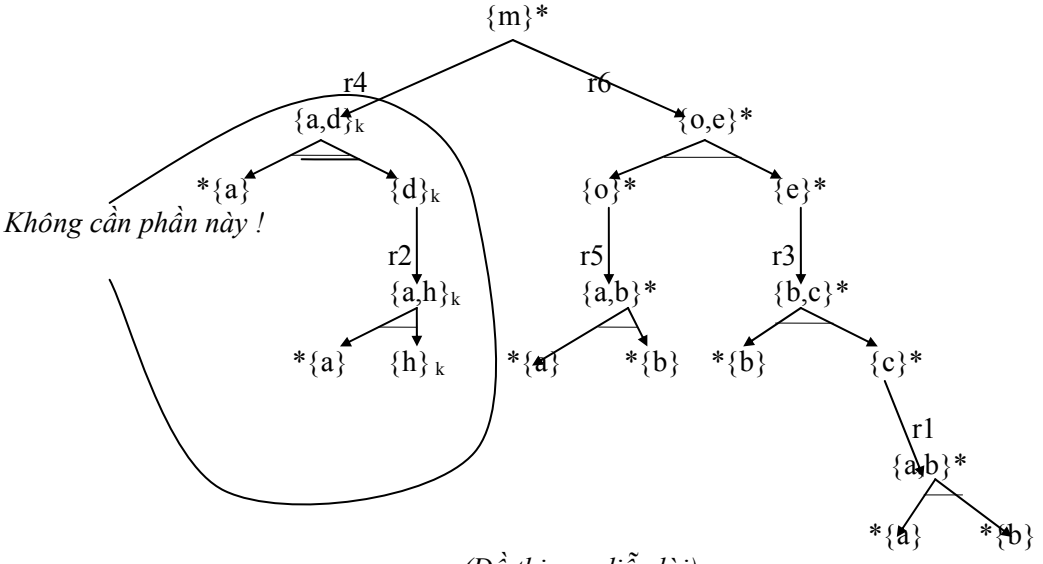
Áp dụng thủ tục *Tìm_Luật*(m, 0, RULE, j), ta được j = 4 (r4 là luật đầu tiên sinh ra m). Khi đó VET = {(m,4)}; GOAL = {(d,0)} (vì a ∈ GT nên chỉ cần xét (d,0)).

Ta tiếp tục quá trình và có bảng theo dõi sau:

GOAL (back)	(f,i)	CMĐ	j	$left_j$\GT	VET	(g,k)	l	$left_l$\GT	Quaylui
{(m,0)}	(m,0)	1	true	d	{(m,4)}				
{(d,0)}	(d,0)		2	h	{(d,2);(m,4)}				
{(h,0)}	(h,0)		7		{(m,4)}	(d,2)	7		true
∅	d				∅	(m,4)	6	o,e	
{(o,0),(e,0)}					{(m,6)}				false(thoát while 6)
{(e,0)}	o		5	∅	{(o,5),(m,6)}				
∅	e		3	c	{(e,3),(o,5),(m,6)}				
{(c,0)}	c		1	∅	{(c,1),(e,3),(o,5),(m,6)}				
∅									

(GOAL = ∅ : thoát vòng while$\{_2\}_2$; *CMĐược = True: Thành công !)*

Ta có thể biểu diễn quá trình suy diễn lùi trên đây thông qua *đồ thị* (VÀ/HOẶC) *suy diễn lùi* như sau:



(Đồ thị suy diễn lùi)

- *Ví dụ 12*: Cho trước tập các sự kiện giả thiết GT = {a}. Sử dụng tập RULE các luật:

r1. $a \wedge b \rightarrow c$
r2. $a \wedge h \rightarrow d$
r3. $b \wedge c \rightarrow e$
r4. $a \wedge d \rightarrow m$
r5. $a \wedge b \rightarrow o$
r6. $o \wedge e \rightarrow m$

Cần suy ra KL = {m}.

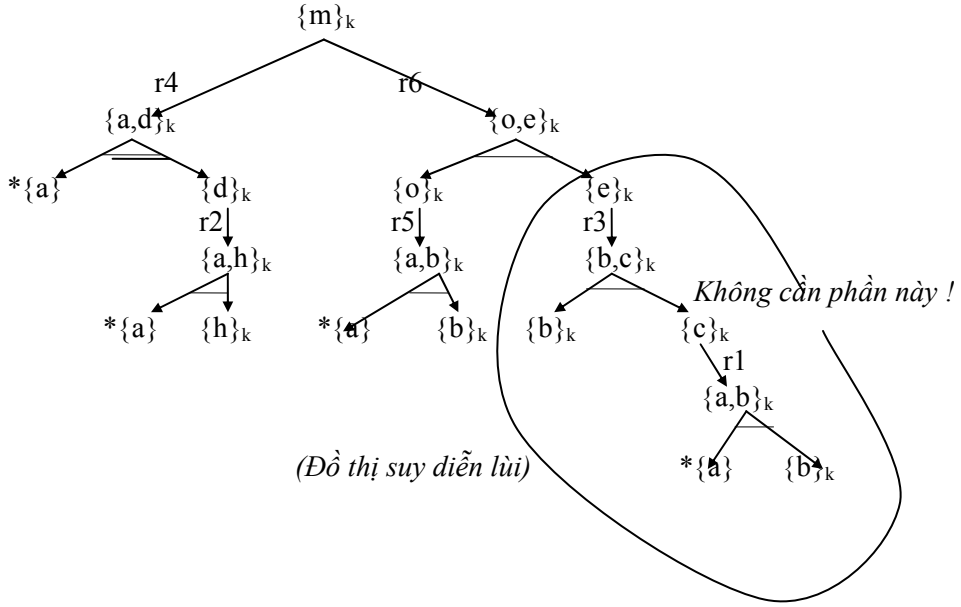
Ban đầu GOAL = VET = ∅. Áp dụng thủ tục *Tìm_Luật*(m, 0, RULE, j), ta được j = 4 (r4 là luật đầu tiên sinh ra m). Khi đó VET = {(m,4)}; GOAL = {(d,0)} (vì a ∈ GT nên chỉ cần xét (d,0)).

Ta tiếp tục quá trình, sẽ có bảng theo dõi sau:

GOAL (back)	(f,i)	CMĐ	j	$left_j$\GT	VET	(g,k)	l	$left_l$\GT	Quaylui
{(m,0)} (m,0)	1	true	d	{(m,4)}					
{(d,0)}	(d,0)		2	h	{(d,2);(m,4)}				
{(h,0)}	(h,0)		7		{(m,4)}	(d,2)	7		true
∅	d				∅	(m,4)	6	{o,e}	
{(o,0),(e,0)}					{(m,6)}			(thoát while 6)	false
{(e,0)}	o		5	b	{(o,5),(m,6)}				
{(b,0),(e,0)}	b		7	∅	{(m,6)}	(o,5)	7		true
∅	o				∅	(m,6)	7		
∅	m	false							

(f = m ∈ KL : thoát while $\{_6\}_6$; do đó: *CMĐược= false (*và GOAL = ∅): thoát vòng while$\{_2\}_2$; *CMĐược= false: Không thành công !)*

Đồ thị suy diễn lùi:



(Đồ thị suy diễn lùi)

* <u>*Nhận xét*</u>:

- Quá trình suy diễn lùi *tương tự như quá trình tìm đồ thị con lời giải trong đồ thị VÀ/HOẶC* biểu diễn tập luật.
- Để *tăng hiệu quả* của thủ tục suy diễn lùi có thể đưa vào 2 tập: tập ĐÚNG chứa các sự kiện đã được khẳng định là đúng và tập SAI chứa các sự kiện đã được khẳng định là sai. Mỗi khi lấy một sự kiện (f, i) nào đó ta cần kiểm tra trước $f \in$ ĐÚNG hay $f \in$ SAI ?

III.4. Xử lý tri thức bất định bằng phương pháp suy diễn logic

III.4.1. Các cơ chế lập luận với tri thức bất định và không chính xác

- *Khái niệm*: Trên thực tế, có nhiều mệnh đề được phát biểu không chính xác, mang tính bất định, *chúng không hẳn hoàn toàn đúng cũng không hoàn toàn sai*. Tính đúng - sai của chúng không được xác định và thể hiện rõ ràng như trong lôgic mệnh đề cổ điển. Ta có thể mở rộng tính đúng sai của mệnh đề *p* thông qua một *hàm μ đo độ đúng* của nó: *μ (p) $\in$ [0; 1]*.

- *Mô hình xác suất*: Xét P là tập các mệnh đề đóng kín đối với các phép toán $\neg$, $\wedge$ (do đó cả $\vee$) và ánh xạ (độ đo xác suất) Pr: $P \to [0, 1]$ thỏa các tính chất:
  - $\Pr(p) + \Pr(\neg p) = 1, \forall p \in P$
  - $\Pr(p \wedge q) = \Pr(p).\Pr(q), \forall p, q \in P$
  - Từ đó: $\Pr(p \vee q) = \Pr(p) + \Pr(q) - \Pr(p \wedge q)$ *(Chứng minh ? Bài tập)*

- <u>*Mô hình khái luật*</u>:

Biểu diễn mệnh đề:

- Nếu p thì q bởi ***luật*** bình thường: $p \to q$
- Nếu p thì *thông thường (nói chung, hầu như, …)* q bởi ***khái luật***: $p \to q$ **!**

III.4.2. Phân bố khả xuất của khái luật và các phép toán nối kết

* <u>*Phân bố khả xuất (PBKX) của khái luật*</u>:

- PBKX của một cặp sự kiện đối ngẫu nhau p và $\overline{p}$ (hay $\neg$p): $\pi(p)$, $\pi(\overline{p})$ thỏa: *max $(\pi(p), \pi(\overline{p})) = 1$*

- *PBKX của một sự kiện p* trong khái luật là bộ: $\begin{pmatrix} \pi(p) \\ \pi(\neg p) \end{pmatrix}$

  Nếu ta không thể khẳng định được p là tuyệt đối đúng thì:

  $\pi(p) = 1$, nhưng $\pi(\overline{p}) = \lambda$ $(\lambda \in [0, 1))$

- *PBKX của một khái luật* $p \to q$! được biểu diễn bởi ma trận:

$$\begin{pmatrix} \pi(p \to q) & \pi(\neg p \to q) \\ \pi(p \to \neg q) & \pi(\neg p \to \neg q) \end{pmatrix}$$

Bài tập

1. +Bằng các phép biến đổi tương đương trong lôgic mệnh đề, hãy chứng minh:
   a. Ta luôn có thể đưa một luật suy diễn ở dạng *chuẩn Horn*:
      $p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q_1 \vee .... \vee q_m$
   về dạng *chuẩn sơ cấp*:
      $p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q$
   b. Nguyên lý *chứng minh phản chứng*:
      (a $\rightarrow$ b *đúng*) $\equiv$ (a $\wedge \neg$ b *sai* hay *mâu thuẫn*)
   c. Nguyên lý *hợp giải.*
   d. Nguyên lý *modus ponens và tollens.*
   e. Trong việc chứng minh tự động tính hằng đúng của các biểu thức lôgic mệnh đề, tại sao người ta thường xét bài toán dạng BTA ?
   f. Quy tắc *Chuyển* và *Tách* trong thủ tục Vương Hạo.
   g. Phương pháp *Qui diễn (Abduction*, tương tự): Cho trước: A $\rightarrow$ B.
      . A ~ A' $\Rightarrow$ A' $\rightarrow$ B
      . B ~ B' $\Rightarrow$ A $\rightarrow$ B'
      . A ~ A' & B ~ B' $\Rightarrow$ A' $\rightarrow$ B'

2. +Hãy đưa về dạng *chuẩn hội* và *chuẩn tuyển* lần lượt cho vế trái (VT) và vế phải (VP) của các biểu thức lôgic mệnh đề sau:
   a. $[(a \wedge b \rightarrow c \vee d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [b \rightarrow e]$
   b. $[(a \wedge b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [b \wedge c \rightarrow e]$
   c. $[(a \vee b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [d \wedge c \vee b]$
   d. $[(a \vee b \rightarrow c \vee d) \wedge (c \wedge b \rightarrow e) \wedge a] \Rightarrow [b \wedge c \rightarrow e]$
   e. $[b \wedge c \wedge \neg e] \Rightarrow [(a \wedge b \wedge (\neg c \vee \neg d)) \vee (c \wedge \neg e) \vee \neg a]$
   f. $[a \rightarrow (b \rightarrow c)] \Leftrightarrow [(a \rightarrow b) \rightarrow c]$
   g. $\{[(a \wedge b \rightarrow c \vee d) \wedge (c \rightarrow e) \wedge a] \vee [(a \wedge b \rightarrow e) \wedge a]\} \Rightarrow [b \rightarrow e]$
   h. $\{[(a \wedge b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \vee [(a \wedge b \rightarrow e \vee d) \wedge a]\} \Rightarrow [b \wedge c \rightarrow e]$

3. *Nếu trong bài toán *BTB* thay tập kết luận KL = $\wedge_{1 \le i \le k} q_i$ bởi KL = $V_{1 \le i \le k} q_i$ thì cần hiệu chỉnh thuật toán *suy diễn tiến* SDT ra sao ?

4. +*Kiểm tra tính hằng đúng* của các biểu thức lôgic trong bài tập 2 chương III (có thể *áp dụng vài heuristics để rút gọn hay chuyển đổi tương đương* cho *việc giải trở nên đơn giản hơn*) bằng các phương pháp:
   i) biến đổi *lôgic*
   ii) phương pháp *suy diễn tiến*
   iii) phương pháp *suy diễn lùi*
   iv) thuật toán *Vương Hạo (Wong Havard)*
   v) thuật toán *Robinson.* Nếu *thay thuật toán Robinson bằng "thuật giải Robinson"* thì các kết quả có luôn trùng nhau không ? Tại sao?

Chương IV

LẬP TRÌNH LÔGIC

IV.1. GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH LÔGIC PROLOG

IV.1.1. Mở đầu

Một trong những ***mục tiêu*** lớn và quan trọng của trí tuệ nhân tạo (TTNT) là *mô phỏng trên máy tính những tư duy và hành vi* của con người. Trước những năm 1960, khi chưa có nhóm các ngôn ngữ lập trình (NNLT) chuyên phục vụ cho lĩnh vực TTNT như các NNLT xử lý ký hiệu (LISP- LISt Processing) và lập trình lôgic (PROLOG – PROgramming in LOGic), những thành tựu lý thuyết trong TTNT chưa được ứng dụng rộng rãi trong thực tế. Chương này nhằm giới thiệu *Prolog*, một trong những NNLT thuộc họ này.

NNLT Prolog do Alain Colmerauer và nhóm cộng sự thiết kế ra vào năm 1972 tại đại học Marseille ở Pháp. ***Prolog*** được xây dựng trên ***cở sở*** lý thuyết của *lôgic vị từ*, nhằm *biểu diễn, mô tả* các tri thức cùng các mối quan hệ cơ bản giữa chúng. Từ đó, khi đã *được trang bị sẵn* một cơ chế suy luận thông qua một ***môtơ suy diễn***, Prolog sẽ *sản sinh ra và kiểm tra nhiều mối quan hệ hệ quả khác*.

Khác với nhiều NNLT thủ tục cổ điển truyền thống, ***Prolog*** thuộc nhóm ***NNLT mô tả***. Để giải quyết một bài toán, đặc biệt là các bài toán xử lý ký hiệu trừu tượng và suy luận lôgic, ta chỉ cần *chọn cách biểu diễn phù hợp và mô tả các đối tượng cùng các tính chất và mối quan hệ cơ bản* đã biết giữa chúng. Thông qua môtơ suy diễn của Prolog, ta sẽ *nhận được các câu trả lời* liên quan đến các mối *quan hệ nhân quả và các kết luận liên quan* đến bài toán cần giải quyết. Khi tri thức đã biết về bài toán *không tăng thêm*, việc *trả lời các kết luận khác liên quan đến bài toán cần giải là hiển nhiên thông qua các câu hỏi thêm mà không cần phải lập trình lại* như các NNLT kiểu thủ tục. Chính vì vậy, NNLT Prolog thuộc kiểu ***hội thoại*** và một *chương trình nguồn* trong Prolog ***thường gọn hơn rất nhiều*** so với một chương trình nguồn của các NNLT thủ tục truyền thống khác như Pascal, C, … nhằm cùng giải quyết một bài toán chuyên biệt trong lĩnh vực TTNT.

* ***Ví dụ 1***: Để mô tả các *sự kiện (fact)*: "Socrate và Tèo là người", *qui tắc lôgic (hoặc luật, rule)*: "hễ bất cứ ai là người thì người đó sẽ phải chết", ta dùng hai vị từ chính: LàNgười(Ai), Chết(Ai) như sau:

LàNgười("Socrate").

LàNgười("Tèo").

Chết(Ai) **:-** LàNgười(Ai). //hoặc: Chết(Ai) **if** LàNgười(Ai).

Dựa trên các tri thức vừa mô tả, ta có thể yêu cầu Prolog *trả lời các câu hỏi* liên quan sau:

- *Goal:* LàNgười("Socrate") → yes
- *Goal:* LàNgười("mèo") → no

- *Goal:* Chết("Socrate") → yes
- *Goal:* Chết("mèo") → no

hoặc:

- *Goal:* LàNgười(Ai) → Ai = "Socrate", Ai = "Tèo" (2 solutions)
- *Goal:* Chết(Ai) → Ai = "Socrate", Ai = "Tèo" (2 solutions)

Các ***khái niệm chính*** trong Prolog là: vị từ (predicate), đối tượng, biến, sự kiện (fact), qui tắc (rule), câu hỏi (đích, mục tiêu: Goal), danh sách. Vài ***kỹ thuật*** thường được dùng trong Prolog là: quay lui (mặc định là tìm kiếm theo chiều sâu trên cây suy diễn), đệ qui, lát cắt.

IV.1.2. Vị từ, sự kiện, qui tắc, mục tiêu trong Prolog

a. Vị từ, đối tượng, hằng, biến

* Quan hệ giữa các đối tượng trong một bài toán, một mệnh đề được biểu diễn bằng các ***vị từ*** và các ***đối*** của vị từ biểu diễn các ***đối tượng*** thuộc *quan hệ* hay có *tính chất nào* đó.

Để ***chuyển một mệnh đề thông thường sang dạng vị từ*** (*predicate*) trong Prolog, người ta thường *chọn động từ chính trong phần vị ngữ làm tên vị từ* và các *thành phần khác làm các đối* tương ứng của vị từ.

- ***Phân loại*** các đối tượng trong Prolog: *hằng, biến* hay *đối tượng phức hợp (*là một loại *đối tượng của một vị từ mà bản thân nó lại là một vị từ khác)*.

- Các ***kiểu dữ liệu chuẩn*** (cơ sở) có sẵn trong Prolog: *int, real, char, string, symbol*. Ngoài ra, Prolog còn cho phép các *kiểu dữ liệu riêng do người dùng định nghĩa*.

- ***Kiểu (dữ liệu) của đối tượng*** có thể thuộc kiểu chuẩn hoặc do người dùng định nghĩa.

- Ta qui ước ***Name*** là một *dãy ký tự* (gồm: *chữ cái* la tinh, chữ *số* hoặc *dấu gạch dưới* "_") mà *ký tự đầu tiên phải là chữ cái*. (Do đó, trong *Name* không được có khoảng trắng " " và không được bắt đầu là chữ số !).

Khi dùng *Name* để biểu diễn tên của các đối tượng hay vị từ, người ta thường dùng một trong hai cách viết sau để *dễ đọc và tránh hiểu nhầm nghĩa*: ghế_dài, ghế_đẩu hay ghếDài, ghếĐẩu; suy_luận hay SuyLuận, so_1_Han_Thuyen hay so_1_HanThuyen, … (cho ví dụ vài dãy ký tự không phải là một *Name* ?)

- *Tên của đối tượng* là một *Name*.

- **Hằng** là đối tượng mà giá trị của chúng được gán ngay khi khai báo trong phần **constants** và *không đổi* trong suốt chương trình. ***Tên của hằng*** là một *Name* mà *ký tự đầu tiên phải là* ***chữ cái thường*** (đặc biệt là khi sử dụng giá trị của chúng).

- **Biến** là đối tượng mà giá trị của chúng có thể *thay đổi* trong chương trình. ***Tên của biến*** (*trừ biến vô danh*, được ký hiệu là *dấu gạch dưới* "_") là một *Name* mà ký tự đầu tiên là ***chữ cái hoa***. Biến được phân thành ***3 loại***:

. ***biến ràng buộc*** là biến mà tại thời điểm đang xét nó *được gán với một giá trị xác định*;

. ***biến tự do*** là biến mà tại thời điểm đang xét nó *chưa hoặc không được gán với một giá trị xác định nào cả*; do đó, tại những thời điểm khác nhau một biến có thể *chuyển từ tự do sang ràng buộc và ngược lại* trong quá trình quay lui khi tìm kiếm lời giải;

. ***biến vô danh*** (được ký hiệu là dấu gạch dưới "_") là biến mà ta *không quan tâm* (do đó, không cần lưu lại hay ghi nhớ) *đến giá trị* của nó.

Chú ý: Các *biến* trong Prolog khi được sử dụng *không cần phải khai báo trước* (vì thông thường, chúng đã được khai báo ngầm trong phần khai báo kiểu cho các vị từ)

b. Sự kiện, qui tắc:

Các vị từ trong Prolog thường được thể hiện dưới hai dạng: sự kiện hay qui tắc.

- ***Sự kiện*** (***fact***) là các *mệnh đề hằng đúng* và nó thường được thể hiện bằng *các vị từ mà các đối của chúng đều là đối tượng hằng*. Chẳng hạn, trong ví dụ 1, *LàNgười("Socrate")* là một sự kiện.

- ***Qui tắc*** (***rule***) là một *mệnh đề kéo theo đúng* và chúng thường được thể hiện bởi *các vị từ mà các đối của chúng chứa các biến*. Qui tắc thường ***gồm hai phần***: *PhầnĐầu* và *PhầnThânQuiTắc*, chúng được nối với nhau bởi từ khoá "***if***" hoặc "**:-**", theo ***cú pháp sau***:

PhầnĐầu **if** *PhầnThânQuiTắc***.**

hoặc:

PhầnĐầu **:-** *PhầnThânQuiTắc***.**

(*PhầnThânQuiTắc luôn được kết thúc* bởi một dấu chấm "**.**") Chẳng hạn, trong ví dụ 1, *Chết(Ai)* là một qui tắc.

. Để tạo nên các qui tắc phong phú, ta có thể dùng thêm trong *PhầnThânQuiTắc* các *phép toán lôgic* để nối kết các mệnh đề: **and** (hay dấu phẩy "**,**"), **or** (hay dấu chấm phẩy "**;**"), **not(vị từ)**.

. Các vị từ (sự kiện, qui tắc) phải được *khai báo kiểu* trong phần *Predicates* trước khi được *liệt kê, định nghĩa và được sử dụng* trong phần *Clauses* và *Goal*.

c. Mục tiêu, câu hỏi:

Sau khi lập trình, các yêu cầu của người dùng có thể được biết thông qua các *câu hỏi*, vị từ đưa vào trong phần mục tiêu *Goal*. Có hai loại *Goal*:

- ***Goal trong*** (đưa nội dung cần hỏi vào phần Goal trong chương trình nguồn): *chỉ tìm* ra *lời giải đầu tiên* và *không tự động xuất* chúng ra màn hình. Nếu muốn *xuất chúng*, có thể sử dụng vị từ chuẩn xuất dữ liệu: *write*; nếu muốn *tìm các lời giải kế tiếp*, có thể dùng thêm vị từ chuẩn luôn sai: *fail*.

- ***Goal ngoài***: *tìm* và *tự động xuất mọi lời giải* (ra cửa sổ hội thoại *Dialog*, với từ khóa Goal: ở đầu mỗi câu hỏi). Nếu chỉ muốn *xuất ra lời giải đầu tiên*, có thể sử dụng vị từ chuẩn *lát cắt* "**!**".

IV.1.3. Cấu trúc chính của một chương trình trong Prolog

Một chương trình của Prolog thường bao gồm *một số các phần chính* và *theo thứ tự* sau:

Constants % Đoạn khai báo các đối tượng hằng
Domains % Đoạn khai báo kiểu các đối tượng riêng của người dùng
Database % Đoạn khai báo kiểu cho các vị từ của cơ sở dữ liệu
Predicates % Đoạn khai báo kiểu cho vị từ sẽ dùng trong các đoạn sau
Clauses % Đoạn liệt kê sự kiện và định nghĩa qui tắc đã khai báo trong phần *Predicates*
Goal % Đoạn đưa vào các vị từ câu hỏi, có thể đặt trước phần *Clauses*

Chú ý: Một chương trình có thể thiếu một số (hoặc tất cả !) phần trên.

a. Đoạn khai báo các đối tượng hằng: được bắt đầu bởi từ khoá *Constants* và theo cú pháp sau:

TênĐốiTượngHằng = TrịĐốiTượngHằng
[*Name*]

TrịĐốiTượngHằng có kiểu dữ liệu chuẩn (*int, real, char, string, symbol*).

b. Đoạn khai báo kiểu dữ liệu riêng của người dùng: được bắt đầu bởi từ khoá *Domains* và theo một trong các cú pháp sau:

. Dãy TênKiểuDữLiệu = *KiểuDữLiệu*
[*Name, ...*]

trong đó: *KiểuDữLiệu* là kiểu dữ liệu chuẩn (*int, real, char, string, symbol*) hoặc kiểu dữ liệu riêng của người dùng;

. Dãy TênKiểuDanhSách = KiểuPhầnTửChung*****
[*Name* **,** ...] [*KiểuDữLiệu*]
. TênKiểuĐốiTượngPhứcHợp = Dãy TênVịTừ(KiểuĐối, …)
[*Name*] [*Name(KiểuDữLiệu, ...)***;** …]
. **file** = Dãy TênFileHìnhThức
[*Name***;** ...]

Các tên file của các thiết bị chuẩn sau không cần phải khai báo: *screen, keyboard, printer*.

c. Đoạn khai báo kiểu cho vị từ sẽ dùng trong các đoạn *Goal* và *Clauses* ở phần tiếp theo được bắt đầu bởi từ khoá *Predicates* và theo cú pháp sau:

. TênVịTừ(DãyKiểuĐối)
[*Name*]([*KiểuDữLiệu, ...*])

Đoạn khai báo kiểu cho các vị từ của cơ sở dữ liệu (CSDL) được bắt đầu bởi từ khoá *Database* và cũng theo cú pháp như phần *Predicates*. Điểm khác biệt của các vị từ khai báo trong đoạn *Database* là: các vị từ tương ứng với chúng

trong phần Clauses có các *đối là các đối tượng hằng hoặc các biến ràng buộc* để có thể *đưa vào CSDL ở bộ nhớ trong* theo các vị từ chuẩn riêng biệt.

Chú ý: Có thể khai báo *nhiều vị từ cùng tên* nhưng: *cùng số đối với kiểu khác nhau* hoặc có *số đối khác nhau* hoặc thậm chí *không có đối nào* ! Khi đó, chúng phải được *khai báo liên tiếp nhau*.

d. <u>Đoạn liệt kê các sự kiện và định nghĩa các qui tắc</u> đã khai báo trong phần *Predicates* hoặc *Database*, được bắt đầu bởi từ khoá Clauses theo cú pháp sau:

. Liệt kê các sự kiện:

TênVịTừ(Dãy TrịHằng)**.**

. Định nghĩa các qui tắc:

TênVịTừ(Dãy ĐốiTượng) **:-** *PhầnThânQuiTắc*(Dãy ĐốiTượng)**.**

hoặc:

TênVịTừ(Dãy ĐốiTượng) **if** *PhầnThânQuiTắc*(Dãy ĐốiTượng)**.**

trong đó: *PhầnThânQuiTắc* có thể gồm các vị từ, mệnh đề *được nối kết với nhau bởi các phép toán lôgic* như: and (“,”), or (“;”), not(*VịTừ(Đối*)), nhưng *Đối* của *VịTừ* trong *not* phải là *hằng* hoặc *biến ràng buộc*. Tất cả các vị từ đều phải được kết thúc bởi một dấu chấm “**.**”.

* ***<u>Chú ý</u>***:

- Các *vị từ cùng tên* phải được liệt kê hay định nghĩa *liên tiếp nhau*.
- Cơ chế tìm kiếm mặc định trong Prolog theo chiều sâu. *Thứ tự* của các *mệnh đề cùng tên* (*cùng các đối của chúng*) trong phần *Clauses* có thể *có ý nghĩa khác nhau* và *ảnh hưởng đến tốc độ tìm kiếm*.
- Cần để ý đến *thứ tự và độ ưu tiên* các phép toán lôgic trong *PhầnThânQuiTắc*.
- Qui tắc:

A:- B1; B2.

tương đương với:

A:- B1**.**

A:- B2**.**

- Để biểu diễn mệnh đề: **B and (C or D)**, dùng qui tắc sau là sai:

A:- B, C; D. % hoặc: A:- B, (C;D). cũng sai !

Khi đó, có nhiều cách đúng để biểu diễn chúng:

C1: A:- B, E.
E:- C; D.

hoặc *C2*: A:- B, C.
A:- B, D.

hoặc *C3*: A:- B, C; B, D.

e.* <u>Đoạn mục tiêu</u>** (đối với ***<u>Goal trong</u>), để đưa vào các *vị từ câu hỏi*, kết hợp với các phép toán lôgic, được bắt đầu bởi từ khóa *Goal*.

Nếu nhìn chương trình theo quan niệm Top-Down, đoạn này có thể *đặt trước* phần *Clauses.*

**** Ví dụ 2*** *(minh họa việc dùng Goal trong, Goal ngoài; cơ chế tìm kiếm nghiệm bội của Goal ngoài)*:

Domains

Ai = symbol

Predicates

Thích(Ai, Ai)

Clauses

Thích("A", b).

Thích(b, c).

Thích(X, Z):- Thích(X, Y), Thích(Y, Z).

Goal %trong: *đưa vào trong chương trình nguồn*

%Các đối trong vị từ đều ràng buộc:

Thích("A", b). %Kết quả: rỗng

Thích("A", c), write("Đúng"). %Kết quả: Đúng

%Có đối trong vị từ là biến tự do:

Thích("A", Ai). %Kết quả: rỗng

Thích("A", Ai), **write("A thích: ", Ai, "; ")**.

%Kết quả (xuất một lời giải): A thích: b

Thích(Ai, c), write(Ai, " thích c; "), **fail**.

% Kết quả (xuất nhiều lời giải): b thích c; A thích c

Goal %ngoài: *chỉ đưa vào cửa sổ hội thoại*

%Các đối trong vị từ đều ràng buộc:

> Thích("A", b) → yes

%Có đối trong vị từ là biến tự do:

> Thích("A", Ai) → Ai=b, Ai=c (2 solutions).

> Thích("A", Ai), **!**. → Ai=b (1 solutions).

> Thích(Ai_1, Ai_2) → *kết quả là gì ? (Bài tập)*

**** Ví dụ 3****(minh họa: các phép toán lôgic trong PhầnThânQuiTắc, các qui tắc cùng tên)*:

Domains

Ai = symbol

Predicates

Me(Ai, Ai)

Cha(Ai, Ai)

Cha(Ai)

ChaMe(Ai, Ai, Ai)

ChaHoacMe(Ai, Ai)

Nam(Ai)

Nu(Ai)

Nguoi(Ai)

Dung()
Sai

Clauses

Cha(ba_1, con_1).
Cha(ba_1, con_2).
Cha(Ba):- Cha(Ba, _).

Me(ma_1, con_1).
Me(ma_2, con_2).

ChaMe(Ba, Ma, Con) :- Cha(Ba, Con), Me(Ma, Con).

ChaHoacMe(BaMa, Con) :- Cha(BaMa, Con) **;** Me(BaMa, Con).

Nam(con_1).
Nam(Ai):- Cha(Ai).

Nguoi(Ai):- Cha(Ai) **;** Me(Ai) **;** ChaHoacMe(_, Ai).

Nu(Ai):- Nguoi(**Ai**), **not**(Nam(**Ai**)).
% Nu(Ai):- ***not***(Nam(Ai)). % *không trả lời được câu hỏi*: Nu(Ai) !?

Dung(). %Hoac: Dung():- true.
Sai:- not(Dung()).

Goal %ngoài: Cho kết quả là gì với từng goal sau ?
> Nu(con_2) → ?
> Nu(Ai) → ?

**** <u>Ví dụ 4</u>****(minh họa đối tượng phức hợp; cơ chế tìm kiếm nghiệm bội của Goal ngoài)*:

Domains

Ai, Ten = symbol
Gi = Mèo(Ten) ; Bồ(Ten) ; Rỗng

Predicates

Thich(Ai, Gi)

Clauses

Thich(a, Meo(miu)).
Thich(tu,rong).
Thich(b, Meo(miu)). Thich(b, Bo(miu)).
Thich(c, X) :- Thich(a, X) , Thich(b, X).

Goal %ngoài
> Thich(c, Gi) → Cho kết quả là gì ? (*Bài tập*)

IV.2. DANH SÁCH, ĐỆ QUI, LÁT CẮT TRONG PROLOG

Danh sách là một trong những cấu trúc dữ liệu rất quan trọng và được sử dụng thường xuyên trong *lập trình xử lý ký hiệu* bằng Prolog. *Đệ qui* là kỹ thuật thường dùng trong Prolog. Do *cơ chế tìm kiếm tự động mặc định trong Prolog theo chiều sâu*, nên, trong nhiều tình huống, nếu muốn *ngăn chặn* việc tìm kiếm tiếp tục không cần thiết, Prolog cho phép dùng *lát cắt* để thực hiện việc này.

IV.2.1. Danh sách

a. <u>Định nghĩa</u>: Danh sách là một *dãy có thứ tự* các phần tử. Một cách *đệ qui*, ta có thể định nghĩa danh sách là:

- *rỗng* (không có phần tử nào), hoặc:
- gồm *một phần tử đầu và danh sách đuôi*

b. <u>Khai báo, biểu diễn danh sách</u>:

TênKiểuDanhSách = KiểuChungCácPhầnTử*****

trong đó: *KiểuChungCácPhầnTử* có thể là kiểu dữ liệu chuẩn, kiểu do người dùng định nghĩa, kiểu phức hợp, hoặc thậm chí là danh sách.

- Danh sách *rỗng*: **[]**
- *Biểu diễn danh sách* theo *kiểu đệ qui*:

  BiếnKiểuDS = [**PhầnTử**Đầu | **DanhSách**Đuôi]
- Biểu diễn *danh sách* theo *kiểu liệt kê*:

  [**PhầnTử**ThứNhất **, PhầnTử**ThứHai | **DSách**CònLại]

Về mặt lý thuyết, có thể biểu diễn danh sách bằng *vị từ hoặc cây, biểu đồ ngang*.

**** <u>Ví dụ 1</u>***: Tìm phần tử đầu và danh sách đuôi của các danh sách sau:
[a, b, c], [a], [[1, 2], [3], [4,5]] , [] ? (*Bài tập*)

IV.2.2. Đệ qui - Cơ chế quay lui và tìm kiếm nghiệm bội trong Prolog

a. <u>Cơ chế quay lui và tìm kiếm nghiệm bội</u>: Minh họa qua ví dụ:

**** <u>Ví dụ 2</u>***:

```
%trace QHệ_1_2
QHệ_1(a, b).
QHệ_1(a, c).

QHệ_2(d, b).
QHệ_2(e, c).
QHệ_2(d, c).
```

QHệ_2(e, b).

QHệ_1_2(B, M, C) :- QHệ_1(B, C), QHệ_2(M, C).

Goal % ngoài
> QHệ_1_2(a, e, Ai) → Ai=b, Ai = c

b. <u>Định nghĩa</u>: Một ***thao tác đệ qui F(x) trên tập D*** gồm 2 phần :
. *Điều kiện dừng*: Một thao tác sơ cấp (đã biết cách thực hiện trực tiếp) trên tập con $X_o \subset D$
. Một *lời gọi đệ qui F(H(x))*, với $H(x) \in D$, sao cho sau một số hữu hạn lần gọi đệ qui sẽ phải dẫn đến điều kiện dừng ($H(H(\ldots H()\ldots)) \in X_o$, $\forall x \in D \backslash X_o$).

**** <u>Ví dụ 3</u>****(thao tác đệ qui thiếu điều kiện dừng)*:
Ba(B, C):- Con(C, B).
Con(C, B):- Ba(B, C).
Goal: > Ba(a,b) → ?

c. <u>Kỹ thuật khử đuôi đệ qui bằng cách dùng biến phụ</u>:
**** <u>Ví dụ 4</u>***: Tính *chiều dài* của một danh sách. Do *một vị từ có thể có nhiều tác dụng*, nên chú thích thêm *dòng vào-ra (I, o)* để chỉ các cách thức sử dụng vị từ cho đúng.

Domains
ptu = real
ds = ptu*
i = integer
Predicates
ChieuDai(ds,i) % (i, i) – (i, o) – *(o, i), ! – (o, o), readchar(_)*
ChieuDaiPhu(ds, i, i) % *dòng vào-ra*: *(i, i, i) – (i, i, o)*
% ChieuDaiPhu(i,0,_): luôn khởi tạo biến thứ 2 là 0
Clauses
ChieuDai([], 0).
ChieuDai([_|Duoi], Kq) :- ChieuDai(Duoi,KqPhu), Kq = KqPhu + 1.

ChieuDaiPhu([], Kq, Kq).
ChieuDaiPhu([_|Duoi], Phu, Kq) :- PhuMoi=Phu+1,
ChieuDaiPhu(Duoi, PhuMoi, Kq).
Goal %ngoài
> ChieuDai([1,2,3], 3) → ? > ChieuDai([1,2,3], Kq) → ?
> ChieuDaiPhu([1,2,3], 0, 3) → ?
> ChieuDaiPhu([1,2,3], 0, Kq) → ?
> ChieuDaiPhu(Nao, 0, 1) → ?

Bài tập: Viết 2 qui tắc đệ qui (có và không có đệ qui đuôi) tính giai thừa của một số tự nhiên.

Chú ý: *Thứ tự các vị từ cùng tên* trong phần Clauses có thể *có ý nghĩa khác nhau*, *đặc biệt là các vị từ đệ qui*. Chẳng hạn, xem tác dụng của 2 cách viết qui tắc kiểm tra danh sách có 1 phần tử sau (và sửa lại cho đúng: *bài tập*):

Cách 1:

```
DS1PTu([_|D]) :-  DS1PTu(D).
DS1PTu([]).
```

Cách 2:

```
DS1PTu([]).
DS1PTu([_|D]) :-  DS1PTu(D).
```

với cách hỏi: DS1PTu(Nào) ?

d. <u>Vài thao tác đệ qui cơ bản trên danh sách</u>:

**** <u>Ví dụ 5</u>***: Lập qui tắc kiểm tra *quan hệ thuộc giữa một phần tử và một danh sách*: Thuoc(Ptu, DSach) % (i, i) – (o, i) – (i, o) ?

```
Thuoc(PT, [PT|_]).
Thuoc(PT, [_|DSDuoi]) :- Thuộc(PT, DSDuoi).
```

**** <u>Ví dụ 6</u>***: Lập qui tắc *nối hai danh sách*: Noi(DSach, DSach, DSach) % (i, i, i) – (i, i, o) – (o, o, i) – (i, o, i) – (o, i, i) ?

```
Noi([], L, L).
Noi([PT|L1], L2, [PT|L3]) :- Noi(L1, L2, L3).

ThuocMoi(PT, DS) :- Noi(_, [PT|_], DS).
```

IV.2.3. Lát cắt (!)

Lý do dùng lát cắt: Do cơ chế tìm kiếm tự động mặc định trong Prolog theo *chiều sâu,* nhưng trong nhiều bài toán với các đặc trưng riêng, ta *không cần cơ chế quay lui* hoặc *tìm tiếp lời giải kế tiếp*. Khi đó, ta có thể dùng lát cắt để đạt được mục đích đó.

a. <u>Cú pháp, tác dụng của lát cắt</u>: Dùng dấu chấm than "**!**" để biểu diễn lát cắt. Lát cắt là *vị từ luôn đúng* và có tác dụng ***ngăn***: *cơ chế quay lui và tiếp tục tìm kiếm lời giải đối với các vị từ cùng tên với những vị từ đứng trước lát cắt* trong một mệnh đề nào đó. Điều đó dẫn đến việc *tiết kiệm thời gian lẫn bộ nhớ* khi dùng lát cắt thích hợp.

Ta minh họa lát cắt qua vài trường hợp sau:

- Đối với qui tắc không đệ qui:

VịTừ_1 :- a, !, b. **% (1)**

VịTừ_1 :- c. **% (2)**

Khi a sai thì vị từ c được thực hiện. Khi *a đúng*, sẽ *gặp "!" và thực hiện tiếp b (kể cả nghiệm bội của b, nếu b đúng)*, nhưng *sẽ không bao giờ tìm tiếp lời giải cho a và (kể cả trường hợp b sai) không quay xuống VịTừ_1 thứ hai trong (2) để kiểm tra c* , với các trị trong đối của VịTừ_1 tương ứng với lần gọi đó.

- Đối với qui tắc đệ qui:

VịTừ_2.

VịTừ_2 :- a, !, VịTừ_2. **% (3)**

VịTừ_2 :- b. **% (4)**

Khi a sai, (4) sẽ được kiểm tra. Khi *a đúng*, *VịTừ_2 trong (4) sẽ không được kiểm tra* ứng với lần gọi đó, *mặc dù VịTừ_2 trong thân qui tắc của (3) là sai* và *sẽ không bao giờ tìm tiếp lời giải cho a* .

b. <u>Các tình huống có thể sử dụng lát cắt</u>:

α- *Chỉ tìm lời giải đầu tiên ở Goal ngoài.*

β- Trong các bài toán *chỉ có duy nhất một lời giải*, *sau khi tìm được lời giải đầu tiên*, ta dùng lát cắt để *dừng ngay* quá trình tìm kiếm.

δ- Ta muốn *dừng khi một mệnh đề sai* (nhưng *không cần tìm cách khác để thỏa mãn mệnh đề*), hoặc *thay cho vị từ not*, bằng cách sử dụng *tổ hợp lát cắt* và *vị từ fail* phù hợp.

γ- Dùng lát cắt trong các chương trình "*Sinh và Thử*".

**** <u>Ví dụ 7</u>** (minh họa vài tình huống dùng lát cắt)*:

```
Domains
        i, ptu = integer
Predicates
        RangBuoc(ptu)  % minh họa tình huống δ
        GiaiThua(i, i) % minh họa tình huống β
        ThuongNguyen(i, i, i) % hai đối đầu ∈N, minh họa tình huống γ
        Sinh_1_So(i)
Clauses
    % RangBuoc(X) :- not(free(X)). % sai khi X là biến không ràng buộc
        RangBuoc(X) :- free(X), !, fail.
        RangBuoc(_).

        GiaiThua(0, 1) :- !.
        GiaiThua(N, Kq) :- N1 = N-1, GiaiThua(N1, Kq1), Kq = Kq1*N.

        ThuongNguyen(B, C, T) :- Sinh_1_So(T), T*C<=B, (T+1)*C>B, !.
```

```
Sinh_1_So(0).
Sinh_1_So(Sau) :- Sinh_1_So(Truoc), Sau = Truoc + 1.
```

.

**** Ví dụ 8*** *(Bài tập: tìm các lỗi sai, hạn chế và mở rộng các trường hợp có dòng vào-ra tốt hơn)*: Tính số cha và mẹ của mỗi người.

```
Domains
        Ai = symbol
        i = integer
Predicates
        SoChaMe(Ai, i)
Clauses
        SoChaMe("Adam", 0).
        SoChaMe("Eva", 0).
        SoChaMe(_,2).
% Gợi ý: SoChaMe("Adam", 0): !. hoặc: SoChaMe("Adam", BN): !, BN=0., …
```

**** Chú ý***:

- Phép toán **X = Y** có thể hiểu theo hai nghĩa:

. Phép ***so sánh*** nếu cả X và Y là biến (hay biểu thức) *ràng buộc*;

. Phép ***gán*** nếu chỉ *một trong hai vế là biến tự do*, còn vế kia là biểu thức hay biến ràng buộc (và sẽ vô nghĩa, bị lỗi khi cả hai vế đều không ràng buộc!)

- Có thể thay phép toán = bởi vị từ *Bằng(X,X)*.

**** Gợi ý tạo các lệnh cấu trúc như các NNLT cổ điển***:

- *if (ĐK) then A;*
  else B;

```
Neu(DK) :- !, A.
Neu( _ ) :- B.
```

- *switch (BT)*
  { case a1: A1; break;
  case a2: A2; break;

```
...
```

  case an: An; break;
  default: B; break;
  };

```
%Gọi sử dụng: Switch(BT)
Switch(a1) :- !, A1.
Switch(a2) :- !, A2.
        …
Switch(an) :- !, An.
Switch( _ ) :- B.
```

- *repeat A until (DK);*

```
Repeat :- Lap, A, DK, !.

Lap.
Lap :- Lap.
```

- *while (ĐK) do A;*

```
While :- ĐK, !, A, While.
While.
```

- *for (i=Bdau; i<=Kthuc; i = i+Step) do A(i);*

```
% Gọi sử dụng: For(Bdau, Kthuc, Step)
For(I, N, Step) :- Dấu(Step, D), D*(I-N) > 0, !.
 For(I, N, Step) :- A(i), I_Sau = I+Step, For(I_Sau, N, Step).
```

IV.3. CÁC VÍ DỤ

IV.3.1. Bài toán "Tháp Hà Nội" *(Minh họa vị từ đệ qui)*

Ví dụ 1: Có 3 cọc A, B, C. Trên cọc A có n đĩa tròn chồng lên nhau, với đường kính khác nhau, sao cho đĩa bé luôn nằm trên đĩa lớn; hai cọc B, C trống. Tìm cách chuyển n đĩa từ A đến C với số lần di chuyển đĩa ít nhất (2^n-1), sao cho: mỗi lần chỉ di chuyển 1 đĩa bé nhất trên cùng của một cọc tùy ý sang một cọc khác để đĩa bé luôn nằm trên đĩa lớn.

```
%Program Thap_HaNoi
Domains
        i  = integer
        s = symbol
Predicates
        HaNoi(i, i)
        ChuyenDia(i, i, s, s, s, i)
Clauses
        HaNoi(N, SoLanDiChuyen) :- ChuyenDia(N, N, a, c, b, SoLanDiChuyen).

        ChuyenDia(1, Nhan, Tu, Den, _, 1) :- !, nl,
                write("Chuyen dia: ", Nhan, " tu ", Tu, " den ", Den).
        ChuyenDia(N, Nhan, Tu, Den, Tgian, Tong):-
                N1=N-1, NhanPhu=Nhan-1,
                ChuyenDia(N1, NhanPhu, Tu, TGian, Den, Tong1),
                ChuyenDia(1, Nhan, Tu, Den, Tgian, 1),
                ChuyenDia(N1, NhanPhu, TGian, Den, Tu, Tong2),
```

```
        Tong = Tong1 + Tong2 + 1.
Goal %trong
    makewindow(1, 3, 7, "Thap Ha Noi", 0, 0, 25, 80),
    write("Nhap so dia n (1<=n<=12): "), readint(N), HaNoi(N, Tong),
    write("Tong so lan di chuyen dia: ", Tong)..
```

Kết quả của chương trình với n=2 là:

```
        Chuyen dia: 1 tu a den b
        Chuyen dia: 2 tu a den c
        Chuyen dia: 1 tu b den c
        Tong so lan di chuyen dia: 3
```

IV.3.2. Bài toán xử lý vi phân ký hiệu *(Minh họa bài toán xử lý ký hiệu)*:

Tính đạo hàm của một hàm số theo một biến bất kỳ.

```
%Program Xử lý vi phân ký hiệu
Domains
    KyHieu = symbol
    So     = real
    BieuThuc = bien(KyHieu); hang(So);
               cong(BieuThuc,BieuThuc); tru(BieuThuc,BieuThuc);
               nhan(BieuThuc,BieuThuc); chia(BieuThuc,BieuThuc);
               cos(BieuThuc); sin(BieuThuc);
               tg(BieuThuc); cotg(BieuThuc);
               ln(BieuThuc); log(BieuThuc,BieuThuc);
               exp(BieuThuc); mu(BieuThuc,BieuThuc);
               luyThua(BieuThuc,BieuThuc)
Predicates
    d(BieuThuc,KyHieu,BieuThuc)
Clauses
    d(hang(_),_,hang(0)).
    d(bien(X),X,hang(1)):-!.
    d(bien(_),_,hang(0)).

    d(cong(U,V),X,cong(U1,V1)):- d(U,X,U1), d(V,X,V1).
    d(tru(U,V),X,tru(U1,V1)):- d(U,X,U1), d(V,X,V1).
    d(nhan(U,V),X,cong(nhan(U1,V),nhan(U,V1))):- d(U,X,U1), d(V,X,V1).
    d(chia(U,V),X,chia(tru(nhan(U1,V),nhan(U,V1)),nhan(V,V))):-
        d(U,X,U1), d(V,X,V1).

    d(sin(U),X,nhan(cos(U),U1)) :- d(U,X,U1).
    d(cos(U),X,tru(hang(0),nhan(sin(U),U1))) :- d(U,X,U1).
    d(tg(U),X,chia(U1, mu(cos(U),hang(2)))) :- d(U,X,U1).
```

```
d(cotg(U),X,nhan(hang(-1),chia(U1, mu(sin(U),hang(2))))) :- d(U,X,U1).

d(ln(U),X,chia(U1,U)):-d(U,X,U1).
d(log(U,V),X, Kq) :- d(chia(ln(V),ln(U)),X,Kq).

d(exp(U),X,nhan(exp(U),U1)) :- d(U,X,U1).
d(mu(U,V),X,nhan(mu(U,V),W)):- d(nhan(V,ln(U)),X,W).

d(luyThua(U,hang(A)),X,nhan(hang(A),nhan(U1,luyThua(U,hang(A_1))))):-
        A_1 = A-1, d(U,X,U1).
Goal %ngoài: tính vi phan của x.sin(x)
    d(nhan(bien(x),sin(bien(x))),x, Kq).
```

Kết quả (*chưa rút gọn*) của chương trình là:

```
Kq=cong(nhan(hang(1),sin(bien(x))),nhan(bien(x),
                                        nhan(cos(bien(x)),hang(1))))
```

IV.3.3. Bài toán suy luận lôgic *(Minh họa lập trình cho bài toán lập luận lôgic)*

Ví dụ 2: Trên một đảo nọ, chỉ có hai nhóm người: một nhóm chỉ gồm những người luôn nói thật và nhóm còn lại chỉ gồm những người luôn nói láo. Một nhà thông thái đến đảo này và gặp ba người A, B, C. A nói: "Trong ba người chúng tôi chỉ có đúng một người nói thật", C nói: "Cả ba người chúng tôi đều nói láo". Hỏi trong ba người A, B, C, ai nói thật, ai nói láo ?

Ta *biểu diễn mỗi phương án* của bài toán bằng danh sách [A, B, C], sao cho mỗi phần tử chỉ gồm một trong hai trạng thái 0, 1, tương ứng với trạng thái nói láo hoặc nói thật. Ta *mô tả câu nói của A và C* như sau: nếu A nói thật thì Tổng = A+B+C = 1, nếu A nói láo thì Tổng ≠ 1; nếu C nói thật thì Tổng = A+B+C = 0, nếu C nói láo thì Tổng > 0.

```
%Program Noi lao, noi that
Domains
    i, ptu = integer
    ds  = ptu*
Predicates
    TaoDS(i,ds)
    Quet(ptu)
    Giai(i,ds)
    SuKien(symbol,ds)
    Tong(ds,integer)
    Viet(ds,i)
Clauses
    Quet(0).
```

```
Quet(1).

TaoDS(N,[]) :- N <= 0, !.
TaoDS(N,[Dau|Duoi]) :- Quet(Dau),N1 = N - 1, TaoDS(N1,Duoi).

Giai(N,DS) :- TaoDS(N,DS), SuKien("A",DS), SuKien("C",DS).

SuKien("A",DS) :- DS = [1,_,_], Tong(DS,1).
SuKien("A",DS) :- DS = [0,_,_], Tong(DS,So), So <> 1.

SuKien("C",DS) :- DS = [_,_,1], Tong(DS,0).  % phi li !
SuKien("C",DS) :- DS = [_,_,0], Tong(DS,So), So > 0.

Tong([],0) :- !.
Tong([Dau|Duoi],Kq) :- Tong(Duoi,KqDuoi), Kq = KqDuoi + Dau.

Viet([],_) :- nl,!.
Viet([1|D],I):-!,I1=I+1,I2 =I1+64,char_int(C,I2), write(C," noi that\n"),
              Viet(D,I1).
Viet([0|D],I):-!,I1=I+1,I2 =I1+64,char_int(C,I2), write(C," noi lao\n"),
              Viet(D,I1).
```

Goal *% trong*

```
Giai(3,DS),Viet(DS,0),nl.
```

Kết quả của chương trình là:

A noi that
B noi lao
C noi lao

IV.4. PHỤ LỤC: VÀI VỊ TỪ CHUẨN CỦA PROLOG

IV.4.1. NHẬP VÀ XUẤT DỮ LIỆU

Các vị từ xuất và nhập dữ liệu xét ở đây sẽ được thao tác từ các thiết bị vào ra hiện thời, ngầm định là màn hình và bàn phím.

1. Các vị từ nhập dữ liệu

Các ký hiệu sẽ sử dụng trong biểu diễn cú pháp các vị từ có ý nghĩa như sau: Sau tên vị từ các biến có các kiểu được liệt kê tiếp theo. Sau đó, dòng vào ra đối với các biến của vị từ sẽ được chỉ ra, trong đó các biến có vị trí tương ứng với ký tự:

. 'o' phải là biến tự do (sau khi vị từ được thực hiện các kết qủa của nó sẽ được gán vào các biến này);

. 'i' phải là biến ràng buộc hoặc đối tượng hằng (chúng cung cấp dữ liệu ban đầu để các vị từ này thực hiện).

a) **readln(StringVariable)**, (string) – (o): trong đó StringVariable là biến tự do có kiểu string. Vị từ này có tác dụng đọc một chuỗi từ thiết bị vào hiện thời (và tất nhiên được gán cho biến StringVariable) cho đến khi gặp ký tự xuống dòng (phím Enter, có mã ASCII tương ứng là '\13') được ấn.

b) **readint(IntgVariable)**, (integer) – (o): đọc một số nguyên từ thiết bị vào hiện thời cho đến khi gặp ký tự xuống dòng được ấn.

c) **readreal(RealVariable)**, (real) – (o): đọc một số thực từ thiết bị vào hiện thời cho đến khi gặp ký tự xuống dòng được ấn.

d) **readchar(CharVariable)**, (char) – (o): đọc một ký tự (nhưng không xuất hiện ký tự trên màn hình) từ thiết bị vào hiện thời và không cần kết thúc bằng ký tự xuống dòng.

e) **file_str(DosFileName, StringVariable)**, (string, string) – (i, o) (i, i): chuyển đổi nội dung văn bản giữa file có tên thực là DosFileName và biến chuỗi có tên StringVariable (cho đến khi gặp ký tự kết thúc file eof, tương ứng với tổ hợp phím Ctl-z hay mã ASCII là '\26').

f) **inkey(CharVariable)**, (char) – (o): đọc một ký tự từ bàn phím (gán cho biến CharVariable). Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

g) **keypressed**: kiểm tra việc một phím được bấm hay chưa nhưng không đọc ký tự này vào. Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

h) **keypressed**: kiểm tra việc một phím được bấm hay chưa nhưng không đọc ký tự này vào. Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

2. Các vị từ xuất dữ liệu

a) **write(Variable|Constant *)**: trong đó dấu * chỉ ra rằng các biến Variable (ràng buộc) hay hằng Constant có thể được lặp lại và chúng được viết cách nhau bằng các dấu phẩy ‘,’. Vị từ này có tác dụng xuất đến thiết bị ra một số đối tượng.

b) **nl**: xuống dòng.

c) **writef(FormatString, Variable|Constant *)**: xuất ra một số đối có định khuôn dạng theo cú pháp sau: %-m_p$, trong đó:

. ‘-’: canh lề bên trái, nếu không có dấu ‘-’ sẽ là canh lề bên phải;

. ‘m’: xác định độ rộng vùng nhỏ nhất cho đối;

. ‘p’: xác định độ chính xác các số sau dấu chấm thập phân hay số lớn nhất các ký tự được in ra của chuỗi;

. ‘$’ là một trong các ký hiệu quy ước chuẩn:

d: số thập phân dạng chuẩn.

x: số thập lục phân.

s: chuỗi (symbols hay strings).

c: ký tự (chars hay integers).

g: số thực dạng ngắn nhất ngầm định.

e: số thực dạng mũ.

f: số thực dạng dấu phẩy động.

d) **“\n”**: ký tự xuống dòng.

“\t”: ký tự lùi vào một số khoảng trắng.

“\nnn”: ký tự đưa vào bảng mã ASCII tương ứng.

IV.4.2. CÁC THAO TÁC FILE

a) **openread(SymbolicFileName, DosFileName)**, (file, string) – (i, i): mở một file trên đĩa DosFileName để đọc và gán cho tên file hình thức (trong phần khai báo kiểu file) SymbolicFileName.

b) **openwrite(SymbolicFileName, DosFileName)**, (file, string) – (i, i): mở một file trên đĩa để ghi và gán cho tên file hình thức; nếu file đó đã tồn tại thì nó sẽ được xóa trước khi gọi.

c) **openappend(SymbolicFileName, DosFileName)**, (file, string) – (i, i):mở một file trên đĩa để nối thêm dữ liệu và gán cho tên file hình thức.

d) **openmodify(SymbolicFileName, DosFileName)**, (file, string) – (i, i):mở một file trên đĩa để sửa đổi (đọc và ghi) và gán cho tên file hình thức. Khi dùng vị từ này, nên kết hợp với vị từ filepos để cập nhật file tại vị trí tùy chọn.

e) **readdevice(SymbolicFileName)**, (file) – (i) (o): đặt lại hay hiển thị thiết bị đọc vào file đang mở (để đọc hay sửa đổi), ngầm định là bàn phím.

f) **writedevice(SymbolicFileName)**, (file) – (i) (o): đặt lại hay hiển thị thiết bị ghi vào file đang mở (để đọc ghi hoặc nối thêm hay sửa đổi), ngầm định là màn hình.

g) **closefile(SymbolicFileName)**, (file) – (i): đóng file (dù nó đã mở hay chưa).

h) **filepos(SymbolicFileName, FilePosition, Mode)**, (file, real, integer) – (i, i, i) (i, o, i): đưa con trỏ đến hay trả lại giá trị của vị trí FilePosition trong file tính từ chế độ Mode: 0: đầu file, 1: vị trí hiện tại trong file, 2: cuối file.

i) **eof(SymbolicFileName)**, (file) – (i): kiểm tra con trỏ có ở cuối file hay không. Vị từ này đúng nếu con trỏ ở vị trí cuối file.

j) **existfile(DosFileName)**, (string) – (i): vị từ này thành công nếu file có tên chỉ ra tồn tại trong thư mục hiện thời.

k) **deletefile(DosFileName)**, (string) – (i): xóa file có tên chỉ ra.

l) **renamefile(OldDosFileName, NewDosFileName)**, (string, string) – (i, i): đổi tên file cũ thành tên file mới.

m) **disk(DosPath)**, (string) – (i) (o): thiết lập hay trả lại ổ đĩa và đường tìm kiếm thư mục mặc định.

IV.4.3. CÁC THAO TÁC ĐỐI VỚI MÀN HÌNH VÀ CỬA SỔ

1. ***Màn hình***

Màu nền	**Giá trị**	**Màu chữ**	**Giá trị**
Black	0	Black	0
Blue	16	Blue	1
Green	32	Green	2
Cyan	48	Cyan	3
Red	64	Red	4
Magenta	80	Magenta	5
Brown	96	Brown	6

White	112	White	7
		Grey	8
		Light Blue	9
		Light Green	10
		Light Cyan	11
		Light Red	12
		Light Magenta	13
		Yellow	14
		White (High Intensity)	15

a) **attribute(Attr)**, (integer) – (i) (o): cho (đặt hay trả lại) giá trị thuộc tính của màn hình. Giá trị đặc trưng cho thuộc tính của màn hình, dựa vào bảng trên đây, được tính như sau:
   1. Chọn một màu chữ và một màu nền;
   2. Cộng những giá trị nguyên tương ứng với màu trong bảng 1;
   3. Cộng thêm giá trị đó với 128 nếu muốn các chữ hiển thị nhấp nháy.

b) **scr_char(Row, Column, Char)**, (integer, integer, char) – (i, i, i) (i, i, o): cho ký tự Char trên màn hình tại vị trí có tọa độ (Row, Column).

c) **scr_attr(Row, Column, Char)**, (integer, integer, char) – (i, i, i) (i, i, o): cho thuộc tính của màn hình tại vị trí có tọa độ (Row, Column).

d) **filed_str(Row, Column, Length, String)**, (integer, integer, integer, string) – (i, i, i, i) (i, i, i, o): cho chuỗi String trên màn hình tại một vùng bắt đầu ở vị trí có tọa độ (Row, Column) và có độ dài là Length ký tự.

e) **filed_attr(Row, Column, Length, Attr)**, (integer, integer, integer, integer) – (i, i, i, i) (i, i, i, o): cho thuộc tính của màn hình tại một vùng bắt đầu ở vị trí có tọa độ (Row, Column) và có độ dài là Length ký tự.

f) **cursor(Row, Column)**, (integer, integer) – (i, i) (i, o): dịch con trỏ đến vị trí có tọa độ (Row, Column) hay trả lại tọa độ của con trỏ hiện thời.

2. Hệ thống cửa sổ

a) **makewindow(WindowNo, ScrAtt, FrameAtt, FrameStr, Row, Column, Height, Width)**, (integer, integer, integer, string, integer, integer, integer, integer) – (i, i, i, i, i, i, i, i) (o, o, o, o, o, o, o, o): xác định một vùng màn hình làm cửa sổ.

b) **makewindow(WindowNo, ScrAtt, FrameAtt, FrameStr, Row, Column, Height, Width, ClearWindow, FrameStrPos, BorderChars)**, (integer, integer, integer, string, integer, integer, integer, integer, integer, integer, string) – (i, i, i, i, i, i, i, i, i, i) (o, o, o, o, o, o, o, o, o, o): xác định một vùng màn hình làm cửa sổ với các thuộc tính sau:

ClearWindow	= 0	không xóa cửa sổ sau khi tạo,
	= 1	xóa cửa sổ sau khi tạo;
FrameStrPos	= 255	canh tít ở giữa,
	<> 255	đặt tít tại vị trí FrameStrPos;
BorderChars	một chuỗi gồm 6 ký tự để vẽ khung:	
	ký tự thứ 1: góc trên bên trái,	
	ký tự thứ 2: góc trên bên phải,	
	ký tự thứ 3: góc dưới bên trái,	
	ký tự thứ 4: góc dưới bên phải,	
	ký tự thứ 5: đường kẻ ngang,	
	ký tự thứ 6: đường kẻ dọc.	

c) **shiftwindow(WindowNo),** (integer) – (i) (o): đổi cửa sổ làm việc đến cửa sổ thứ WindowNo (vẫn giữ nguyên trạng thái của cửa sổ trước đó) hay trả lại số của cửa sổ đang làm việc.

d) **gotowindow(WindowNo),** (integer) – (i): di chuyển nhanh đến cửa sổ thứ WindowNo mà không lưu nội dung của cửa sổ cũ vào bộ đệm cửa sổ.

e) **resizewindow**: điều chỉnh lại kích thước cửa sổ như kiểu thực đơn.

f) **resizewindow(StartRow, NoOfRows, StartCol, NoOfCols)**, (integer, integer, integer, integer) – (i, i, i, i): điều chỉnh lại kích thước cửa sổ một cách trực tiếp.

g) **colorsetup(Main_Frame)**, (integer) – (i): thay đổi màu cửa sổ hay khung tùy theo giá trị của:

Main_Frame	= 0	đổi màu cửa sổ
	= 1	đổi màu khung

h) **existwindow(WindowNo)**, (integer) – (i): vị từ này thành công nếu tồn tại cửa sổ số WindowNo.

i) **removewindow**: loại bỏ cửa sổ hiện thời.

j) **removewindow(WindowNo, Refresh)**, (integer, integer) – (i, i): loại bỏ cửa sổ số WindowNo với chế độ xóa nền hay không xóa nền (nếu Refresh bằng 1 hoặc 0).

k) **clearwindow**: xóa cửa sổ hiện thời về màu nền của nó.

l) **window_str(ScreenString)**, (string) – (i) (o): chuyển đổi nội dung giữa cửa sổ hiện thời và chuỗi ScreenString.

m) **window_at(Attribute**), (integer) – (i): xác định thuộc tính cho cửa sổ hiện thời.

n) **scroll(NoOfRows, NoOfCols**), (integer, integer) – (i, i): cuộn nội dung của cửa sổ lên (hay xuống) X=NoOfRows dòng tùy theo nó > 0 (hay < 0) và sang trái (hay phải) X=NoOfCols cột tùy theo nó > 0 (hay < 0).

o) **framewindow(FrameAttr**), (integer) – (i): thay đổi thuộc tính của khung cửa sổ.

p) **framewindow(FrameAttr, FrameStr, FrameStrPos, FrameTypeStr**), (integer, string, integer, integer) – (i, i, i, i): thay đổi thuộc tính của khung cửa sổ theo các chế độ sau:

FrameAttr	= thuộc tính khung
FrameStr	= tít của khung
FrameStrPos	= nhận giá trị từ 0 đến độ rộng cửa sổ. Nó đặt vị trí cho tít trên khung. Tít sẽ đặt ở giữa nếu nó bằng 255
FrameTypeStr	= một chuỗi gồm 6 ký tự để chỉ dạng của khung.

IV.4.4. CÁC VỊ TỪ CHUẨN KHÁC

1. Thao tác chuỗi

a) **frontchar(String, FrontChar, RestString)**, (string, char, string) – (i, o, o) (i, i, o) (i, o, i) (i, i, i) (o, i, i): tách một chuỗi thành ký tự đầu và chuỗi còn lại.

b) **fronttoken(String, Token, RestString)**, (string, string, string) – (i, o, o) (i, i, o) (i, o, i) (i, i, i) (o, i, i): tách một chuỗi thành một 'từ' và chuỗi còn lại.

c) **frontstr(Length, InpString, StartString, RestString)**, (integer, string, string, string) – (i, i, o, o): tách từ một chuỗi cho trước InpString ra một chuỗi con StartString bắt đầu từ ký tự thứ nhất có độ dài là Length và chuỗi còn lại RestString.

d) **concat(String1, String2, String3)**, (string, string, string) – (i, i, o) (i, o, i) (o, i, i) (i, i, i): nối String1 và String2 thành String3: String3 = String1 + String2.

e) **strlen(String, Length)**, (string, integer) – (i, i) (i, o) (o, i): chiều dài của String là Length. Với dòng vào ra (o, i) ta được một chuỗi gồm Length ký tự trắng.

f) **isname(StringParam)**, (string) – (i): kiểm tra chuỗi có là một 'name' (một dãy các ký tự chữ hoặc số hoặc dấu '_' nhưng bắt đầu bằng chữ hay ký tự '_' trong Turbo Prolog không.

2. Chuyển đổi kiểu

a) **char_int(CharParam, IntgParam)**, (char, integer) – (i, o) (o, i) (i, i): chuyển đổi giữa một ký tự và mã ASCII tương ứng với nó.

b) **str_int(StringParam, IntgParam)**, (string, integer) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một số nguyên tương ứng với nó.

c) **str_char(StringParam, CharParam)**, (string, char) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một ký tự tương ứng với nó.

d) **str_real(StringParam, RealParam)**, (string, real) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một số thực tương ứng với nó.

e) **upper_lower(StringInUpperCase, StringInLowerCase)**, (string, string) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi ký tự hoa và một chuỗi ký tự thường.

f) **upper_lower(CharInUpperCase, CharInLowerCase)**, (char, char) – (i, o) (o, i) (i, i): chuyển đổi giữa một ký tự hoa và một ký tự thường.

3. Thao tác dối với cơ sở dữ liệu trong

a) **consult(DosFileName)**, (string) – (i): gọi ra và bổ sung một file cơ sở dữ liệu trong (file đó được khởi tạo và lưu một cơ sở dữ liệu trong không tên bằng vị từ save(?)).

b) **consult(DosFileName, InternalDatabaseName)**, (string, InternalDatabaseName) – (i, i): gọi ra và bổ sung một file cơ sở dữ liệu trong (file đó được khởi tạo và lưu một cơ sở dữ liệu trong không tên bằng vị từ save(?, ?)).

c) **save(DosFileName)**, (string) – (i): lưu mọi mệnh đề của cơ sở dữ liệu trong không tên.

d) **save(DosFileName, InternalDatabaseName)**, (string, InternalDatabaseName) – (i, i): lưu mọi mệnh đề của cơ sở dữ liệu trong có tên.

e) **assert(Term)**, (InternalDatabaseDomain) – (i) hay **asserta(Term)**, (InternalDatabaseDomain) – (i): chèn một sự kiện vào đuôi một cơ sở dữ liệu trong không tên.

f) **assertz(Term)**, (InternalDatabaseDomain) – (i): chèn một sự kiện vào đuôi một cơ sở dữ liệu trong không tên.

g) **retract(Term)**, (InternalDatabaseDomain) – (_): vị từ không tất định (nondeterm) này loại bỏ sự kiện chỉ ra của cơ sở dữ liệu trong không tên.

h) **retract(Term, InternalDatabaseDomain)**, (InternalDatabaseDomain) – (_, i): vị từ không tất định (nondeterm) này loại bỏ sự kiện chỉ ra của cơ sở dữ liệu trong có tên.

i) **retractall(Term)**, (InternalDatabaseDomain) – (_): vị từ này loại bỏ mọi sự kiện của cơ sở dữ liệu trong không tên. Nên sử dụng biến vô danh trong vị từ này. Vị từ này không bao giờ sai.

j) **retractall(_, InternalDatabaseDomain)**, (_, InternalDatabaseDomain) – (_, i): vị từ này loại bỏ mọi sự kiện của cơ sở dữ liệu trong có tên. Nên sử dụng biến vô danh trong vị từ này.

4. Các lệnh liên quan đến hệ điều hành DOS

a) **system(DosCommandString)**, (string) – (i): thực hiện một lệnh của hệ điều hành DOS trong môi trường Turbo Prolog.

b) **dir(Path, Filespec, Filename)**, (string, string, string) – (i, i, o): xem các files trong thư mục và chọn một file bằng menu để xem nội dung.

c) **exit**: thoát ra một chương trình và trở về môi trường làm việc của Prolog.

5. Các lệnh linh tinh khác

a) **random(RealVariable)**, (real) – (o): tạo một số ngẫu nhiên thực trong khoảng [0, 1).

b) **random(MaxValue, RandomInt)**, (integer, integer) – (i, o): tạo một số nguyên ngẫu nhiên trong khoảng [0, RandomInt).

c) **sound(Duration, Frequency)**, (integer, integer) – (i, i): tạo ra âm thanh có trường độ Duration và tần số Frequency.

d) **beep**: tạo ra một tiếng còi ngắn.

e) **date(Year, Month, Day)**, (integer, integer, integer) – (o, o, o) (i, i, i): cho ngày, tháng, năm của đồng hồ hệ thống.

f) **time(Hours, Minutes, Seconds, Hundredths)**, (integer, integer, integer, integer) – (o, o, o, o) (i, i, i, i): cho giờ của đồng hồ hệ thống.

g) **trace(on/off)**, (string) – (o) (i): đặt chế độ lần vết hay không cho chương trình.

h) **findall(Variable, Atom, ListVariable)**: ghi các giá trị của biến trong Atom trùng tên với Variable vào danh sách ListVariable.

i) **not(Atom)**: vị từ này có giá trị logic phủ định của vị từ Atom. Lưu ý: không được dùng biến tự do trong vị từ Atom.

j) **free(Variable)**: vị từ này chỉ đúng khi Variable là biến tự do.

k) **bound(Variable)**: vị từ này chỉ đúng khi Variable là biến ràng buộc.

l) **fail**: vị từ này luôn luôn sai.

m) **true**: vị từ này luôn luôn đúng.

n) **Các phép toán số học trong Turbo Prolog**: +, -, *, /, mod, div.

o) **Các phép toán quan hệ trong Turbo Prolog** (có giá trị không chỉ đối với các số mà còn đối với các ký tự, chuỗi, symbol): >, <, =, >=, <=, <>, ><.

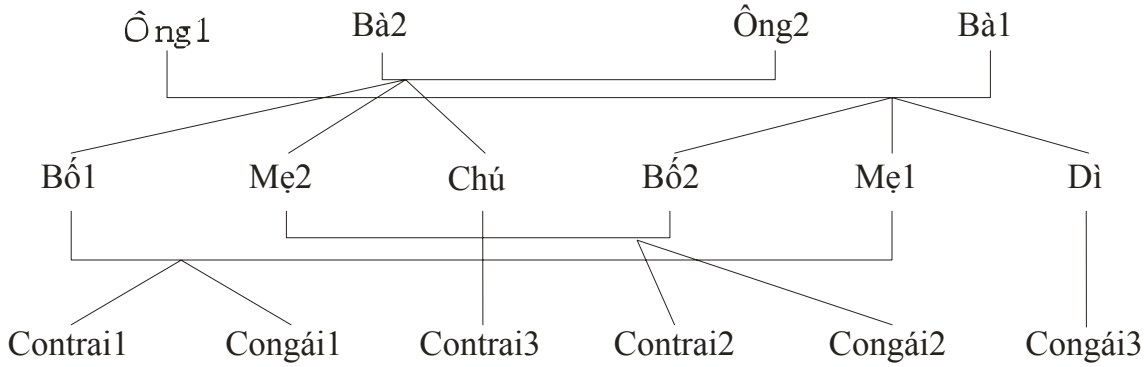
p) **Các hàm trong Turbo Prolog**: sin, cos, tan, arctan, ln, log, exp, sqrt, round (số nguyên làm tròn), trunc (số nguyên chặt cụt), abs.

BÀI TẬP CHƯƠNG IV

(Phần bài tập làm quen với vị từ trong Prolog)

Bài 1:

Xét cây gia phả sau:



Cây được mô tả bởi các vị từ quan hệ sau: cha(Cha, Con), mẹ(Mẹ, Con), nam(DO), nữ (DB).

Hãy liệt kê các quan hệ còn lại như: anhEm(Ng1, Ng2), con(Con, Cha, Mẹ), chú(Chú, Cháu), bà(Bà, Cháu), anhEmHọ(Ng1, Ng2)... từ những vị từ trên.

Suy ra mối quan hệ giữa hai người bất kỳ.

Bài 2:

Một dịch vụ du lịch phục vụ những chuyến đi trong 1 tuần hoặc 2 tuần ở Rome, London, Tunis.

Trong một quyển sách giới thiệu về dịch vụ du lịch, với mỗi địa điểm có giá cả đi lại (tùy vào thời gian của chuyến đi) và chi phí cho 1 tuần, tùy vào địa điểm và mức độ tiện nghi: khách sạn, nhà trọ, cắm trại ngoài trời.

a) Nhập vào các sự kiện mô tả quyển sách trên (giá cả do bạn nhập vào).
b) Hãy mô tả quan hệ ChuyếnĐi(TP, SốTuần, TiệnNghi, ChiPhí).
c) Mô tả quan hệ ChuyếnĐi_ÍtTốnKém(TP, SốTuần, TiệnNghi, ChiPhí, ChiPhíMax) sao cho chi phí của chuyến đi thấp hơn ChiPhíMax.

Bài 3:

Trong một dịch vụ tư vấn hôn nhân, liệt kê một danh sách các cặp xứng đôi có thể làm quen theo các nhận định sau:

- nam(n, t, c, a), nữ(n, t, c, a).
  n: tên; t: chiều cao.
  c: màu tóc (vàng, hung, nâu, hạt dẻ).
  a: tuổi (trẻ, trung niên, già).

- SởThích(n, m, l, s) (sở thích) có nghĩa là: Người n thích loại nhạc m (classique, pop, jazz), thích văn học loại l (phiêu lưu, văn học viễn tưởng, trinh thám), và chơi môn thể thao s(quần_vợt, bơi, chạy bộ).
- tìm(n, t, c, a): Người n tìm bạn có các tính chất t, c, a.

Hai người x và y gọi là hợp nhau nếu có cùng ngoại hình (t, c, a) và sở thích về: l, m, s.

a) Nhập dữ liệu về các khách hàng.
b) Mô tả qui tắc hợp_ngoạihình(x, y) và cùng_sở thích(x, y).
c) Viết chương trình xác định những cặp xứng đôi.

(Phần bài tập về danh sách, đệ qui, lát cắt)

Bài 4:

Xây dựng các vị từ đệ quy (có hoặc không sử dụng nhát cắt và sửa đổi phần đuôi của đệ quy, nếu có thể) để:

a) Tìm (hoặc xóa) phần tử thứ 1, thứ 2, phần tử cuối cùng trong một danh sách.
b) Tìm (hoặc lấy ra) phần tử đầu tiên trong danh sách trùng với một giá trị cho trước.
c) Tìm và lấy ra tất cả các phần tử trong danh sách trùng với một giá trị cho trước.
d) Tính giá trị trung bình cộng của một danh sách kiểu số thực.
e) Xuất ra tất cả các cặp phần tử kế tiếp nhau của danh sách.
f) Tạo danh sách tất cả các phần tử ở vị trí chẵn của danh sách dữ liệu L:
    - Giữ nguyên trật tự xuất hiện trong L.
    - Đảo ngược trật tự.
g) Viết chương trình trộn(u, v, w) nhận xen kẽ từng phần của u và v để tạo thành danh sách w.
h) Tạo danh sách chứa các số chẵn của một danh sách số nguyên cho trước.
i) Đảo ngược trật tự mọi phần tử của một danh sách.
j) Xuất ra mọi hoán vị của những phần tử của danh sách cho trước.
k) Xây dựng vị từ Thuộc trong đó có thêm một đối để trả về trị TRUE nếu phần tử x thuộc danh sách l cho trước và FALSE trong trường hợp ngược lại.
l) Xây dựng qui tắc NằmNgoài(x, L) để kiểm tra xem x không thuộc danh sách L.
m) Xây dựng qui tắc KhácNhau(L) để kiểm tra danh sách L không có phần tử trùng nhau.
n) Xây dựng qui tắc PhầnĐầu(m, L) để kiểm tra danh sách m là phần đầu của danh sách L.
o) Xây dựng qui tắc NằmTrongLiênTiếp(m, L) để kiểm tra mọi phần tử của m đều nằm liên tiếp trong danh sách L.

p) Xây dựng qui tắc PhânHoạch(L, u, v, w) để phân hoạch danh sách L thành 3 danh sách u, v, w.
q) Thêm một phần tử vào cuối danh sách.
r) Tìm và xóa phần tử thứ k của danh sách l cho trước.
s) Chèn một phần tử x sau phần tử thứ k của danh sách L (0 <= k <= length(L)).
t) Xây dựng danh sách mà các phần tử là căn bậc hai của các phần tử tương ứng trong danh sách L các số thực cho trước.

Bài 5:

Tìm n bộ 3 số tự nhiên (x, y, z) đầu tiên thỏa tính chất Pythagore sau:

$0 < x < y < z$

$x^2 + y^2 = z^2$

UCLN(x, y, 1): Ước chung lớn nhất của x và y bằng 1.

Bài 6 (+):

Tìm n_k số tự nhiên đầu tiên $\overline{a_1, a_2, ..., a_n}$ thỏa:

$$\overline{a_1.a_2.a_n} = \sum_{i=1}^{n} a_i^k \qquad (k = 3, 4, 5)$$

VD: k = 3, $153 = 1^3 + 5^3 + 3^3$.

Bài 7:

Kiểm tra số tự nhiên N có phải là số nguyên tố không? Tìm n số nguyên tố đầu tiên.

Bài 8:

Tìm các số nguyên tố <= n (sàng Eratosthene).

Bài 9:

a) Nhập 2 danh sách số nguyên dương không có phần tử trùng lặp từ bàn phím.
b) Xây dựng các phép toán và các quan hệ trên tập hợp: Hội, Giao, Hiệu, Thuộc, Bao hàm.

Bài 10:

Cài đặt các thuật toán sắp xếp: chọn, chèn, tráo đổi đơn giản và sắp xếp nhanh.

Bài 11 (+):

Các đỉnh của một đồ thị được đánh số từ 0 -> n. Mỗi cung của đồ thị tương ứng với qui tắc Cung(i, j) nối gốc i với ngọn j.

Xây dựng qui tắc ĐườngĐi(x, y, L) với L là danh sách biểu diễn đường đi không lặp từ x đến y.

Bài 12 (+):

Cài đặt cây nhị phân tìm kiếm và các thao tác cơ bản.

Bài 13:

Giải quyết bài toán "Tháp Hà Nội" bằng đệ quy.

Bài 14 (*): (8 hậu)

Đặt 8 con hậu lên bàn cờ sao cho không con nào ăn con nào.

Bài 15 (*): (Dominos)

Có một danh sách các quân cờ đôminô. Mỗi quân cờ đôminô được biểu diễn bởi Cặp(i, j) ứng với 2 con số của nó. Người ta tìm cách sắp xếp chúng theo tính chất sau: Mỗi con đôminô có thể xếp vào một trong hai đầu của hàng đã xếp.

Xuất phát từ một con đôminô trong số các con được cho, hãy xây dựng một dãy hợp lệ chứa tất cả các đôminô.

Bài 16 (*):

Giải quyết bài toán ma phương cấp n lẻ.

(Phần bài tập về trò chơi, suy luận lôgic)

Bài 17: (Ngựa vằn)

Có 5 người quốc tịch khác nhau sống trong 5 căn nhà màu khác nhau. Mỗi người có một con thú riêng, một thức uống riêng, và hút thuốc của những hãng khác nhau.

- Người Anh sống trong căn nhà màu đỏ.
- Con chó thuộc về người Tây Ban Nha.
- Người uống cà phê trong căn nhà màu xanh lá cây.
- Người Ukrain uống trà.
- Ngôi nhà xanh lá cây ở cạnh ngôi nhà trắng, bên phải.
- Người hút thuốc OldGold nuôi những con sên.
- Người uống sữa ở ngôi nhà chính giữa.
- Người hút thuốc Kool sống trong căn nhà màu vàng.
- Người Nauy sống trong căn nhà đầu tiên bên trái.
- Người hút thuốc Chesterfield sống cạnh người có nuôi cáo.
- Người hút thuốc Kool sống cạnh người có nuôi ngựa.
- Người hút thuốc Gitanes uống rượu.
- Người Nhật hút thuốc Craven
- Người Nauy sống cạnh ngôi nhà màu xanh da trời.

Hỏi: Ai uống nước? Ai là chủ con ngựa vằn?

Bài 18 (*): (Trò chơi tốt đen và tốt trắng)

Bắt đầu từ 3 tốt đen và 3 tốt trắng đã được sắp xếp và cách nhau bởi một khoảng trắng.

VD: BBB_NNN.

Mỗi lần đánh chỉ có 1 trong 4 phép dịch chuyển sau được thi hành:

Tốt đen trượt sang trái:

BNB_NNB -> BNBN_NB

Tốt đen nhảy sang trái:

BN_BNNB -> BNNB_NB

Tốt trắng trượt sang phải:

BNB_NNB -> BN_BNNB

Tốt trắng nhảy sang phải:

BBN_NNB -> B_NBNNB

Viết 1 chương trình để chuyển từ trạng thái đầu sang trạng thái kết (mọi tốt trắng đều nằm bên phải).

Bài 19 (*):

Cho 1 biểu thức với phép * (Nhan) và + (Cong). Hãy sửa đổi để được 1 tổng các tích bằng cách thêm vào dấu ngoặc.

VD: a * (b + c) + (d + e) * (f + g) = (((((a * b + a * c) + d * f) + e * f) + d * g) + e * g)

Có nghĩa là:

Cong(Nhan(a, Cong(b, c), Nhan(Cong(d, e), Cong(f, g)))) cho ra Cong(Cong(Cong(Cong(Cong(Nhan(a, b), Nhan(a, c)), Nhan(d, f)), Nhan(e, f)), Nhan(d, g)), Nhan(e, g))

Hãy xây dựng chương trình trên theo 2 giai đoạn:

- Phân phối Nhan sao cho không có Nhan nào có Cong làm đối.
- Chuyển thành cây sao cho nhánh phải của Cong không chứa Cong khác, nhánh phải của Nhan không chứa Nhan khác.

Bài 20 (*):

Cho biểu thức dạng And(p, q), Or(p, q), Not(q), trong đó p, q là biến hoặc là biểu thức cùng dạng. Ta có:

And(Or(p, q), r) <=> Or(And(p, r), And(q, r))
And(p, Or(q, r)) <=> Or(And(p, q), And(p, r))
Not(Or(p, q)) <=> And(Not(p), Not(q))
Not(And(p, q)) <=> Or(not(p), not(q))
Not(Not(p)) <=> p

Hãy chuyển biểu thức theo yêu cầu sau:

Mỗi nút Or chỉ có tổ tiên là Or.
Mỗi nút And chỉ có tổ tiên là And.

a) Viết chương trình thực hiện công việc trên.
b) Thay đổi chương trình để mỗi nhánh trái Or không chứa nút Or khác, và mỗi nhánh trái And không chứa And khác.

Bài 21 (*): (Những ông chồng hay ghen)

Có ba cặp vợ chồng phải qua sông nhưng chỉ có một chiếc thuyền chỉ chở được tối đa hai người. Hãy lập trình cho các chuyến đi để sao cho không có bà vợ nào ở trên bờ hay trên thuyền với những người đàn ông X khác mà không có chồng mình hoặc không có đồng thời với vợ của X.

Bài 22 (*): (Sói, dê và bắp cải)

Có một con sói, một con dê và một bắp cải phải qua sông. Chỉ có một người chèo thuyền, chỉ chở được mỗi lần một trong số chúng. Nếu không có mặt người chèo thuyền thì dê sẽ ăn bắp cải và sói sẽ ăn thịt dê. Hãy lập trình cho các chuyến đi.

TÀI LIỆU THAM KHẢO

[1] Bạch Hưng Khang, Hoàng Kiếm. *Trí tuệ nhân tạo - Các phương pháp và ứng dụng*. NXB Khoa học Kỹ thuật, 1989.
[2] Hoàng Kiếm. *Giải một bài toán trên máy tính như thế nào (tập 1 và 2).* NXB Giáo dục, 2001.
[3] Nguyễn Thanh Thủy. *Trí tuệ nhân tạo - Các phương pháp giải quyết vấn đề và kỹ thuật xử lý tri thức.* NXB Giáo dục, 1995.
[4] A. Thayse. *Approche logique de l'intelligence artificielle (T1, T2, T3).* Dunod, Paris, 1990.
[5]. Ivan Bratko, *Prolog - Programming for artificial intelligence,* Addison-Wesley, 1990.

Giáo trình

TRÍ TUỆ NHÂN TẠO

ARTIFICIAL INTELLIGENCE

Phạm Thọ Hoàn, Phạm Thị Anh Lê

Khoa Công nghệ thông tin

Trường Đại học Sư phạm Hà Nội

Hà nội, 2011

MỤC LỤC

Chương 1 – Giới thiệu

1. Trí tuệ nhân tạo là gì?

Để hiểu trí tuệ nhân tạo (artificial intelligence) là gì chúng ta bắt đầu với khái niệm sự bay nhân tạo (flying machines), tức là cái máy bay.

Đã từ lâu, loài người mong muốn làm ra một cái máy mà có thể di chuyển được trên không trung mà không phụ thuộc vào địa hình ở dưới mặt đất, hay nói cách khác là máy có thể bay được. Không có gì ngạc nhiên khi những ý tưởng đầu tiên làm máy bay là từ nghiên cứu cách con chim bay. Những chiếc máy biết bay được thiết kế theo nguyên lý "vỗ cánh" như con chim chỉ có thể bay được quãng đường rất ngắn và lịch sử hàng không thực sự sang một trang mới kể từ anh em nhà Wright thiết kế máy bay dựa trên các nguyên lý của khí động lực học (aerodynamics).

Các máy bay hiện nay, như đã thấy, có sức trở rất lớn và bay được quãng đường có thể vòng quanh thế giới. Nó không nhất thiết phải có nguyên lý bay của con chim nhưng vẫn bay được như chim (dáng vẻ), và còn tốt hơn chim.

Quay lại câu hỏi Trí tuệ nhân tạo là gì. Trí tuệ nhân tạo là trí thông minh của máy do con người tạo ra. Ngay từ khi chiếc máy tính điện tử đầu tiên ra đời, các nhà khoa học máy tính đã hướng đến phát hiển hệ thống máy tính (gồm cả phần cứng và phần mềm) sao cho nó có khả năng thông minh như loài người. Mặc dù cho đến nay, theo quan niệm của người viết, ước mơ này vẫn còn xa mới thành hiện thực, tuy vậy những thành tựu đạt được cũng không hề nhỏ: chúng ta đã làm được các hệ thống (phần mềm chơi cờ vua chạy trên siêu máy tinh GeneBlue) có thể thắng được vua cờ thế giới; chúng ta đã làm được các phần mềm có thể chứng minh được các bài toán hình học; v.v. Hay nói cách khác, trong một số lĩnh vực, máy tính có thể thực hiện tốt hơn hoặc tương đương con người (tất nhiên không phải tất cả các lĩnh vực). Đó chính là các hệ thống thông minh.

Có nhiều cách tiếp cận để làm ra trí thông minh của máy (hay là trí tuệ nhân tạo), chẳng hạn là nghiên cứu cách bộ não người sản sinh ra trí thông minh của loài người như

thế nào rồi ta bắt chước nguyên lý đó, nhưng cũng có những cách khác sử dụng nguyên lý hoàn toàn khác với cách sản sinh ra trí thông minh của loài người mà vẫn làm ra cái máy thông minh như hoặc hơn người; cũng giống như máy bay hiện nay bay tốt hơn con chim do nó có cơ chế bay không phải là giống như cơ chế bay của con chim.

Như vậy, trí tuệ nhân tạo ở đây là nói đến khả năng của máy khi thực hiện các công việc mà con người thường phải xử lý; và khi dáng vẻ ứng xử hoặc kết quả thực hiện của máy là tốt hơn hoặc tương đương với con người thì ta gọi đó là máy thông minh hay máy đó có trí thông minh. Hay nói cách khác, đánh giá sự thông minh của máy không phải dựa trên nguyên lý nó thực hiện nhiệm vụ đó có giống cách con người thực hiện hay không mà dựa trên kết quả hoặc dáng vẻ ứng xử bên ngoài của nó có giống với kết quả hoặc dáng vẻ ứng xử của con người hay không.

Các nhiệm vụ của con người thường xuyên phải thực hiện là: ***giải bài toán*** (tìm kiếm, chứng minh, lập luận), ***học***, ***giao tiếp***, ***thể hiện cảm xúc***, ***thích nghi với môi trường xung quanh***, v.v., và dựa trên kết quả thực hiện các nhiệm vụ đó để kết luận rằng một ai đó có là thông minh hay không. Môn học Trí tuệ nhân tạo nhằm cung cấp các phương pháp luận để làm ra hệ thống có khả năng thực hiện các nhiệm vụ đó: giải toán, học, giao tiếp, v.v. bất kể cách nó làm có như con người hay không mà là kết quả đạt được hoặc dáng vẻ bên ngoài như con người.

Trong môn học này, chúng ta sẽ tìm hiểu các phương pháp để làm cho máy tính biết cách giải bài toán, biết cách lập luận, biết cách học, v.v.

2. Lịch sử

Vào năm 1943, Warren McCulioch và Walter Pitts bắt đầu thực hiện nghiên cứu ba cơ sở lý thuyết cơ bản: triết học cơ bản và chức năng của các noron thần kinh; phân tích các mệnh đề logic; và lý thuyết dự đoán của Turing. Các tác giả đã nghiên cứu đề xuất mô hình noron nhân tạo, mỗi noron đặc trưng bởi hai trạng thái "bật", "tắt" và phát hiện mạng noron có khả năng học.

Thuật ngữ “Trí tuệ nhân tạo” (Artificial Intelligence - AI) được thiết lập bởi John McCarthy tại Hội thảo đầu tiên về chủ đề này vào mùa hè năm 1956. Đồng thời, ông cũng đề xuất ngôn ngữ lập trình Lisp – một trong những ngôn ngữ lập trình hàm tiêu biểu, được sử dụng trong lĩnh vực AI. Sau đó, Alan Turing đưa ra "Turing test" như là một phương pháp kiểm chứng hành vi thông minh.

Thập kỷ 60, 70 Joel Moses viết chương trình Macsyma - chương trình toán học sử dụng cơ sở tri thức đầu tiên thành công. Marvin Minsky và Seymour Papert đưa ra các chứng minh đầu tiên về giới hạn của các mạng nơ-ron đơn giản. Ngôn ngữ lập trình logic Prolog ra đời và được phát triển bởi Alain Colmerauer. Ted Shortliffe xây dựng thành công một số hệ chuyên gia đầu tiên trợ giúp chẩn đoán trong y học, các hệ thống này sử dụng ngôn ngữ luật để biểu diễn tri thức và suy diễn.

Vào đầu những năm 1980, những nghiên cứu thành công liên quan đến AI như các hệ chuyên gia (expert systems) – một dạng của chương trình AI mô phỏng tri thức và các kỹ năng phân tích của một hoặc nhiều chuyên gia con người

Vào những năm 1990 và đầu thế kỷ 21, AI đã đạt được những thành tựu to lớn nhất, AI được áp dụng trong logic, khai phá dữ liệu, chẩn đoán y học và nhiều lĩnh vực ứng dụng khác trong công nghiệp. Sự thành công dựa vào nhiều yếu tố: tăng khả năng tính toán của máy tính, tập trung giải quyết các bài toán con cụ thể, xây dựng các mối quan hệ giữa AI và các lĩnh vực khác giải quyết các bài toán tương tự, và một sự chuyển giao mới của các nhà nghiên cứu cho các phương pháp toán học vững chắc và chuẩn khoa học chính xác.

3. Các lĩnh vực ứng dụng của AI

- Bài toán lập luận, suy diễn

Khái niệm lập luận (reasoning), và suy diễn (reference) được sử dụng rất phổ biến trong lĩnh vực AI. Lập luận là suy diễn logic, dùng để chỉ một tiến trình rút ra kết luận (tri thức mới) từ những giả thiết đã cho (được biểu diễn dưới dạng cơ sở tri thức). Như vậy, để thực hiện lập luận người ta cần có các phương pháp lưu trữ cơ sở tri thức và các thủ tục lập luận trên cơ sở tri thức đó.

- Biểu diễn tri thức

Muốn máy tính có thể lưu trữ và xử lý tri thức thì cần có các phương pháp biểu diễn tri thức. Các phương pháp biểu diễn tri thức ở đây bao gồm các ngôn ngữ biểu diễn và các kỹ thuật xử lý tri thức. Một ngôn ngữ biểu diễn tri thức được đánh giá là "tốt" nếu nó có *tính biểu đạt* cao và các *tính hiệu quả* của thuật toán lập luận trên ngôn ngữ đó. Tính biểu đạt của ngôn ngữ thể hiện khả năng biểu diễn một phạm vi rộng lớn các thông tin trong một miền ứng dụng. Tính hiệu quả của các thuật toán lập luận thể hiện chi phí về thời gian và không gian dành cho việc lập luận. Tuy nhiên, hai yếu tố này dường như đối nghịch nhau, tức là nếu ngôn ngữ có tính biểu đạt cao thì thuật toán lập luận trên đó sẽ có độ phức tạp lớn (tính hiệu quả thấp) và ngược lại (ngôn ngữ đơn giản, có tính biểu đạt thấp thì thuật toán lập luận trên đó sẽ có hiệu quả cao). Do đó, một thách thức lớn trong lĩnh vực AI là xây dựng các ngôn ngữ biểu diễn tri thức mà có thể cân bằng hai yếu tố này, tức là ngôn ngữ có tính biểu đạt đủ tốt (tùy theo từng ứng dụng) và có thể lập luận hiệu quả.

- Lập kế hoạch: khả năng suy ra các mục đích cần đạt được đối với các nhiệm vụ đưa ra, và xác định dãy các hành động cần thực hiện để đạt được mục đích đó.
- Học máy: là một lĩnh vực nghiên cứu của AI đang được phát triển mạnh mẽ và có nhiều ứng dụng trong các lĩnh vực khác nhau như khai phá dữ liệu, khám phá tri thức,…
- Xử lý ngôn ngữ tự nhiên: là một nhánh của AI, tập trung vào các ứng dụng trên ngôn ngữ của con người. Các ứng dụng trong nhận dạng tiếng nói, nhận dạng chữ viết, dịch tự động, tìm kiếm thông tin,…
- Hệ chuyên gia: cung cấp các hệ thống có khả năng suy luận để đưa ra những kết luận. Các hệ chuyên gia có khả năng xử lý lượng thông tin lớn và cung cấp các kết luận dựa trên những thông tin đó. Có rất nhiều hệ chuyên gia nổi tiếng như các hệ chuyên gia y học MYCIN, đoán nhận cấu trúc phân tử từ công thức hóa học DENDRAL, …

- Robotics
- …

4. *Nội dung môn học*

Giáo trình này được viết với các nội dung nhập môn về AI cho các sinh viên chuyên ngành Tin học và Công nghệ thông tin. Các tác giả có tham khảo một số tài liệu, giáo trình của các trường Đại học Quốc gia Hà nội, Đại học Bách khoa Hà nội, … Nội dung gồm các phần sau:

- *Chương 1. Giới thiệu*: trình bày tổng quan về AI, lịch sử ra đời và phát triển và các lính vực ứng dụng của AI.
- *Chương 2. Các phương pháp tìm kiếm lời giải*: trình bày các kỹ thuật tìm kiếm cơ bản được áp dụng để giải quyết các vấn đề và được áp dụng rộng rãi trong các lĩnh vực của trí tuệ nhân tạo.
- *Chương 3. Các giải thuật tìm kiếm lời giải cho trò chơi*: trình bày một số kỹ thuật tìm kiếm trong các trò chơi có đối thủ.
- *Chương 4. Các phương pháp lập luận trên logic mệnh đề*: trình bày cú pháp, ngữ nghĩa của logic mệnh đề và một số thuật toán lập luận trên logic mệnh đề.
- *Chương 5. Các phương pháp lập luận trên logic vị từ cấp một*: trình bày cú pháp, ngữ nghĩa của logic vị từ cấp một và một số thuật toán lập luận cơ bản trên logic vị từ cấp một.
- *Chương 6. Prolog*: Giới thiệu chung về ngôn ngữ Prolog, cú pháp, ngữ nghĩa và cấu trúc chương trình trong Prolog, một số phiên bản mới của Prolog như SWI Prolog,…
- *Chương 7. Lập luận với tri thức không chắc chắn*: Giới thiệu về tri thức không chắc chắn và một số cách tiếp cận biểu diễn và xử lý tri thức không chắc chắn.
- *Chương 8. Học mạng noron nhân tạo*: Giới thiệu về phương pháp và các kỹ thuật cơ bản trong lập luận sử dụng mạng noron nhân tạo.

Chương 2 – Các phương pháp tìm kiếm lời giải

Một lớp các nhiệm vụ mà máy tính (với phần mềm trí tuệ nhân tạo phù hợp) có thể thực hiện tốt hơn con người nhờ vào tốc độ thực hiện của CPU và bộ nhớ dung lượng lớn là ***tìm kiếm lời giải*** trong không gian (hữu hạn hoặc vô hạn) các lời giải tiềm năng của một bài toán cụ thể. Một lời giải tiềm năng có thể là một đường đi trong không gian đồ thị trạng thái của bài toán hoặc là một trạng thái của bài toán. Với tốc độ xử lý nhanh, máy tính chỉ cần sinh ra và duyệt các lời giải tiềm năng (cây tìm kiếm) một cách có hệ thống (tìm kiếm mù, tìm kiếm theo chiều rộng hoặc theo chiều sâu) và kiểm tra nó có là lời giải thực sự (tối ưu) của bài toán đã cho không. Trong nhiều trường hợp, máy tính có thể sử dụng một hàm đánh giá/định hướng (hàm heuristic) để hạn chế không sinh ra các phần của cây tìm kiếm mà khả năng sẽ không chứa lời giải thực sự.

Rất nhiều bài toán từ đơn giản đến phức tạp có thể giải được bằng các phương pháp tìm kiếm đơn giản như trên, như các bài toán chơi cờ, các bài toán tìm đường đi trên đồ thị, các bài toán duyệt các tổ hợp, chỉnh hợp, v.v. Các bài toán có cùng đặc trưng là có thể biểu diễn bởi 4 thành tố sinh ra không gian các lời giải tiềm năng của bài toán: trạng thái đầu, trạng thái đích, các thao tác chuyển trạng thái, chi phí các phép chuyển trạng thái. Với các bài toán này, rõ ràng máy tính có khả năng giải quyết tốt hơn con người.

2.1 Hình thành bài toán

Trong thực tế, nhiều bài toán có thể đưa về bài toán tìm kiếm. Chẳng hạn, muốn tìm nghiệm của phương trình, ta đi tìm những giá trị của biến số trong miền xác định mà khi thay vào phương trình được thỏa mãn; với các bài toán chứng minh, ta đi tìm dãy các lập luận sao cho xuất phát từ giả thiết có thể đi đến kết luận; hoặc muốn đi từ Hà nội đến Sài gòn, người ta phải tra cứu trên bản đồ để tìm những thành phố mà theo đó có thể đi đến Sài gòn nếu xuất phát từ Hà nội.

Bài toán tìm kiếm là xác định trong không gian tìm kiếm (miền) những đối tượng mà thỏa mãn các điều kiện đặt ra.

Trong lĩnh vực AI, chúng ta nghiên cứu hành trình của các agent thông minh. Cơ sở của các bài toán là: trạng thái đầu (trạng thái xuất phát), các hành động biến đổi trạng thái và trạng thái kết thúc (trạng thái đích). Vấn đề đặt ra là xác định dãy các trạng thái hợp lý để sao cho từ trạng thái xuất phát có thể đến được trạng thái đích.

Không gian trạng thái: là tập các trạng thái có thể đạt được bằng cách thực hiện chuỗi các hành động xuất phát từ trạng thái ban đầu. Một hành trình không gian trạng thái là thực hiện dãy các hành động từ trạng thái này đến trạng thái khác.

Giải bài toán : xác định trạng thái xuất phát, tìm dãy các hành động hoặc phép biến đổi (toán tử) các trạng thái sao cho từ trạng thái xuất phát có thể dẫn đến trạng thái đích. Với mỗi bài toán, có thể có một hoặc nhiều cách giải, trong đó, người ta luôn mong muốn tìm lời giải "tốt nhất" dựa vào việc tình chi phí thực hiện.

Hàm chi phí: Giá trị đánh giá chi phí thực hiện biến đổi trạng thái.

Hiệu quả của việc tìm kiếm thể hiện qua việc đánh giá:

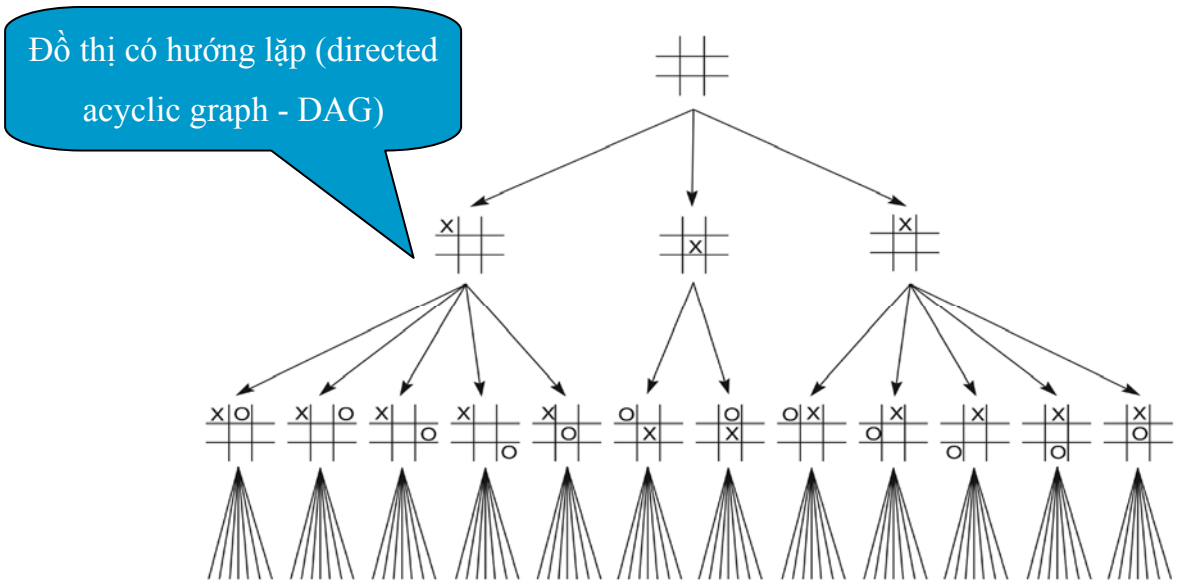
- Việc tìm kiếm có kết thúc không?
- Có tìm thấy lời giải của bài toán không?
- Có tìm được lời giải tối ưu không?

Để thực hiện tìm kiếm, trước hết phải tìm cách biểu diễn bài toán trong *không gian tìm kiếm*. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm tìm kiếm (có thể là không gian liên tục, không gian các đối tượng rời rạc, không gian vecto,…). Không gian tìm kiếm được thể hiện bởi *không gian trạng thái*. Việc biểu diễn bài toán trong không gian trạng thái, cần xác định các yếu tố sau:

- Trạng thái xuất phát
- Tập hợp các toán tử
- Tập hợp các trạng thái kết thúc (trạng thái đích).

Không gian trạng thái có thể được biểu diễn bởi đồ thị có hướng: mỗi đỉnh của đồ thị tương ứng với 1 trạng thái, nếu toán tử R biến đổi từ trạng thái u đến trạng thái v thì có 1 cung gán nhãn R nối hai đỉnh u và v.

Ví dụ:



Hình 2.1: Không gian trạng thái của trò chơi tic-tac-toe

Khi biểu diễn một vấn đề như một đồ thị không gian trạng thái, chúng ta có thể sử dụng lý thuyết đồ thị để phân tích cấu trúc và độ phức tạp của các vấn đề cũng như thực hiện các thủ tục tìm kiếm. Quá trình tìm kiếm là đi xây dựng *cây tìm kiếm* với gốc của cây tương ứng với trạng thái xuất phát, các đỉnh tương ứng với các trạng thái trong đồ thị không gian trạng thái.

Các kỹ thuật tìm kiếm được áp dụng rộng rãi trong lĩnh vực TTNT :

- Tìm kiếm mù : không có hiểu biết gì về các đối tượng để hướng dẫn tìm kiếm

- Tìm kiếm kinh nghiệm (heuristic) : dựa vào kinh nghiệm và hiểu biết về vấn đề cần giải quyết để xây dựng *hàm đánh giá* hướng dẫn sự tìm kiếm.

+ Tìm kiếm tối ưu

+ Tìm kiếm có đối thủ : tìm kiếm nước đi trong các trò chơi hai người (cờ vua, cờ tướng,...)

2.2 Tìm kiếm có hệ thống

Tìm kiếm mù là chiến lược tìm kiếm không có sự hướng dẫn nào cho tìm kiếm, chỉ phát triển các trạng thái từ trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có hai thuật toán tìm kiếm đơn giản:

2.2.1 Tìm kiếm theo chiều rộng:

- Ý tưởng: Bắt đầu mở rộng từ nút gốc, sau đó lần lượt mở rộng các nút được *sinh ra* từ nút gốc, tiếp đến những nút kế tiếp của các nút này, và cứ như vậy cho đến khi tìm thấy một nút đích nào đó hoặc không còn nút nào được sinh ra. Trong đó, nút B gọi là được sinh ra từ nút A nếu B là kề với A trong đồ thị không gian trạng thái. Do đó, tại mỗi bước, trạng thái chọn để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác

- Thuật toán:

Procedure *Breadth_first_Search*;

begin

1. Khởi tạo dsách L chỉ chứa trạng thái ban đầu;

2. **Loop do**

2.1 **if** L rỗng **then** {thông báo tìm kiếm thất bại; stop};

2.2 Loại trạng thái u đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo tìm kiếm thành công; stop};

2.4 **for** mỗi trạng thái v kề u **do**

{Đặt v vào cuối danh sách L;

father(v) ← u};

end;

- Đánh giá thuật toán: Danh sách L được xử lý như hàng đợi

+ Nếu bài toán có nghiệm (tồn tại đường đi từ trạng thái ban đầu tới trạng thái đích) thì thuật toán sẽ tìm ra nghiệm và đường đi là ngắn nhất.

+ Nếu bài toán vô nghiệm, không gian trạng thái hữu hạn, thuật toán dừng và thông báo vô nghiệm.

+ Gọi b là nhân tố nhánh, nghiệm của bài toán là đường đi có độ dài d, độ phức tạp $O(b^d)$.

- Ví dụ:

2.2.2 Tìm kiếm theo chiều sâu

- Ý tưởng: Bắt đầu mở rộng từ nút gốc, sau đó mở rộng một trong các nút ở mức sâu nhất của cây. Nếu tìm đến một điểm cụt (tức là đến nút không phải nút đích và lại không mở rộng được nữa). Như vậy, tại mỗi bước, trạng thái chọn để phát triển là trạng thái được sinh ra sau các trạng thái chờ phát triển khác

- Thuật toán:

Procedure *Deepth_first_Search*;

begin

1. Khởi tạo dsách L chỉ chứa trạng thái ban đầu;

2. **Loop do**

2.1 **if** L rỗng **then** {thông báo tìm kiếm thất bại; stop};

2.2 Loại trạng thái u đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo tìm kiếm thành công; stop};

2.4 **for** mỗi trạng thái v kề u **do**

{Đặt v vào đầu danh sách L;

father(v) ← u};

end;

- Ví dụ:

- Đánh giá thuật toán: Danh sách L được xử lý như ngăn xếp

+ Nếu bài toán có nghiệm, không gian trạng thái hữu hạn thì thuật toán sẽ tìm ra nghiệm. Nếu không gian trạng thái vô hạn thì có thể không tìm ra nghiệm→ không nên dùng thuật toán này với bài toán có cây tìm kiếm chứa các nhánh vô hạn

+ Nghiệm bài toán là đường đi có độ dài d, cây tìm kiếm có nhân tố nhánh b, độ phức tạp trong trường hợp tồi nhất $O(b^d)$, độ phức tạp không gian là O(db).

2.3 Tìm kiếm có sử dụng hàm đánh giá

Tìm kiếm kinh nghiệm (heuristic) là kỹ thuật tìm kiếm dựa vào kinh nghiệm, sự hiểu biết, trực giác để đưa các phỏng đoán, ước chừng trong các bước tìm kiếm.

Lĩnh vực AI giải quyết các bài toán trong hai tình huống cơ bản sau:

- Bài toán được định nghĩa chính xác nhưng *chi phí tìm lời giải bằng tìm kiếm vét cạn là không thể chấp nhận*. Ví dụ: sự bùng nổ không gian tìm kiếm trong trò chơi cờ vua,...

- Lời phát biểu bài toán hay dữ liệu cũng như tri thức sẵn có của bài toán mang *tính mơ hồ*. Ví dụ: chẩn đoán trong y học, các sự cố hỏng hóc của máy móc,…

Việc giải quyết bài toán bằng tìm kiếm heuristic thường thực hiện ba bước chính sau:

- Tìm biểu diễn thích hợp mô tả các trạng thái và các toán tử biến đổi trạng thái của bài toán

- Xây dựng hàm đánh giá *(evaluation function)*, dùng để đánh giá các đặc điểm của một trạng thái trong KGTT
- Thiết kế chiến lược chọn các trạng thái để phát triển ở mỗi bước: *Tìm kiếm tốt nhất-đầu tiên (best-first search) và tìm kiếm leo đồi (hill-climbing).*

Hàm đánh giá: Không có một phương pháp tổng quát để xác định hàm đánh giá, mà tùy vào từng bài toán cụ thể và dựa vào kinh nghiệm người ta có thể đưa ra các đánh giá khác nhau. Dưới đây là một số ví dụ:

Ví dụ*:* trò chơi 8 số (8-puzzle): các con số liên tiếp từ 1 đến 8 được đặt trong một bàn cờ 3x3 (9 ô), như vậy sẽ có 1 ô trống. Người ta di chuyển ô trống này sao cho có thể đưa bàn cờ về trạng thái mà tất cả các con số đều xếp theo thứ tự lien tiếp theo từ ngoài vào trong.

1	5	2
8	3	4
6	7	

a.Trạng thái đầu

1	2	3
8		4
7	6	5

b.Trạng thái đích

Hình 2.2: Ví dụ về trò chơi 8-puzzle

Một số hàm đánh giá có thể được xác định như sau:

- Hàm đánh giá h_1 = số quân cờ nằm sai vị trí. Trong hình 2.2.a) thì h_1= 5

- Hàm đánh giá h_2 = tổng số khoảng cách của các quân cờ so với vị trí mục tiêu (trạng thái đích). Khoảng cách các quân cờ được tính bằng số di chuyển theo chiều ngang hoặc theo chiều dọc của các quân cờ để đến vị trí mục tiêu. Ví dụ trong hình 2.2.a) thì h_2 = 1+2+3+1+1= 8.

2.3.1 Tìm kiếm tốt nhất đầu tiên

- Ý tưởng: tìm kiếm theo chiều rộng kết hợp với hàm đánh giá

- Thuật toán:

Procedure Best-first search;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

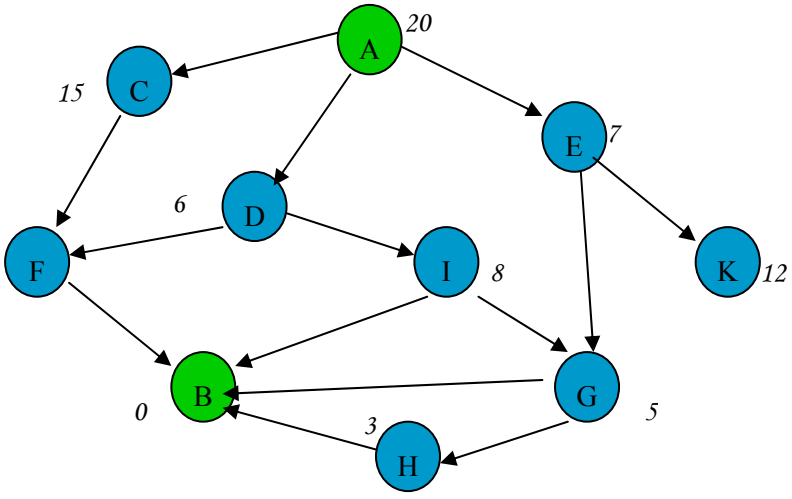
{thông báo thành công; stop};

2.4 **For** mỗi trạng thái v kề u **do**

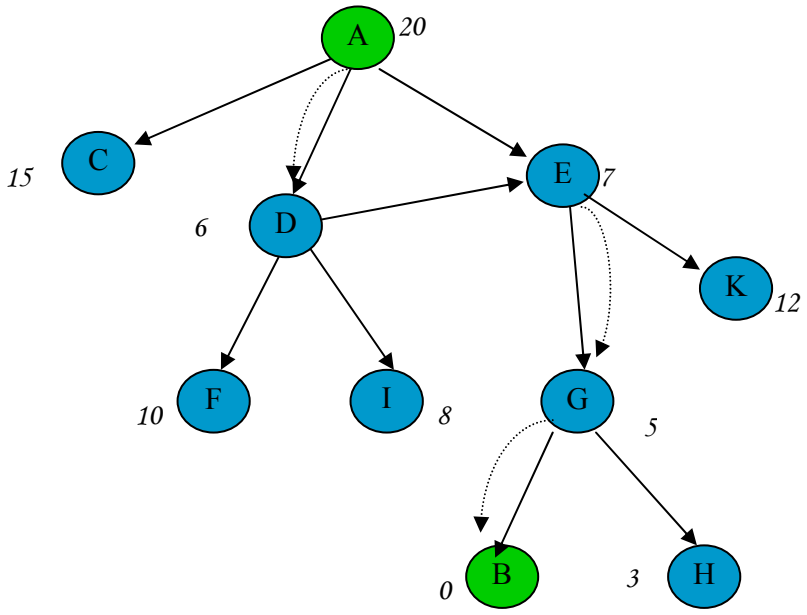
Chèn v vào danh sách L sao cho L được sắp theo thứ tự tăng dần của hàm đánh giá;

End;

- Ví dụ:



Hình 2.2: Đồ thị không gian trạng thái



Hình 2.3: Cây tìm kiếm tốt nhất-đầu tiên

2.3.2 Tìm kiếm leo đồi

- Ý tưởng: Tìm kiếm theo chiều sâu kết hợp với hàm đánh giá. Mở rộng trạng thái hiện tại và đánh giá các trạng thái con của nó bằng hàm đánh giá heuristic. Tại mỗi bước, con "tốt nhất" sẽ được chọn để đi tiếp.

- Thuật toán:

Procedure Hill-Climbing_search;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

{thông báo thành công; stop};

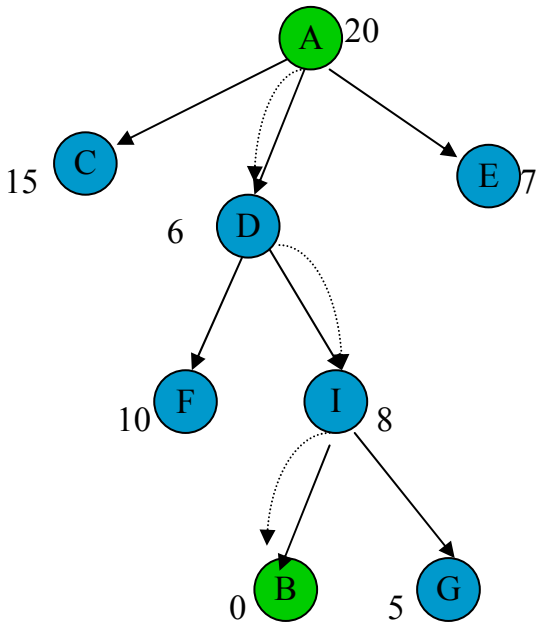
2.4 **For** mỗi trạng thái v kề u **do** đặt v vào L1;

2.5 Sắp xếp L1 theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái tốt nhất ở đầu danh sách L1;

2.6 Chuyển danh sách L1vào đầu danh sách L;

End;

Ví dụ : Với ví dụ đồ thị không gian trạng thái như hình 2.2 thì cây tìm kiếm leo đồi tương ứng như hình 2.4 :



Hình 2.4: Cây tìm kiếm leo đồi

Hạn chế của thuật toán :

- Giải thuật có khuynh hướng bị sa lầy ở những cực đại cục bộ:

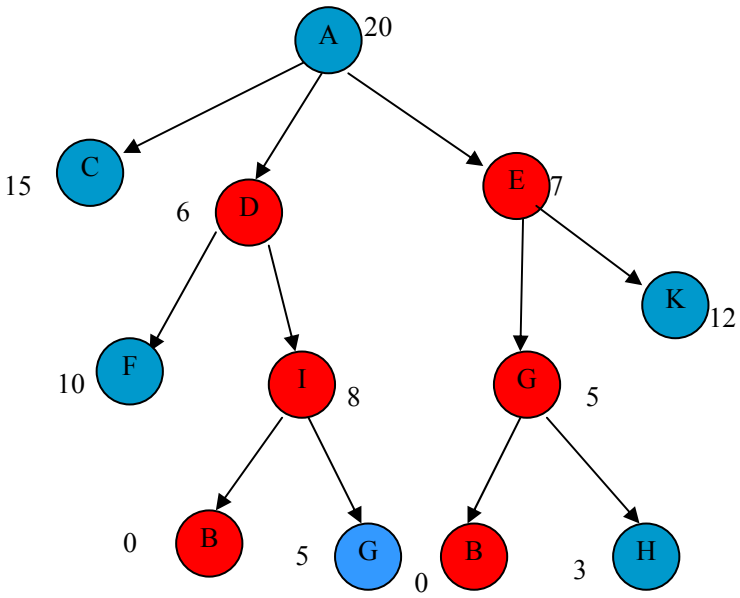
+ Lời giải tìm được không tối ưu

+ Không tìm được lời giải mặc dù có tồn tại lời giải

- Giải thuật có thể gặp vòng lặp vô hạn do không lưu giữ thông tin về các trạng thái đã duyệt.

2.3.3 Tìm kiếm Beam

Để hạn chế không gian tìm kiếm, người ta đưa ra phương pháp tìm kiếm Beam. Đây là phương pháp tìm kiếm theo chiều rộng nhưng có hạn chế số đỉnh phát triển ở mỗi mức. Trong tìm kiếm theo chiều rộng, tại mỗi mức ta phát triển tất cả các đỉnh, còn tìm kiếm Beam thì chọn *k* đỉnh tốt nhất để phát triển. Các đỉnh này được xác định bởi hàm đánh giá. Ví dụ, với đồ thị không gian trạng thái như hình 2.2 và lấy *k*=2 thì cây tìm kiếm Beam như hình 2.5. Các đỉnh được chọn ở mỗi mức là các đỉnh được tô màu đỏ:



Hình 2.5: Cây tìm kiếm Beam

2.4 Tìm kiếm tối ưu

Tìm kiếm tối ưu là tối thiểu chi phí ước lượng để đi đến mục tiêu. Một cách tổng quát, trong không gian tìm kiếm, mỗi đối tượng x được gán một giá trị hàm giá f(x), ta cần tìm đối tượng x mà f(x) là lớn nhất (hoặc nhỏ nhất). Hàm f(x) được gọi là *hàm mục tiêu*. Trong phần này, chúng tôi trình bày một số thuật toán trong bài toán tìm đường

đi ngắn nhất (thuật toán A*, thuật toán nhánh-cận) ; tìm đối tượng tốt nhất và thuật toán di truyền.

2.4.1 Tìm đường đi ngắn nhất

Vấn đề tìm kiếm như đã trình bày trong phần trước là tìm đường đi từ trạng thái xuất phát đến trạng thái đích trong không gian trạng thái. Trong thực tế, thường người ta phải tính đến chi phí di chuyển các trạng thái, từ trạng thái u đến trạng thái v, biểu diễn bởi một số không âm. Giá trị này được gọi là trọng số của cung (u, v) trong đồ thị không gian trạng thái. Trọng số này được xác định tùy thuộc từng bài toán cụ thể. Chẳng hạn, trong bài toán tìm đường đi trên bản đồ, trọng số của cung (u,v) là độ dài của đoạn đường từ địa điểm u đến địa điểm v. Độ dài đường đi được xác định bằng tổng độ dài các cung trên đường đi. Mục tiêu của chúng ta là tìm đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái đích.

Không gian tìm kiếm là tất cả các đường đi từ trạng thái xuất phát đến trạng thái đích, hàm mục tiêu là độ dài đường đi. Người ta có thể giải quyết bài toán bằng các phương pháp tìm kiếm mù (tìm tất cả các đường đi từ trạng thái xuất phát đến trạng thái đích), sau đó so sánh độ dài của chúng và tìm ra đường đi ngắn nhất. Tuy nhiên, trong thực tế không thể áp dụng kỹ thuật này vì với bài toán có không gian tìm kiếm lớn thì đòi hỏi chi phí về thời gian rất cao. Mặt khác, các phương pháp heuristic như tìm kiếm tốt nhất - đầu tiên cho kết quả là đường đi ngắn nhất nhưng có thể kém hiệu quả, hay tìm kiếm leo đồi cũng chỉ cho kết quả là đường đi «tương đối tốt » chứ không đảm bảo đường đi là ngắn nhất. Do đó, để tăng hiệu quả tìm kiếm, chúng ta cần các phương pháp tìm kiếm heuristic sử dụng hàm đánh giá kết hợp :

- Hàm g(u) : đánh giá độ dài đường đi ngắn nhất từ đỉnh xuất phát u_0 đến u. Trong đó, đường đi từ u_0 đến u không phải là trạng thái đích thì được gọi là đường đi một phần, đường đi từ u_0 đến trạng thái đích u gọi là đường đi đầy đủ.

- Hàm h(u) : đánh giá độ dài đường đi ngắn nhất từ đỉnh u đến đích.

Trong đó, hàm h(u) gọi là *chấp nhận được* (đánh giá thấp) nếu với mọi trạng thái *u*, h(u) <= độ dài đường đi ngắn nhất trong thực tế từ *u* đến đích. Ví dụ, trong bài toán tìm đường đi thì h(u) là độ dài đường chim bay từ địa điểm *u* đến đích.

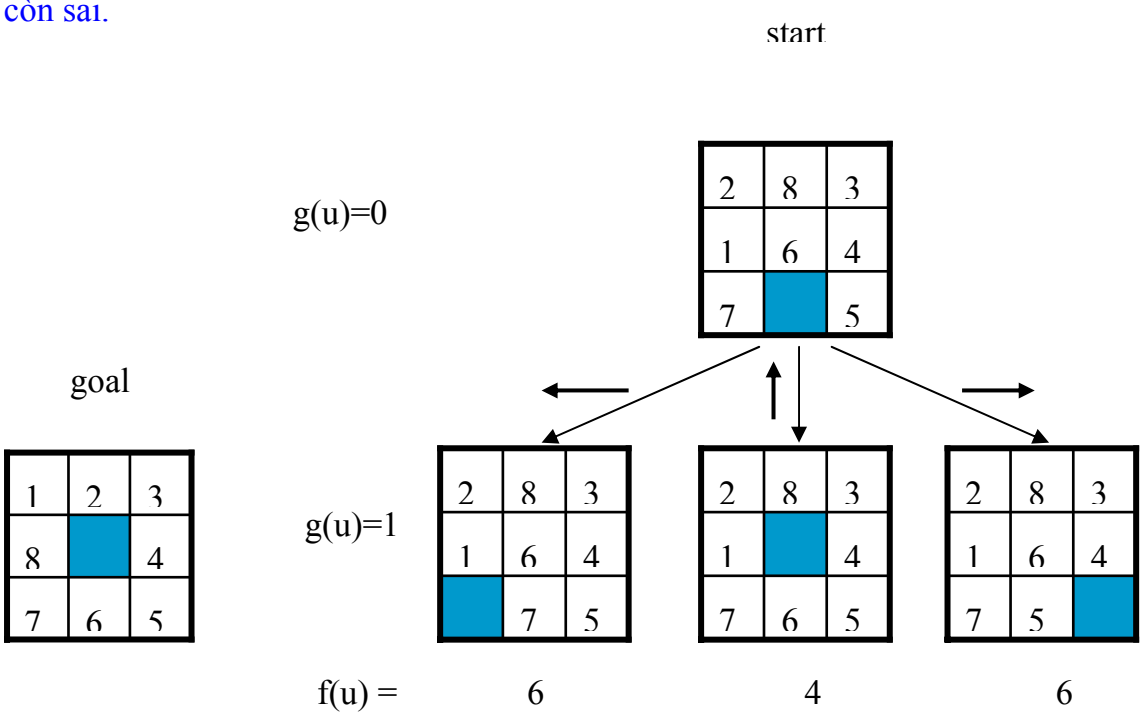
Như vậy, hàm đánh giá là f(u) = g(u) + h(u), đánh giá độ dài đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái đích mà đi qua trạng thái *u*.

Ví dụ về cài đặt hàm đánh giá : Xét trò chơi 8-puzzle.

Cho mỗi trạng thái u một giá trị f(u) = g(u) + h(u)

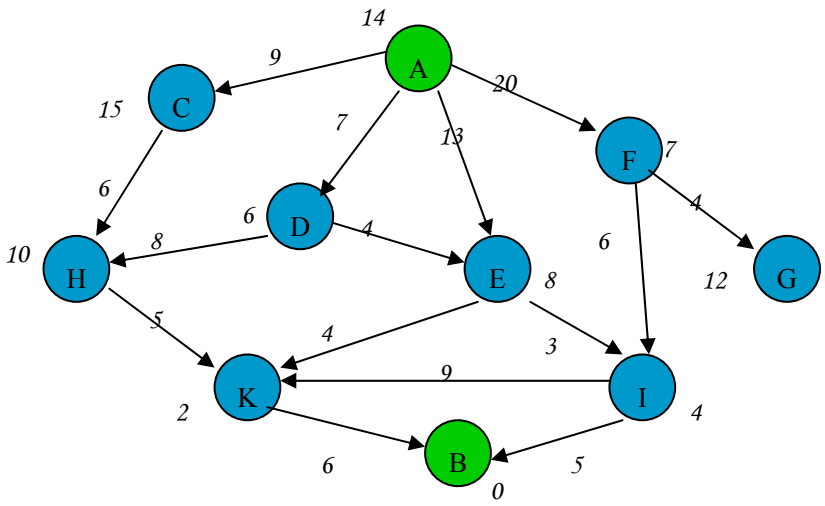
g(u) là khoảng cách thực sự từ u đến trạng thái bắt đầu

h(u) là hàm heuristic đánh giá khoảng cách từ trạng thái u đến mục tiêu, là số vị trí còn sai.



Hình 2.6: Minh họa hàm đánh giá cho trò chơi 8-puzzle

Ví dụ : hình 2.7 minh họa một đồ thị không gian trạng thái với hàm đánh giá: các số ghi cạnh các đỉnh là giá trị của hàm *h*, các số ghi trên các cung là độ dài cung đó.



Hình 2.7: Đồ thị không gian trạng thái với hàm đánh giá

a)- Thuật toán A*

- Ý tưởng : thuật toán tìm kiếm tốt nhất – đầu tiên với hàm đánh giá f(u)

- Thuật toán :

Procedure A*;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

{thông báo thành công; stop};

2.4 **For** mỗi trạng thái v kề u **do**

{g(v)←g(u)+k(u,v)

f(v)←g(v)+h(v);

đặt v vào danh sách L;}

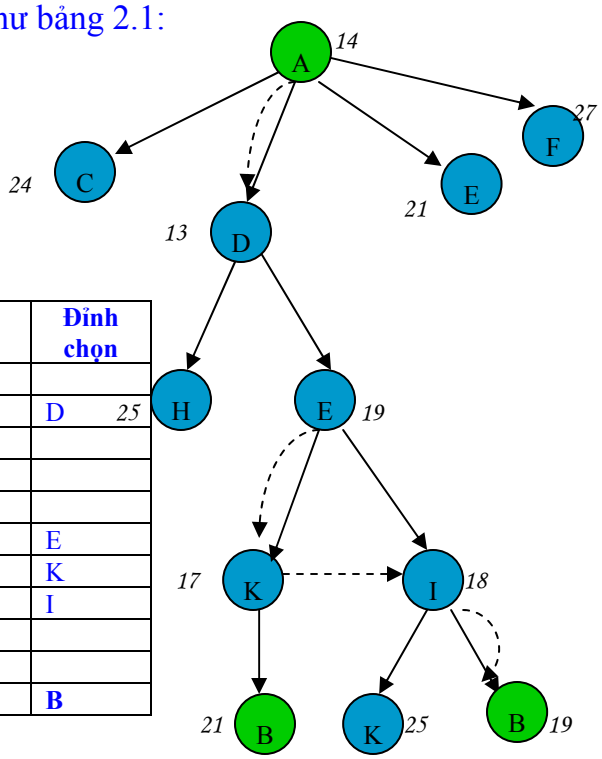
2.5 Sắp xếp L theo thứ tự tăng dần của hàm f;

End;

- Ví dụ : Với đồ thị không gian trạng thái như hình 2.7, đỉnh xuất phát A và đỉnh đích B. Áp dụng thuật toán A*, ta xây dựng được cây tìm kiếm như hình 2.8 và giá trị của hàm *f* tại các đỉnh được tính như bảng 2.1:

Đỉnh phát triển (u)	Đỉnh con (v)	g(v)	f(v)	Đỉnh chọn
A	C	9	9+15=24	
	D	7	7+6=13	D
	E	13	13+8=21	
	F	20	20+7=27	
D	H	7+8=15	15+10=25	
	E	7+4=11	11+8=19	E
E	K	11+4=15	15+2=17	K
	I	11+3=14	14+4=18	I
K	B	15+6=21	21+0=21	
I	K	14+9=23	23+2=25	
	B	14+5=19	19+0=19	**B**

*Bảng 2.1: Tính giá trị hàm f cho thuật toán A**



*Hình 2.8: Cây tìm kiếm A**

- Nhận xét :

+ Nếu h(u) là đánh giá thấp (đặc biệt h(u)=0 với mọi trạng thái *u*), thì A* là thuật toán tối ưu, tức là nghiệm tìm được là tối ưu. Nếu độ dài các cung không nhỏ hơn một số dương δ nào đó thì A* là thuật toán đầy đủ, tức là nó luôn dừng và tìm ra nghiệm.

+ Trường hợp hàm đánh giá h(u)=0 với mọi *u*, thuật toán A* chính là thuật toán tìm kiếm tốt nhất – đầu tiên với hàm đánh giá g(u).

+ Thuật toán A* đã được chứng minh là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho bài toán tìm đường đi ngắn nhất.

b)- Thuật toán nhánh - cận

- Ý tưởng : thuật toán tìm kiếm leo đồi kết hợp với hàm đánh giá *f(u)*. Tại mỗi bước, khi phát triển trạng thái *u*, chọn trạng thái con v tốt nhất (*f(v)* nhỏ nhất) của *u* để phát triển ở bước sau. Quá trình tiếp tục như vậy cho đến khi gặp trạng thái *w* là đích, hoặc *w* không có đỉnh kề, hoặc *w* có *f(w)* lớn hơn độ dài đường đi tối ưu tạm thời (đường đi đầy đủ ngắn nhất trong số những đường đi đầy đủ đã tìm được). Trong các trường hợp này, chúng ta không phát triển đỉnh *w* nữa, tức là cắt bỏ những nhánh xuất phát từ *w*, và quay lên cha của *w* để tiếp tục đi xuống trạng thái tốt nhất trong số những trạng thái còn lại chưa được phát triển.

- Thuật toán :

Procedure Branch-and-Bound;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

Gán giá trị ban đầu cho cost; */*cost là giá trị đường đi tối ưu tạm thời*/*

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

if g(u)<=cost then {cost ←g(u); quay lại 2.1};

2.4 **if** f(u)>cost **then** quay lại 2.1;

2.5 **For** mỗi trạng thái v kề u **do**

{g(v) ←g(u)+k(u,v);

f(v) ←g(v) +h(v);

đặt v vào danh sách L1};

2.6 Sắp xếp L1 theo thứ tự tăng dần của hàm f;

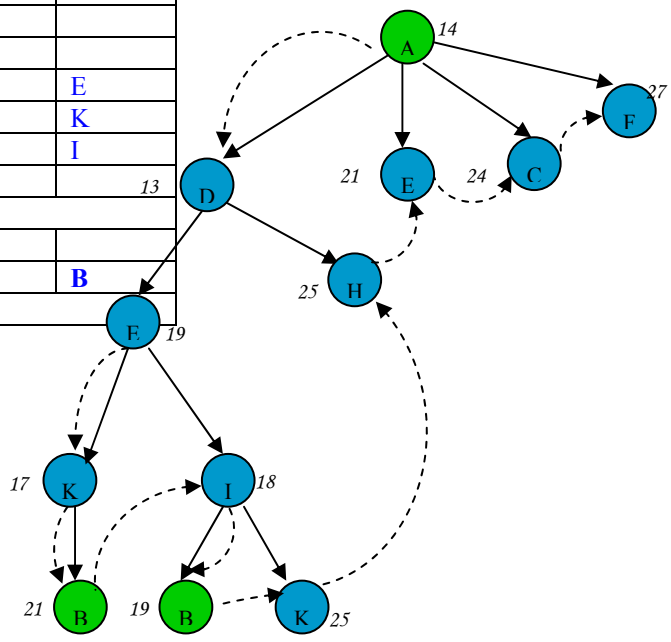
2.7 Chuyển danh sách L1vào đầu danh sách L sao cho L1 ở đầu danh sách L;

End;

- Ví dụ : Với đồ thị không gian trạng thái như hình 2.7, đỉnh xuất phát A và đỉnh đích B. Áp dụng thuật toán nhánh – cận, ta xây dựng được cây tìm kiếm như hình 2.9 và giá trị của hàm *f* tại các đỉnh được tính như bảng 2.2:

Đỉnh phát triển (u)	Đỉnh con (v)	g(v)	f(v)	Đỉnh chọn
A	C	9	9+15=24	
	D	7	7+6=13	D
	E	13	13+8=21	
	F	20	20+7=27	
D	H	7+8=15	15+10=25	
	E	7+4=11	11+8=19	E
E	K	11+4=15	15+2=17	K
	I	11+3=14	14+4=18	I
K	B	15+6=21	21+0=21	
B	**cost := 21**			
I	K	14+9=23	23+2=25	
	B	14+5=19	19+0=19	**B**
B	**cost := 19**			

Bảng 2.2: Tính giá trị hàm f cho thuật toán nhánh-cận



Hình 2.9 : Cây tìm kiếm nhánh-cận

- Nhận xét : Thuật toán nhánh-cận cũng là thuật toán đầy đủ và tối ưu nếu h(u) là hàm đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó

2.4.2 Tìm đối tượng tốt nhất

Chương 3 – Các giải thuật tìm kiếm lời giải cho trò chơi

Chương trình chơi cờ đầu tiên được viết bởi Claude Shannon vào năm 1950 đã là một minh chứng cho khả năng máy tính có thể làm được những việc đòi hỏi trí thông minh của con người. Từ đó người ta nghiên cứu các chiến lược chơi cho máy tình với các trò chơi có đối thủ (có hai người tham gia). Việc giải quyết bài toán này có thể đưa về bài toán tìm kiếm trong không gian trạng thái, tức là tìm một chiến lược chọn các nước đi hợp lệ cho máy tính. Tuy nhiên, vấn đề tìm kiếm ở đây phức tạp hơn so với vấn đề tìm kiếm trong chương trước, vì người chơi không biết trước đối thủ sẽ chọn nước đi nào tiếp theo. Chương này sẽ trình bày một số chiến lược tìm kiếm phổ biến như Minimax, phương pháp cắt cụt α-β.

3.1 Cây trò chơi đầy đủ

Các trò chơi có đối thủ có các đặc điểm: hai người thay phiên nhau đưa ra các nước đi tuân theo các luật của trò chơi (các nước đi hợp lệ), các luật này là như nhau đối với cả hai người chơi, chẳng hạn các trò chơi cờ: cờ vua, cờ tướng, cờ ca rô (tic-tăc-toe), …. Ví dụ, trong chơi cờ vua, một người điều khiển quân Trắng và một người điều khiển quân Đen. Người chơi có thể lựa chọn các nước đi theo các luật với các quân tốt, xe, mã,… Luật đi quân tốt Trắng, xe Trắng, mã Trắng,… giống luật đi quân tốt Đen, xe Đen, mã Đen,…Hơn nữa, cả hai người chơi đều biết đầy đủ các thông tin về tình thế cuộc chơi. Thực hiện trò chơi là người chơi tìm kiếm nước đi *tốt nhất* trong số rất nhiều nước đi hợp lệ, tại mỗi lượt chơi của mình, sao cho sau một dãy nước đi đã thực hiện người chơi phải thắng cuộc.

Vấn đề chơi cờ có thể được biểu diễn trong không gian trạng thái, ở đó, mỗi trạng thái là một tình thế của cuộc chơi (sự sắp xếp các quân cờ trên bàn cờ):

- Trạng thái xuất phát là sự sắp xếp các quân cờ của hai bên khi bắt đầu cuộc chơi (chưa ai đưa ra nước đi)
- Các toán tử biến đổi trạng thái là các nước đi hợp lệ

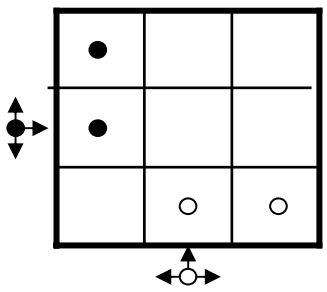
- Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường được xác định bởi một số điều kiện dừng (chẳng hạn, quân Trắng thắng hoặc quân Đen thắng hoặc hai bên hòa nhau)
- Hàm kết cuộc: mang giá trị tương ứng với mỗi trạng thái kết thúc. Chẳng hạn, trong cờ vua, hàm kết cuộc có giá trị là 1 tại các trạng thái mà Trắng thắng, -1 tại các trạng thái mà Trắng thua và 0 tại các trạng thái hai bên hòa nhau. Trong các trò chơi tính điểm khác thì hàm kết cuộc có thể nhận các giá trị nguyên trong đoạn [-m, m], với m là một số nguyên dương nào đó.

Như vậy, trong các trò chơi có đối thủ, người chơi (điều khiển quân Trắng – gọi tắt là Trắng) luôn tìm một dãy các nước đi xen kẽ với các nước đi của đối thủ (điều khiển quân Đen – gọi tắt là Đen) để tạo thành một đường đi từ trạng thái ban đầu đến trạng thái kết thúc là thắng cho Trắng.

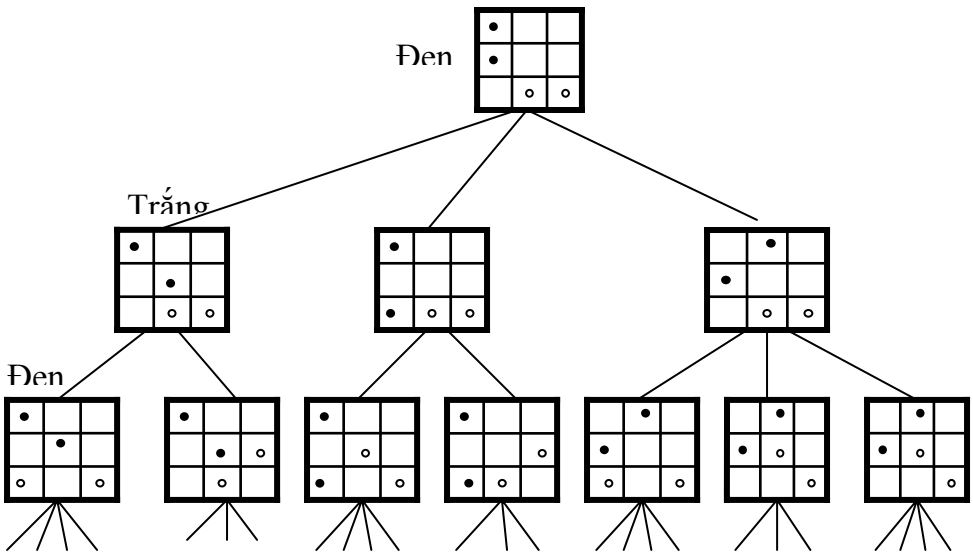
Không gian tìm kiếm đối với các trò chơi này có thể được biểu diễn bởi *cây trò chơi* như sau: gốc của cây ứng với trạng thái xuất phát, các đỉnh trên cây tương ứng với các trạng thái của bàn cờ, các cung (u, v) nếu có biến đổi từ trạng thái u đến trạng thái v. Các đỉnh trên cây được gán nhãn là đỉnh Trắng (Đen) ứng với trạng thái mà quân Trắng (Đen) đưa ra nước đi. Nếu một đỉnh u được gán nhãn là Trắng (Đen) thì các đỉnh con v của nó là tất cả các trạng thái nhận được từ u do Trắng (Đen) thực hiện một nước đi hợp lệ nào đó. Do đó, các đỉnh trên cùng một mức của cây đều có nhãn là Trắng hoặc đều có nhãn là Đen, các lá của cây ứng với trạng thái kết thúc.

Ví dụ: trò chơi Dodgem:

Có hai quân Trắng và hai quân Đen được xếp vào bàn cờ 3x3. Ban đầu các quân cờ được xếp như hình bên. Quân Đen có thể đi đến ô trống bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi đến ô trống bên trên, bên trái hoặc bên phải. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa được cả hai quân của mình ra khỏi bàn cờ hoặc tạo ra tình thế mà đối phương không đi được là thắng cuộc.



Trò chơi Dodgem



Cây trò chơi Dodgem với Đen đi trước

3.2 Giải thuật Minimax

Quá trình chơi cờ là quá trình mà Trắng và Đen thay phiên nhau đưa ra các nước đi hợp lệ cho đến khi dẫn đến trạng thái kết thúc cuộc chơi. Quá trình này biểu diễn bởi đường đi từ nút gốc tới nút lá trên cây trò chơi. Giả sử tại một đỉnh u nào đó trên đường đi, nếu u là đỉnh Trắng (Đen) thì cần chọn một nước đi nào đó đến một trong các đỉnh con Đen (Trắng) v của u. Tại đỉnh Đen (Trắng) v sẽ chọn đi tiếp đến một đỉnh con Trắng (Đen) w của v. Quá trình này tiếp tục cho đến khi đạt đến một đỉnh lá của cây.

Chiến lược tìm nước đi của Trắng hay Đen là luôn tìm những nước đi dẫn tới trạng thái tốt nhất cho mình và tồi nhất cho đối thủ. Giả sử Trắng cần tìm nước đi tại đỉnh u, nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con v sao cho v là tốt nhất trong số các đỉnh con của u. Đến lượt Đen chọn nước đi từ v, Đen cũng chọn nước đi tốt nhất cho mình. Để chọn nước đi tối ưu cho Trắng tại đỉnh u, cần xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá ứng với giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất (các đỉnh lá)

lên gốc u. Giả sử cần xác định giá trị của đỉnh v mà các đỉnh con của nó đã xác định. Khi đó, nếu v là đỉnh Trắng thì giá trị của nó là giá trị lớn nhất trong các đỉnh con, nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các đỉnh con.

Sau đây là thủ tục chọn nước đi cho Trắng tại đỉnh u Minimax(u, v), trong đó v là đỉnh con được chọn của u:

```
Procedure Minimax(u, v);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w)  then
            {val ← MinVal(w);  v ←  w}
end;
Function MinVal(u); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc  then  MinVal(u) ← f(u)
    else MinVal(u) ← min{MaxVal(v) | v là đỉnh con của u}
end;
Function MaxVal(u); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc  then  MaxVal(u) ← f(u)
    else MaxVal(u) ← max{MinVal(v) | v là đỉnh con của u}
end;
```

Trong các thủ tục và hàm trên, f(u) là giá trị của hàm kết cuộc tại đỉnh kết thúc u.

Thuật toán Minimax là thuật toán tìm kiếm theo chiều sâu. Về lý thuyết, chiến lược Minimax cho phép tìm nước đi tối ưu cho Trắng. Tuy nhiên trong thực tế, ta không có đủ thời gian để tính toán nước đi tối ưu này. Bởi vì thuật toán tính toán trên toàn bộ cây trò chơi (xem xét tất cả các đỉnh của cây theo kiểu vét cạn). Trong các trò chơi hay thì kích thước của cây trò chơi là cực lớn. Chẳng hạn, trong cờ vua, chỉ tính đến độ sâu 40 thì cây trò chơi đã có đến 10^{120} đỉnh. Nếu cây có độ cao m và tại mỗi đỉnh có b nước đi thì độ phức tạp về thời gian của thuật toán Minimax là $O(b^m)$.

Trong thực tế, các trò chơi đều có giới hạn về thời gian. Do đó, để có thể tìm nhanh nước đi tốt (không phải tối ưu) thay vì sử dụng hàm kết cuộc và xét tất cả các đỉnh của cây trò chơi, ta sử dụng hàm đánh giá và chỉ xem xét một bộ phận của cây trò chơi.

3.3 Giải thuật Minimax với độ sâu hạn chế

3.3.1 Hàm đánh giá

Hàm đánh giá eval cho mỗi đỉnh u là đánh giá "mức độ lợi thế" của trạng thái u. Giá trị của eval(u) là số dương càng lớn thì trạng thái u càng có lợi cho Trắng, giá trị của eval(u) là số dương càng nhỏ thì trạng thái u càng có lợi cho Đen, eval(u)=0 thì trạng thái u không có lợi cho đối thủ nào, eval(u)=+∞ thì u là trạng thái thắng cuộc cho Trắng, eval(u)=-∞ thì u là trạng thái thắng cuộc cho Đen.

Hàm đánh giá đóng vai trò rất quan trọng trong các trò chơi, nếu hàm đánh giá tốt sẽ định hướng chính xác việc lựa chọn các nước đi tốt. Việc thiết kế hàm đánh giá phụ thuộc vào nhiều yếu tố: các quân cờ còn lại của hai bên, sự bố trí các quân cờ này,… Để đưa ra hàm đánh giá chính xác đòi hỏi nhiều thời gian tính toán, tuy nhiên, trong thực tế người chơi bị giới hạn thời gian đưa ra nước đi. Vì vậy, việc đưa ra hàm đánh giá phụ thuộc vào kinh nghiệm của người chơi. Sau đây là một số ví dụ về cách xây dựng hàm đánh giá:

Ví dụ 1: Hàm đánh giá cho cờ vua. Mỗi loại quân được gán một giá trị số phù hợp với "sức mạnh" của nó. Chẳng hạn, quân tốt Trắng (Đen) được gán giá trị 1 (-1), mã hoặc

tượng Trắng (Đen) được gán giá trị 3 (-3), xe Trắng (Đen) được gán giá trị 5 (-5) và hậu Trắng (Đen) được gán giá trị 9 (-9). Hàm đánh giá của một trạng thái được tính bằng cách lấy tổng giá trị của tất cả các quân cờ trong trạng thái đó. Hàm đánh giá này được gọi là hàm tuyến tính có trọng số, vì có thể biểu diễn dưới dạng:

$$s_1w_1 + s_2w_2 + \ldots + s_nw_n$$

Trong đó, w_i là giá trị của quân cờ loại i, s_i là số quân loại đó.

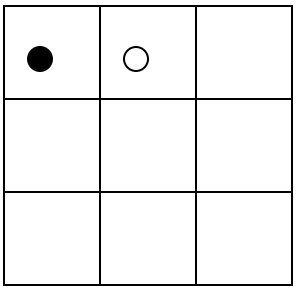
Đây là cách đánh giá đơn giản, vì nó không tính đến sự bố trí của các quân cờ, các mối tương quan giữa chúng.

Ví dụ 2: Hàm đánh giá trạng thái trong trò chơi Dodgem. Mỗi quân Trắng được gán giá trị tương ứng với các vị trí trên bàn cờ như trong hình bên trái. Mỗi quân Đen được gán giá trị ở các vị trí tương ứng nhu hình bên phải:

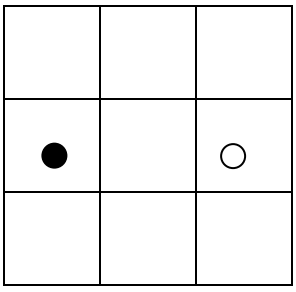
30	35	40
15	20	25
0	5	10

-10	-25	-40
-5	-20	-35
0	-15	-30

Ngoài ra, nếu quân Trắng cản trực tiếp một quân Đen, nó được thêm 40 điểm, nếu cản gián tiếp được thêm 30 điểm (xem hình dưới). Tương tự, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, cản gián tiếp được thêm -30 điểm.

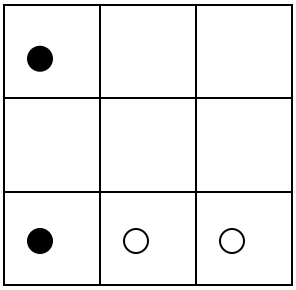


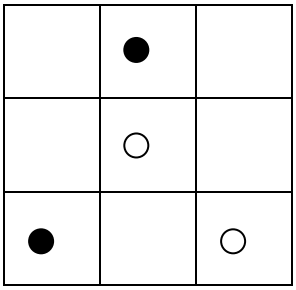
Trắng cản trực tiếp Đen được thêm 40 điểm



Trắng cản gián tiếp Đen được thêm 30 điểm

Áp dụng cách tính hàm đánh giá nêu trên, ta tính được giá trị của các trạng thái ở các hình dưới như sau:





Giá trị hàm đánh giá:75=
(-10+0+5+10)+(40+30)

Giá trị hàm đánh giá:-5=
(-25+0+20+10)+(-40+30)

3.3.2 Thuật toán

Để hạn chế không gian tìm kiếm, khi xác định nước đi cho Trắng tại u, ta chỉ xem xét cây gốc u tại độ cao h nào đó. Áp dụng thủ tục Minimax cho cây trò chơi gốc u, độ cao h và sử dụng hàm đánh giá để xác định giá trị cho các lá của cây.

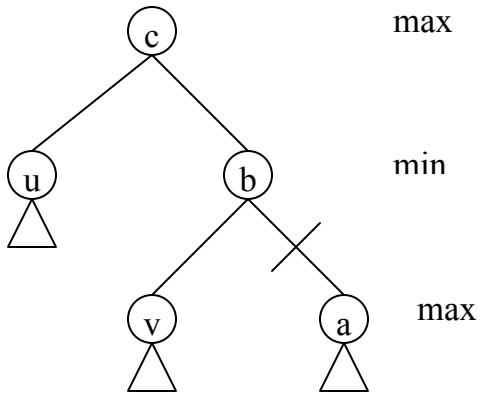
```
Procedure Minimax(u, v, h);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w, h-1)  then
            {val ← MinVal(w, h-1);  v ←  w}
end;
Function MinVal(u, h); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc or h = 0 then  MinVal(u, h) ← eval(u)
    else MinVal(u, h) ← min{MaxVal(v, h-1) | v là đỉnh con của u}
```

```
end;
Function MaxVal(u, h); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc or h =0  then  MaxVal(u, h) ← eval(u)
    else MaxVal(u, h) ← max{MinVal(v, h-1) | v là đỉnh con của u}
end;
```

3.4 Giải thuật Minimax với cắt tỉa alpha-beta

Trong chiến lược Minimax với độ sâu hạn chế thì số đỉnh của cây trò chơi phải xét vẫn còn rất lớn với h>=3. Khi đánh giá đỉnh u tới độ sâu h, thuật toán Minimax đòi hỏi phải đánh giá tất cả các đỉnh của cây gốc u với độ sâu h. Tuy nhiên, phương pháp cắt cụt alpha-beta cho phép cắt bỏ những nhánh không cần thiết cho việc đánh giá đỉnh u. Phương pháp này làm giảm bớt số đỉnh phải xét mà không ảnh hưởng đến kết quả đánh giá đỉnh u.

Ý tưởng: Giả sử tại thời điểm hiện tại đang ở đỉnh Trắng a, đỉnh a có anh em là v đã được đánh giá. Giả sử cha của đỉnh a là b, b có anh em là u đã được đánh giá, và cha của b là c như hình sau:



Cắt bỏ cây con gốc a nếu eval(u)>eval(v)

Khi đó ta có giá trị đỉnh Trắng *c* ít nhất là giá trị của *u*, giá trị của đỉnh Đen *b* nhiều nhất là giá trị của *v*. Do đó, nếu eval(u) > eval(v) ta không cần đi xuống để đánh giá đỉnh *a* nữa mà vẫn không ảnh hưởng đến đánh giá đỉnh *c*. Hay nói cách khác, ta có thể cắt bỏ cây con gốc *a*.

Lập luận tương tự cho trường hợp *a* là đỉnh Đen, trường hợp này nếu eval(u)<eval(v) ta cũng cắt bỏ cây con gốc *a*.

Để cài đặt kỹ thuật này, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Trắng, tham số β để ghi lại giá trị nhỏ nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Đen.

Thuật toán:

```
Procedure   Alpha_beta(u, v);
begin
    α←-∞; β←-∞;
    for  mỗi w là đỉnh con của u  do
    if  α <= MinVal(w, α, β)  then
            {α ← MinVal(w, α, β);  v ←  w}
end;
Function MinVal(u, α, β); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc or u là lá của cây hạn chế then
        MinVal(u, α, β) ← eval(u)
    else for  mỗi đỉnh v là con của u  do
            {β ← min{β, MaxVal(v, α, β)} ;
```

```
            If α >= β  then  exit};
        /*cắt bỏ các cây con từ các đỉnh v còn lại */
        MinVal(u, α, β) ← β;
    end;
    Function MaxVal(u, α, β); { hàm xác định giá trị cho các đỉnh Trắng}
    begin
        if  u là đỉnh kết thúc or là lá của cây hạn chế  then
            MaxVal(u, α, β) ← eval(u)
        Else for mỗi đỉnh v là con của u  do
            α ← max{α, MinVal(v, α, β)} ;
            If α >= β then exit};
        /*cắt bỏ các cây con từ các đỉnh v còn lại */
        MaxVal(u, α, β) ← α
end;
```

Đinh Mạnh Tường

Giáo trình

Trí tuệ Nhân tạo

Khoa CNTT - Đại Học Quốc Gia Hà Nội

Phần I
Giải quyết vấn đề bằng tìm kiếm

Vấn đề tìm kiếm, một cách tổng quát, có thể hiểu là tìm một đối tượng thỏa mãn một số đòi hỏi nào đó, trong một tập hợp rộng lớn các đối tượng. Chúng ta có thể kể ra rất nhiều vấn đề mà việc giải quyết nó được quy về vấn đề tìm kiếm.

Các trò chơi, chẳng hạn cờ vua, cờ carô có thể xem như vấn đề tìm kiếm. Trong số rất nhiều nước đi được phép thực hiện, ta phải tìm ra các nước đi dẫn tới tình thế kết cuộc mà ta là người thắng.

Chứng minh định lý cũng có thể xem như vấn đề tìm kiếm. Cho một tập các tiên đề và các luật suy diễn, trong trường hợp này mục tiêu của ta là tìm ra một chứng minh (một dãy các luật suy diễn được áp dụng) để được đưa đến công thức mà ta cần chứng minh.

Trong các lĩnh vực nghiên cứu của **Trí Tuệ Nhân Tạo,** chúng ta thường xuyên phải đối đầu với vấn đề tìm kiếm. Đặc biệt trong lập kế hoạch và học máy, tìm kiếm đóng vai trò quan trọng.

Trong phần này chúng ta sẽ nghiên cứu các kỹ thuật tìm kiếm cơ bản được áp dụng để giải quyết các vấn đề và được áp dụng rộng rãi trong các lĩnh vực nghiên cứu khác của **Trí Tuệ Nhân Tạo**. Chúng ta lần lượt nghiên cứu các kỹ thuật sau:

- Các kỹ thuật tìm kiếm mù, trong đó chúng ta không có hiểu biết gì về các đối tượng để hướng dẫn tìm kiếm mà chỉ đơn thuần là xem xét theo một hệ thống nào đó tất cả các đối tượng để phát hiện ra đối tượng cần tìm.
- Các kỹ thuật tìm kiếm kinh nghiệm (tìm kiếm heuristic) trong đó chúng ta dựa vào kinh nghiệm và sự hiểu biết của chúng ta về vấn đề cần giải quyết để xây dựng nên hàm đánh giá hướng dẫn sự tìm kiếm.
- Các kỹ thuật tìm kiếm tối ưu.
- Các phương pháp tìm kiếm có đối thủ, tức là các chiến lược tìm kiếm nước đi trong các trò chơi hai người, chẳng hạn cờ vua, cờ tướng, cờ carô.

Chương I
Các chiến lược tìm kiếm mù

Trong chương này, chúng tôi sẽ nghiên cứu các chiến lược tìm kiếm mù (blind search): tìm kiếm theo bề rộng (breadth-first search) và tìm kiếm theo độ sâu (depth-first search). Hiệu quả của các phương pháp tìm kiếm này cũng sẽ được đánh giá.

1.1 Biểu diễn vấn đề trong không gian trạng thái

Một khi chúng ta muốn giải quyết một vấn đề nào đó bằng tìm kiếm, đầu tiên ta phải xác định không gian tìm kiếm. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm tìm kiếm. Nó có thể là không gian liên tục, chẳng hạn không gian các véctơ thực n chiều; nó cũng có thể là không gian các đối tượng rời rạc.

Trong mục này ta sẽ xét việc biểu diễn một vấn đề trong không gian trạng thái sao cho việc giải quyết vấn đề được quy về việc tìm kiếm trong không gian trạng thái.

Một phạm vi rộng lớn các vấn đề, đặc biệt các câu đố, các trò chơi, có thể mô tả bằng cách sử dụng khái niệm trạng thái và toán tử (phép biến đổi trạng thái). Chẳng hạn, một khách du lịch có trong tay bản đồ mạng lưới giao thông nối các thành phố trong một vùng lãnh thổ (hình 1.1), du khách đang ở thành phố A và anh ta muốn tìm đường đi tới thăm thành phố B. Trong bài toán này, các thành phố có trong các bản đồ là các trạng thái, thành phố A là trạng thái ban đầu, B là trạng thái kết thúc. Khi đang ở một thành phố, chẳng hạn ở thành phố D anh ta có thể đi theo các con đường để nối tới các thành phố C, F và G. Các con đường nối các thành phố sẽ được biểu diễn bởi các toán tử. Một toán tử biến đổi một trạng thái thành một trạng thái khác. Chẳng hạn, ở trạng thái D sẽ có ba toán tử dẫn trạng thái D tới các trạng thái C, F và G. Vấn đề của du khách bây giờ sẽ là tìm một dãy toán tử để đưa trạng thái ban đầu A tới trạng thái kết thúc B.

Một ví dụ khác, trong trò chơi cờ vua, mỗi cách bố trí các quân trên bàn cờ là một trạng thái. Trạng thái ban đầu là sự sắp xếp các quân lúc bắt đầu cuộc chơi. Mỗi nước đi hợp lệ là một toán tử, nó biến đổi một cảnh huống trên bàn cờ thành một cảnh huống khác.

Như vậy muốn biểu diễn một vấn đề trong không gian trạng thái, ta cần xác định các yếu tố sau:

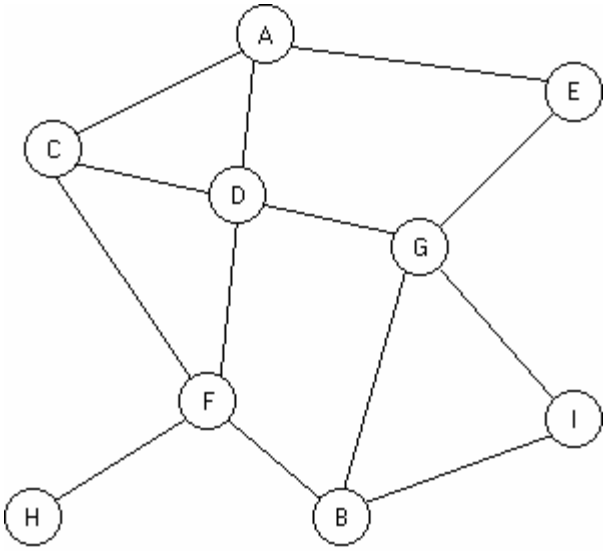
- Trạng thái ban đầu.
- Một tập hợp các toán tử. Trong đó mỗi toán tử mô tả một hành động hoặc một phép biến đổi có thể đưa một trạng thái tới một trạng thái khác.

Tập hợp tất cả các trạng thái có thể đạt tới từ trạng thái ban đầu bằng cách áp dụng một dãy toán tử, lập thành không gian trạng thái của vấn đề.

Ta sẽ ký hiệu không gian trạng thái là U, trạng thái ban đầu là u_0 ($u_0 \in U$). Mỗi toán tử R có thể xem như một ánh xạ R: U→U. Nói chung R là một ánh xạ không xác định khắp nơi trên U.

 Một tập hợp T các trạng thái kết thúc (trạng thái đích). T là tập con của không gian U. Trong vấn đề của du khách trên, chỉ có một trạng thái đích, đó là thành phố B. Nhưng trong nhiều vấn đề (chẳng hạn các loại cờ) có thể có nhiều trạng thái đích và ta không thể xác định trước được các trạng thái đích. Nói chung trong phần lớn các vấn đề hay, ta chỉ có thể mô tả các trạng thái đích là các trạng thái thỏa mãn một số điều kiện nào đó.

Khi chúng ta biểu diễn một vấn đề thông qua các trạng thái và các toán tử, thì việc tìm nghiệm của bài toán được quy về việc tìm đường đi từ trạng thái ban đầu tới trạng thái đích. (Một đường đi trong không gian trạng thái là một dãy toán tử dẫn một trạng thái tới một trạng thái khác).

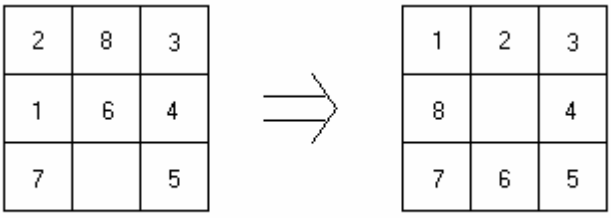


Hình 1.1 Tìm đường đi từ A đến B

Chúng ta có thể biểu diễn không gian trạng thái bằng đồ thị định hướng, trong đó mỗi đỉnh của đồ thị tương ứng với một trạng thái. Nếu có toán tử R biến đổi trạng thái u thành trạng thái v, thì có cung gán nhãn R đi từ đỉnh u tới đỉnh v. Khi đó một đường đi trong không gian trạng thái sẽ là một đường đi trong đồ thị này.

Sau đây chúng ta sẽ xét một số ví dụ về các không gian trạng thái được xây dựng cho một số vấn đề.

Ví dụ 1: Bài toán 8 số. Chúng ta có bảng 3x3 ô và tám quân mang số hiệu từ 1 đến 8 được xếp vào tám ô, còn lại một ô trống, chẳng hạn như trong hình 2 bên



Hình 1.2 Trạng thái ban đầu và trạng thái kết thúc của bài toán số.

trái. Trong trò chơi này, bạn có thể chuyển dịch các quân ở cạch ô trống tới ô trống đó. Vấn đề của bạn là tìm ra một dãy các chuyển dịch để biến đổi cảnh huống ban đầu (hình 1.2 bên trái) thành một cảnh huống xác định nào đó, chẳng hạn cảnh huống trong hình 1.2 bên phải.

Trong bài toán này, trạng thái ban đầu là cảnh huống ở bên trái hình 1.2, còn trạng thái kết thúc ở bên phải hình 1.2. Tương ứng với các quy tắc chuyển dịch các quân, ta có bốn toán tử: ***up*** (đẩy quân lên trên), ***down*** (đẩy quân xuống dưới), ***left*** (đẩy quân sang trái), ***right*** (đẩy quân sang phải). Rõ ràng là, các toán tử này chỉ là các toán tử bộ phận; chẳng hạn, từ trạng thái ban đầu (hình 1.2 bên trái), ta chỉ có thể áp dụng các toán tử ***down, left, right***.

Trong các ví dụ trên việc tìm ra một biểu diễn thích hợp để mô tả các trạng thái của vấn đề là khá dễ dàng và tự nhiên. Song trong nhiều vấn đề việc tìm hiểu được biểu diễn thích hợp cho các trạng thái của vấn đề là hoàn toàn không đơn giản. Việc tìm ra dạng biểu diễn tốt cho các trạng thái đóng vai trò hết sức quan trọng trong quá trình giải quyết một vấn đề. Có thể nói rằng, nếu ta tìm được dạng biểu diễn tốt cho các trạng thái của vấn đề, thì vấn đề hầu như đã được giải quyết.

Ví dụ 2: Vấn đề triệu phú và kẻ cướp. Có ba nhà triệu phú và ba tên cướp ở bên bờ tả ngạn một con sông, cùng một chiếc thuyền chở được một hoặc hai người. Hãy tìm cách đưa mọi người qua sông sao cho không để lại ở bên bờ sông kẻ cướp nhiều hơn triệu phú. Đương nhiên trong bài toán này, các toán tử tương ứng với các hành động chở 1 hoặc 2 người qua sông. Nhưng ở đây ta cần lưu ý rằng, khi hành động xảy ra (lúc thuyền đang bơi qua sông) thì ở bên bờ sông thuyền vừa dời chỗ, số kẻ cướp không được nhiều hơn số triệu phú. Tiếp theo ta cần quyết định cái gì là trạng thái của vấn đề. ở đây ta không cần phân biệt các nhà triệu phú và các tên cướp, mà chỉ số lượng của họ ở bên bờ sông là quan trọng. Để biểu diễn các trạng thái, ta sử dụng bộ ba (a, b, k), trong đó a là số triệu phú, b là số kẻ cướp ở bên bờ tả ngạn vào các thời điểm mà thuyền ở bờ này hoặc bờ kia, k = 1 nếu thuyền ở bờ tả ngạn và k = 0 nếu thuyền ở bờ hữu ngạn. Như vậy, không gian trạng thái cho bài toán triệu phú và kẻ cướp được xác định như sau:

- Trạng thái ban đầu là (3, 3, 1).
- Các toán tử. Có năm toán tử tương ứng với hành động thuyền chở qua sông 1 triệu phú, hoặc 1 kẻ cướp, hoặc 2 triệu phú, hoặc 2 kẻ cướp, hoặc 1 triệu phú và 1 kẻ cướp.
- Trạng thái kết thúc là (0, 0, 0).

1.2 Các chiến lược tìm kiếm

Như ta đã thấy trong mục 1.1, để giải quyết một vấn đề bằng tìm kiếm trong không gian trạng thái, đầu tiên ta cần tìm dạng thích hợp mô tả các trạng thái cảu vấn đề. Sau đó cần xác định:

- Trạng thái ban đầu.

- Tập các toán tử.
- Tập T các trạng thái kết thúc. (T có thể không được xác định cụ thể gồm các trạng thái nào mà chỉ được chỉ định bởi một số điều kiện nào đó).

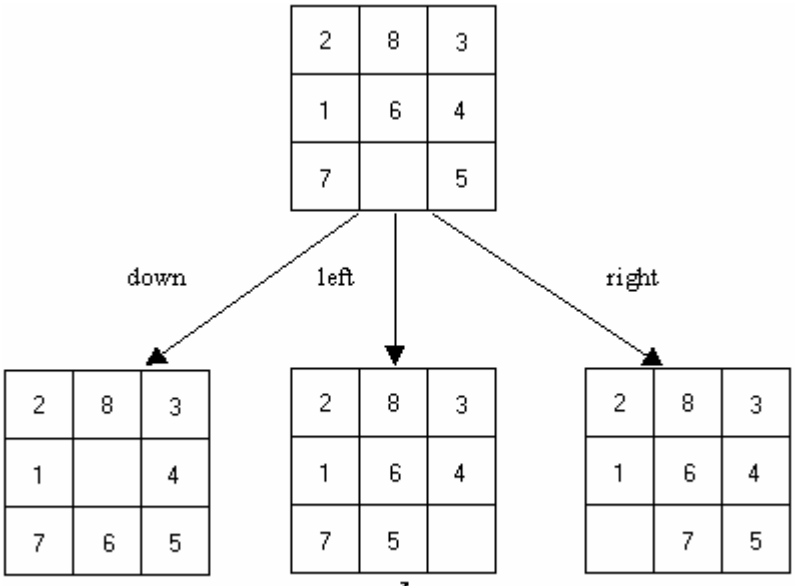
Giả sử u là một trạng thái nào đó và R là một toán tử biến đổi u thành v. Ta sẽ gọi v là trạng thái kề u, hoặc v được sinh ra từ trạng thái u bởi toán tử R. Quá trình áp dụng các toán tử để sinh ra các trạng thái kề u được gọi là phát triển trạng thái u. Chẳng hạn, trong bài toán toán số, phát triển trạng thái ban đầu (hình 2 bên trái), ta nhận được ba trạng thái kề (hình 1.3).

Khi chúng ta biểu diễn một vấn đề cần giải quyết thông qua các trạng thái và các toán tử thì việc tìm lời giải của vấn đề được quy về việc tìm đường đi từ trạng thái ban đầu tới một trạng thái kết thúc nào đó.

Có thể phân các chiến lược tìm kiếm thành hai loại:

- Các chiến lược tìm kiếm mù. Trong các chiến lược tìm kiếm này, không có một sự hướng dẫn nào cho sự tìm kiếm, mà ta chỉ phát triển các trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có hai kỹ thuật tìm kiếm mù, đó là tìm kiếm theo bề rộng và tìm kiếm theo độ sâu.

Tư tưởng của tìm kiếm theo bề rộng là các trạng thái được phát triển theo thứ tự mà chúng được sinh ra, tức là trạng thái nào được sinh ra trước sẽ được phát triển trước.



Hình 1.3 Phát triển một trạng thái

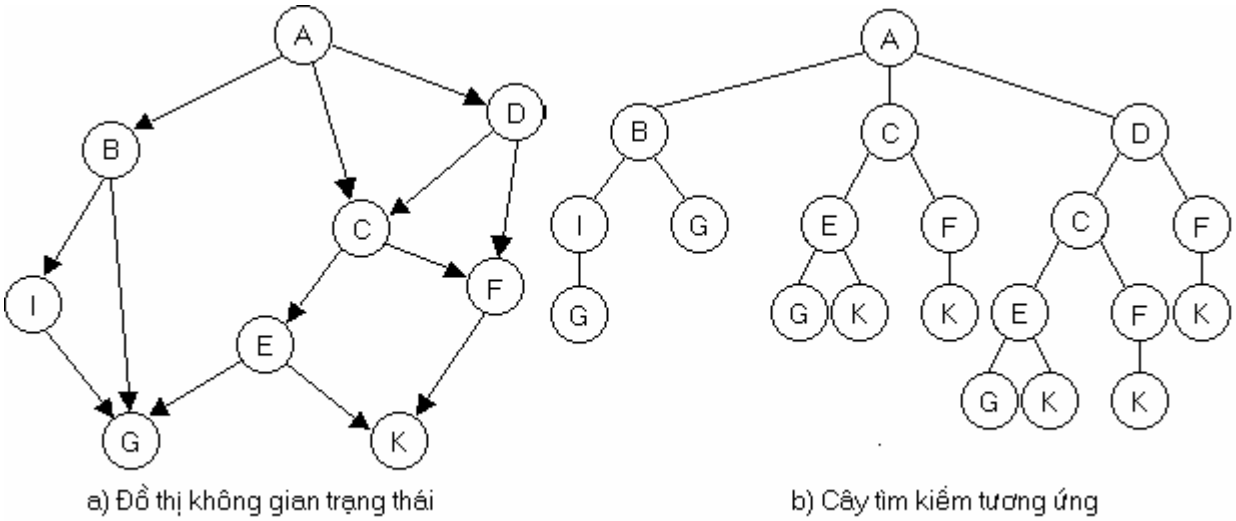
Trong nhiều vấn đề, dù chúng ta phát triển các trạng thái theo hệ thống nào (theo bề rộng hoặc theo độ sâu) thì số lượng các trạng thái được sinh ra trước khi ta gặp trạng thái đích thường là cực kỳ lớn. Do đó các thuật toán tìm kiếm mù kém hiệu quả, đòi hỏi rất nhiều không gian và thời gian. Trong thực tế, nhiều vấn đề không thể giải quyết được bằng tìm kiếm mù.

 Tìm kiếm kinh nghiệm (tìm kiếm heuristic). Trong rất nhiều vấn đề, chúng ta có thể dựa vào sự hiểu biết của chúng ta về vấn đề, dựa vào kinh nghiệm, trực giác, để đánh giá các trạng thái. Sử dụng sự đánh giá các trạng thái để hướng dẫn sự tìm kiếm: trong quá trình phát triển các trạng thái, ta sẽ chọn trong số các trạng thái chờ phát triển, trạng thái được đánh giá là tốt nhất để phát triển. Do đó tốc độ tìm kiếm sẽ nhanh hơn. Các phương pháp tìm kiếm dựa vào sự đánh giá các trạng thái để hướng dẫn sự tìm kiếm gọi chung là các phương pháp tìm kiếm kinh nghiệm.

Như vậy chiến lược tìm kiếm được xác định bởi chiến lược chọn trạng thái để phát triển ở mỗi bước. Trong tìm kiếm mù, ta chọn trạng thái để phát triển theo thứ tự mà đúng được sinh ra; còn trong tìm kiếm kinh nghiệm ta chọn trạng thái dựa vào sự đánh giá các trạng thái.

Cây tìm kiếm

Chúng ta có thể nghĩ đến quá trình tìm kiếm như quá trình xây dựng ***cây tìm kiếm***. Cây tìm kiếm là cây mà các đỉnh được gắn bởi các trạng thái của không gian trạng thái. Gốc của cây tìm kiếm tương ứng với trạng thái ban đầu. Nếu một đỉnh ứng với trạng thái u, thì các đỉnh con của nó ứng với các trạng thái v kề u. Hình



Hình 1.4

1.4a là đồ thị biểu diễn một không gian trạng thái với trạng thái ban đầu là A, hình 1.4b là cây tìm kiếm tương ứng với không gian trạng thái đó.

Mỗi chiến lược tìm kiếm trong không gian trạng thái tương ứng với một phương pháp xây dựng cây tìm kiếm. Quá trình xây dựng cây bắt đầu từ cây chỉ có một đỉnh là trạng thái ban đầu. Giả sử tới một bước nào đó trong chiến lược tìm kiếm, ta đã xây dựng được một cây nào đó, các lá của cây tương ứng với các trạng thái chưa được phát triển. Bước tiếp theo phụ thuộc vào chiến lược tìm kiếm mà một đỉnh nào đó trong các lá được chọn để phát triển. Khi phát triển đỉnh đó, cây tìm kiếm được mở rộng bằng cách thêm vào các đỉnh con của đỉnh đó. Kỹ thuật

tìm kiếm theo bề rộng (theo độ sâu) tương ứng với phương pháp xây dựng cây tìm kiếm theo bề rộng (theo độ sâu).

1.3 Các chiến lược tìm kiếm mù

Trong mục này chúng ta sẽ trình bày hai chiến lược tìm kiếm mù: tìm kiếm theo bề rộng và tìm kiếm theo độ sâu. Trong tìm kiếm theo bề rộng, tại mỗi bước ta sẽ chọn trạng thái để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác. Còn trong tìm kiếm theo độ sâu, trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển.

Chúng ta sử dụng danh sách L để lưu các trạng thái đã được sinh ra và chờ được phát triển. Mục tiêu của tìm kiếm trong không gian trạng thái là tìm đường đi từ trạng thái ban đầu tới trạng thái đích, do đó ta cần lưu lại vết của đường đi. Ta có thể sử dụng hàm father để lưu lại cha của mỗi đỉnh trên đường đi, ***father***(v) = u nếu cha của đỉnh v là u.

1.3.1 Tìm kiếm theo bề rộng

Thuật toán tìm kiếm theo bề rộng được mô tả bởi thủ tục sau:

procedure *Breadth_First_Search***;**

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo tìm kiếm thất bại***; stop};**

*2.2 Loại trạng thái u ở đầu danh sách L***;**

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo tìm kiếm thành công***; stop};**

2.4 **for** *mỗi trạng thái v kề u* **do {**

*Đặt v vào cuối danh sách L***;**

*father(v) <- u***}**

end;

Chúng ta có một số nhận xét sau đây về thuật toán tìm kiếm theo bề rộng:

 Trong tìm kiếm theo bề rộng, trạng thái nào được sinh ra trước sẽ được phát triển trước, do đó danh sách L được xử lý như hàng đợi. Trong bước 2.3, ta cần kiểm tra xem u có là trạng thái kết thúc hay không. Nói chung các trạng thái kết thúc được xác định bởi một số điều kiện nào đó, khi đó ta cần kiểm tra xem u có thỏa mãn các điều kiện đó hay không.

 Nếu bài toán có nghiệm (tồn tại đường đi từ trạng thái ban đầu tới trạng thái đích), thì thuật toán tìm kiếm theo bề rộng sẽ tìm ra nghiệm, đồng thời đường đi

tìm được sẽ là ngắn nhất. Trong trường hợp bài toán vô nghiệm và không gian trạng thái hữu hạn, thuật toán sẽ dừng và cho thông báo vô nghiệm.

Đánh giá tìm kiếm theo bề rộng

Bây giờ ta đánh giá thời gian và bộ nhớ mà tìm kiếm theo bề rộng đòi hỏi. Giả sử rằng, mỗi trạng thái khi được phát triển sẽ sinh ra b trạng thái kề. Ta sẽ gọi b là ***nhân tố nhánh***. Giả sử rằng, nghiệm của bài toán là đường đi có độ dài d. Bởi nhiều nghiệm có thể được tìm ra tại một đỉnh bất kỳ ở mức d của cây tìm kiếm, do đó số đỉnh cần xem xét để tìm ra nghiệm là:

$\mathbf{1 + b + b^2 + ... + b^{d-1} + k}$

Trong đó k có thể là 1, 2, ..., b^d. Do đó số lớn nhất các đỉnh cần xem xét là:

$\mathbf{1 + b + b^2 + ... + b^d}$

Như vậy, độ phức tạp thời gian của thuật toán tìm kiếm theo bề rộng là $O(b^d)$. Độ phức tạp không gian cũng là $O(b^d)$, bởi vì ta cần lưu vào danh sách L tất cả các đỉnh của cây tìm kiếm ở mức d, số các đỉnh này là b^d.

Để thấy rõ tìm kiếm theo bề rộng đòi hỏi thời gian và không gian lớn tới mức nào, ta xét trường hợp nhân tố nhánh b = 10 và độ sâu d thay đổi. Giả sử để phát hiện và kiểm tra 1000 trạng thái cần 1 giây, và lưu giữ 1 trạng thái cần 100 bytes. Khi đó thời gian và không gian mà thuật toán đòi hỏi được cho trong bảng sau:

Độ sâu d	**Thời gian**	**Không gian**
4	11 giây	1 megabyte
6	18 giây	111 megabytes
8	31 giờ	11 gigabytes
10	128 ngày	1 terabyte
12	35 năm	111 terabytes
14	3500 năm	11.111 terabytes

1.3.2 Tìm kiếm theo độ sâu

Như ta đã biết, tư tưởng của chiến lược tìm kiếm theo độ sâu là, tại mỗi bước trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển. Do đó thuật toán tìm kiếm theo độ sâu là hoàn toàn tương tự như thuật toán tìm kiếm theo bề rộng, chỉ có một điều khác là, ta xử lý danh sách L các trạng thái chờ phát triển không phải như hàng đợi mà như ngăn xếp. Cụ thể là trong bước 2.4 của thuật toán tìm kiếm theo bề rộng, ta cần sửa lại là "Đặt v vào ***đầu*** danh sách L".

Sau đây chúng ta sẽ đưa ra các nhận xét so sánh hai chiến lược tìm kiếm mù:

 Thuật toán tìm kiếm theo bề rộng luôn luôn tìm ra nghiệm nếu bài toán có nghiệm. Song không phải với bất kỳ bài toán có nghiệm nào thuật toán tìm kiếm theo độ sâu cũng tìm ra nghiệm! Nếu bài toán có nghiệm và không gian trạng thái hữu hạn, thì thuật toán tìm kiếm theo độ sâu sẽ tìm ra nghiệm. Tuy nhiên, trong trường hợp không gian trạng thái vô hạn, thì có thể nó không tìm ra nghiệm, lý do là ta luôn luôn đi xuống theo độ sâu, nếu ta đi theo một nhánh vô hạn mà nghiệm không nằm trên nhánh đó thì thuật toán sẽ không dừng. Do đó người ta khuyên rằng, không nên áp dụng tìm kiếm theo dộ sâu cho các bài toán có cây tìm kiếm chứa các nhánh vô hạn.

 Độ phức tạp của thuật toán tìm kiếm theo độ sâu.

Giả sử rằng, nghiệm của bài toán là đường đi có độ dài d, cây tìm kiếm có nhân tố nhánh là b và có chiều cao là d. Có thể xẩy ra, nghiệm là đỉnh ngoài cùng bên phải trên mức d của cây tìm kiếm, do đó độ phức tạp thời gian của tìm kiếm theo độ sâu trong trường hợp xấu nhất là $O(b^d)$, tức là cũng như tìm kiếm theo bề rộng. Tuy nhiên, trên thực tế đối với nhiều bài toán, tìm kiếm theo độ sâu thực sự nhanh hơn tìm kiếm theo bề rộng. Lý do là tìm kiếm theo bề rộng phải xem xét toàn bộ cây tìm kiếm tới mức d-1, rồi mới xem xét các đỉnh ở mức d. Còn trong tìm kiếm theo độ sâu, có thể ta chỉ cần xem xét một bộ phận nhỏ của cây tìm kiếm thì đã tìm ra nghiệm.

Để đánh giá độ phức tạp không gian của tìm kiếm theo độ sâu ta có nhận xét rằng, khi ta phát triển một đỉnh u trên cây tìm kiếm theo độ sâu, ta chỉ cần lưu các đỉnh chưa được phát triển mà chúng là các đỉnh con của các đỉnh nằm trên đường đi từ gốc tới đỉnh u. Như vậy đối với cây tìm kiếm có nhân tố nhánh b và độ sâu lớn nhất là d, ta chỉ cần lưu ít hơn db đỉnh. Do đó độ phức tạp không gian của tìm kiếm theo độ sâu là O(db), trong khi đó tìm kiếm theo bề rộng đòi hỏi không gian nhớ $O(b^d)$!

1.3.3 Các trạng thái lặp

Như ta thấy trong mục 1.2, cây tìm kiếm có thể chứa nhiều đỉnh ứng với cùng một trạng thái, các trạng thái này được gọi là trạng thái lặp. Chẳng hạn, trong cây tìm kiếm hình 4b, các trạng thái C, E, F là các trạng thái lặp. Trong đồ thị biểu diễn không gian trạng thái, các trạng thái lặp ứng với các đỉnh có nhiều đường đi dẫn tới nó từ trạng thái ban đầu. Nếu đồ thị có chu trình thì cây tìm kiếm sẽ chứa các nhánh với một số đỉnh lập lại vô hạn lần. Trong các thuật toán tìm kiếm sẽ lãng phí rất nhiều thời gian để phát triển lại các trạng thái mà ta đã gặp và đã phát triển. Vì vậy trong quá trình tìm kiếm ta cần tránh phát sinh ra các trạng thái mà ta đã phát triển. Chúng ta có thể áp dụng một trong các giải pháp sau đây:

1. Khi phát triển đỉnh u, không sinh ra các đỉnh trùng với cha của u.
2. Khi phát triển đỉnh u, không sinh ra các đỉnh trùng với một đỉnh nào đó nằm trên đường đi dẫn tới u.

3. Không sinh ra các đỉnh mà nó đã được sinh ra, tức là chỉ sinh ra các đỉnh mới.

Hai giải pháp đầu dễ cài đặt và không tốn nhiều không gian nhớ, tuy nhiên các giải pháp này không tránh được hết các trạng thái lặp.

Để thực hiện giải pháp thứ 3 ta cần lưu các trạng thái đã phát triển vào tập Q, lưu các trạng thái chờ phát triển vào danh sách L. Đương nhiên, trạng thái v lần đầu được sinh ra nếu nó không có trong Q và L. Việc lưu các trạng thái đã phát triển và kiểm tra xem một trạng thái có phải lần đầu được sinh ra không đòi hỏi rất nhiều không gian và thời gian. Chúng ta có thể cài đặt tập Q bởi bảng băm (xem [ ]).

1.3.4 Tìm kiếm sâu lặp

Như chúng ta đã nhận xét, nếu cây tìm kiếm chứa nhánh vô hạn, khi sử dụng tìm kiếm theo độ sâu, ta có thể mắc kẹt ở nhánh đó và không tìm ra nghiệm. Để khắc phục hoàn cảnh đó, ta tìm kiếm theo độ sâu chỉ tới mức d nào đó; nếu không tìm ra nghiệm, ta tăng độ sâu lên d+1 và lại tìm kiếm theo độ sâu tới mức d+1. Quá trình trên được lặp lại với d lần lượt là 1, 2, ... đến một độ sâu max nào đó. Như vậy, thuật toán tìm kiếm sâu lặp (iterative deepening search) sẽ sử dụng thủ tục tìm kiếm sâu hạn chế (depth_limited search) như thủ tục con. Đó là thủ tục tìm kiếm theo độ sâu, nhưng chỉ đi tới độ sâu d nào đó rồi quay lên.

Trong thủ tục tìm kiếm sâu hạn chế, d là tham số độ sâu, hàm depth ghi lại độ sâu của mỗi đỉnh

procedure *Depth_Limited_Search(d)***;**

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu u_0**;**

depth(u_0*)* ← *0***;**

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại***; stop};**

*2.2 Loại trạng thái u ở đầu danh sách L***;**

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo thành công***; stop};**

2.4 **if** *depth(u) <= d* **then**

for *mỗi trạng thái v kề u* **do**

{*Đặt v vào đầu danh sách L***;**

depth(v) ← *depth(u) + 1***};**

end;

procedure *Depth_Deepening_Search*;

begin

for *d* ← *0* **to** *max* **do**

{*Depth_Limited_Search(d)*;

if *thành công* **then exit}**

end;

Kỹ thuật tìm kiếm sâu lặp kết hợp được các ưu điểm của tìm kiếm theo bề rộng và tìm kiếm theo độ sâu. Chúng ta có một số nhận xét sau:

- Cũng như tìm kiếm theo bề rộng, tìm kiếm sâu lặp luôn luôn tìm ra nghiệm (nếu bài toán có nghiệm), miễn là ta chọn độ sâu mã đủ lớn.
- Tìm kiếm sâu lặp chỉ cần không gian nhớ như tìm kiếm theo độ sâu.
- Trong tìm kiếm sâu lặp, ta phải phát triển lặp lại nhiều lần cùng một trạng thái. Điều đó làm cho ta có cảm giác rằng, tìm kiếm sâu lặp lãng phí nhiều thời gian. Thực ra thời gian tiêu tốn cho phát triển lặp lại các trạng thái là không đáng kể so với thời gian tìm kiếm theo bề rộng. Thật vậy, mỗi lần gọi thủ tục tìm kiếm sâu hạn chế tới mức d, nếu cây tìm kiếm có nhân tố nhánh là b, thì số đỉnh cần phát triển là:

$$\mathbf{1 + b + b^2 + ... + b^d}$$

Nếu nghiệm ở độ sâu d, thì trong tìm kiếm sâu lặp, ta phải gọi thủ tục tìm kiếm sâu hạn chế với độ sâu lần lượt là 0, 1, 2, ..., d. Do đó các đỉnh ở mức 1 phải phát triển lặp d lần, các đỉnh ở mức 2 lặp d-1 lần, ..., các đỉnh ở mức d lặp 1 lần. Như vậy tổng số đỉnh cần phát triển trong tìm kiếm sâu lặp là:

$$\mathbf{(d+1)1 + db + (d-1)b^2 + ... + 2b^{d-1} + 1b^d}$$

Do đó thời gian tìm kiếm sâu lặp là $O(b^d)$.

Tóm lại, tìm kiếm sâu lặp có độ phức tạp thời gian là $O(b^d)$ (như tìm kiếm theo bề rộng), và có độ phức tạp không gian là O(biểu diễn) (như tìm kiếm theo độ sâu). Nói chung, chúng ta nên áp dụng tìm kiếm sâu lặp cho các vấn đề có không gian trạng thái lớn và độ sâu của nghiệm không biết trước.

1.4 Quy vấn đề về các vấn đề con. Tìm kiếm trên đồ thị và/hoặc.

1.4.1 Quy vấn đề về các vấn đề con:

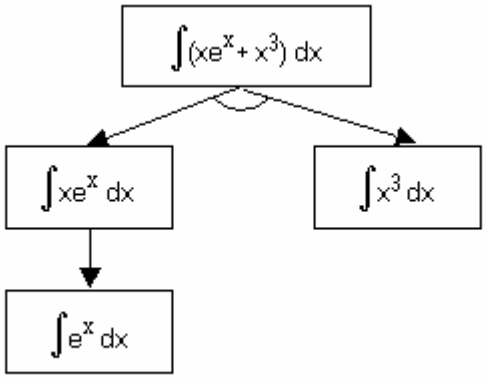
Trong mục 1.1, chúng ta đã nghiên cứu việc biểu diễn vấn đề thông qua các trạng thái và các toán tử. Khi đó việc tìm nghiệm của vấn đề được quy về việc tìm đường trong không gian trạng thái. Trong mục này chúng ta sẽ nghiên cứu một phương pháp luận khác để giải quyết vấn đề, dựa trên việc quy vấn đề về các vấn đề con. Quy vấn đề về các vấn đề con (còn gọi là rút gọn vấn đề) là một phương pháp được sử dụng rộng rãi nhất để giải quyết các vấn đề. Trong đời sống hàng ngày, cũng như trong khoa học kỹ thuật, mỗi khi gặp một vấn đề cần giải quyết, ta

vẫn thường cố gắng tìm cách đưa nó về các vấn đề đơn giản hơn. Quá trình rút gọn vấn đề sẽ được tiếp tục cho tới khi ta dẫn tới các vấn đề con có thể giải quyết được dễ dàng. Sau đây chúng ta xét một số vấn đề.

Vấn đề tính tích phân bất định

Giả sử ta cần tính một tích phân bất định, chẳng hạn $\int (xe^x + x^3)\, dx$. Quá trình chúng ta vẫn thường làm để tính tích phân bất định là như sau. Sử dụng các quy tắc tính tích phân (quy tắc tính tích phân của một tổng, quy tắc tính tích phân từng phần...), sử dụng các phép biến đổi biến số, các phép biến đổi các hàm (chẳng hạn, các phép biến đổi lượng giác),... để đưa tích phân cần tính về tích phân của các hàm số sơ cấp mà chúng ta đã biết cách tính. Chẳng hạn, đối với tích phân $\int (xe^x + x^3)\, dx$, áp dụng quy tắc tích phân của tổng ta đưa về hai tích phân $\int xe^x dx$ và $\int x^3 dx$. áp dụng quy tắc tích phân từng phần ta đưa tích phân $\int xe^x dx$ về tích phân $\int e^x dx$. Quá trình trên có thể biểu diễn bởi đồ thị trong hình 1.5.

Các tích phân $\int e^x dx$ và $\int x^3 dx$ là các tích phân cơ bản đã có trong bảng tích phân. Kết hợp các kết quả của các tích phân cơ bản, ta nhận được kết quả của tích



Hình 1.5 Quy một số tích phân về các tích phân cơ bản.

phân đã cho.

Chúng ta có thể biểu diễn việc quy một vấn đề về các vấn đề con cơ bởi các trạng thái và các toán tử. ở đây, bài toán cần giải là trạng thái ban đầu. Mỗi cách quy bài toán về các bài toán con được biểu diễn bởi một toán tử, toán tử $A \rightarrow B, C$ biểu diễn việc quy bài toán A về hai bài toán B và C. Chẳng hạn, đối với bài toán tính tích phân bất định, ta có thể xác định các toán tử dạng:

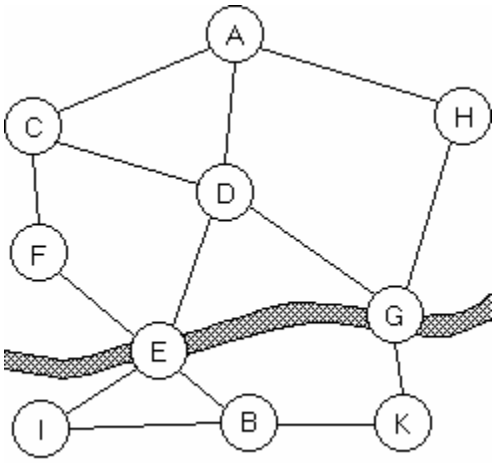
$\int (f_1 + f_2)\, dx \rightarrow \int f_1\, dx, \int f_2\, dx$ và $\int u\, dv \rightarrow \int v\, du$

Các trạng thái kết thúc là các bài toán sơ cấp (các bài toán đã biết cách giải). Chẳng hạn, trong bài toán tính tích phân, các tích phân cơ bản là các trạng thái kết thúc. Một điều cần lưu ý là, trong không gian trạng thái biểu diễn việc quy vấn đề về các vấn đề con, các toán tử có thể là đa trị, nó biến đổi một trạng thái thành nhiều trạng thái khác.

Vấn đề tìm đường đi trên bản đồ giao thông

Bài toán này đã được phát triển như bài toán tìm đường đi trong không gian trạng thái (xem 1.1), trong đó mỗi trạng thái ứng với một thành phố, mỗi toán tử ứng với một con đường nối, nối thành phố này với thành phố khác. Bây giờ ta đưa ra một cách biểu diễn khác dựa trên việc quy vấn đề về các vấn đề con. Giả sử ta có bản đồ giao thông trong một vùng lãnh thổ (xem hình 1.6). Giả sử ta cần tìm đường đi từ thành phố A tới thành phố B. Có con sông chảy qua hai thành phố E và G và có cầu qua sông ở mỗi thành phố đó. Mọi đường đi từ A đến B chỉ có thể qua E hoặc G. Như vậy bài toán tìm đường đi từ A đến B được quy về:

1) Bài toán tìm đường đi từ A đến B qua E (hoặc)
2) Bài toán tìm đường đi từ A đến b qua G.

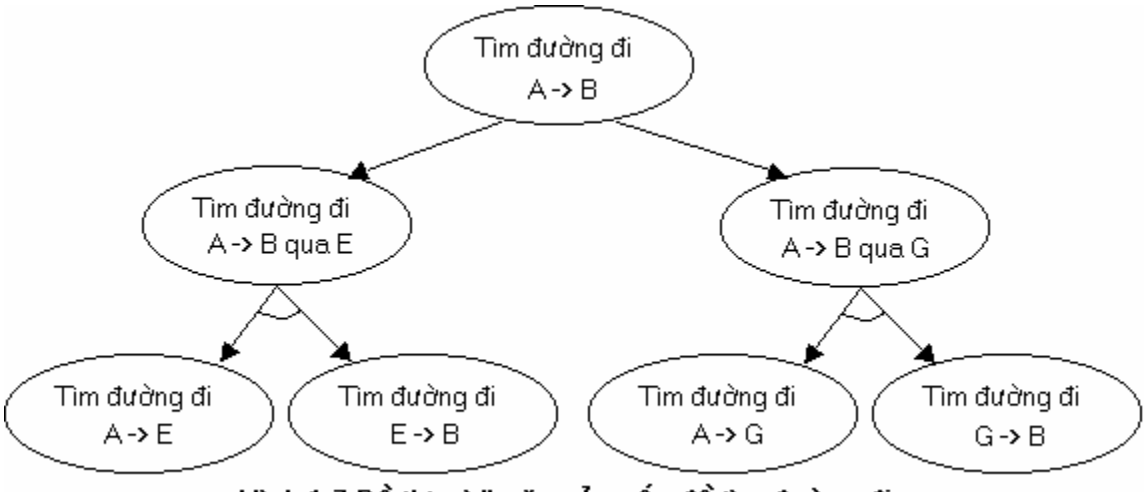


Hình 1.6

Mỗi một trong hai bài toán trên lại có thể phân nhỏ như sau

1) Bài toán tìm đường đi từ A đến B qua E được quy về:
    1.1 Tìm đường đi từ A đến E (và)
    1.2 Tìm đường đi từ E đến B.
2) Bài toán tìm đường đi từ A đến B qua G được quy về:
    2.1 Tìm đường đi từ A đến G (và)
    2.2 Tìm đường đi từ G đến B.

Quá trình rút gọn vấn đề như trên có thể biểu diễn dưới dạng đồ thị (đồ thị và/hoặc) trong hình 1.7. ở đây mỗi bài toán tìm đường đi từ một thành phố tới một thành phố khác ứng với một trạng thái. Các trạng thái kết thúc là các trạng thái ứng với các bài toán tìm đường đi, chẳng hạn từ A đến C, hoặc từ D đến E, bởi vì đã có đường nối A với C, nối D với E.

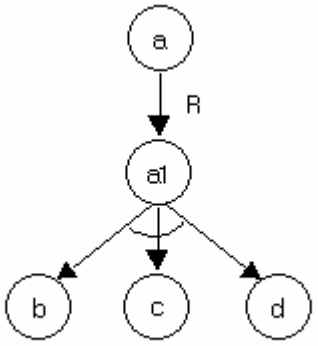


Hình 1.7 Đồ thị và/hoặc của vấn đề tìm đường đi.

1.4.2 Đồ thị và/hoặc

Không gian trạng thái mô tả việc quy vấn đề về các vấn đề con có thể biểu diễn dưới dạng đồ thị định hướng đặc biệt được gọi là đồ thị và/hoặc. Đồ thị này được xây dựng như sau:

Mỗi bài toán ứng với một đỉnh của đồ thị. Nếu có một toán tử quy một bài toán về một bài toán khác, chẳng hạn R : a →b, thì trong đồ thị sẽ có cung gán nhãn đi từ đỉnh a tới đỉnh b. Đối với mỗi toán tử quy một bài toán về một số bài toán con, chẳng hạn R : a →b, c, d ta đưa vào một đỉnh mới a_1, đỉnh này biểu diễn tập các bài toán con {b, c, d} và toán tử R : a →b, c, d được biểu diễn bởi đồ thị



Hình 1.8 Đồ thị biểu diễn toán tử R : a->b, c, d

hình 1.8.

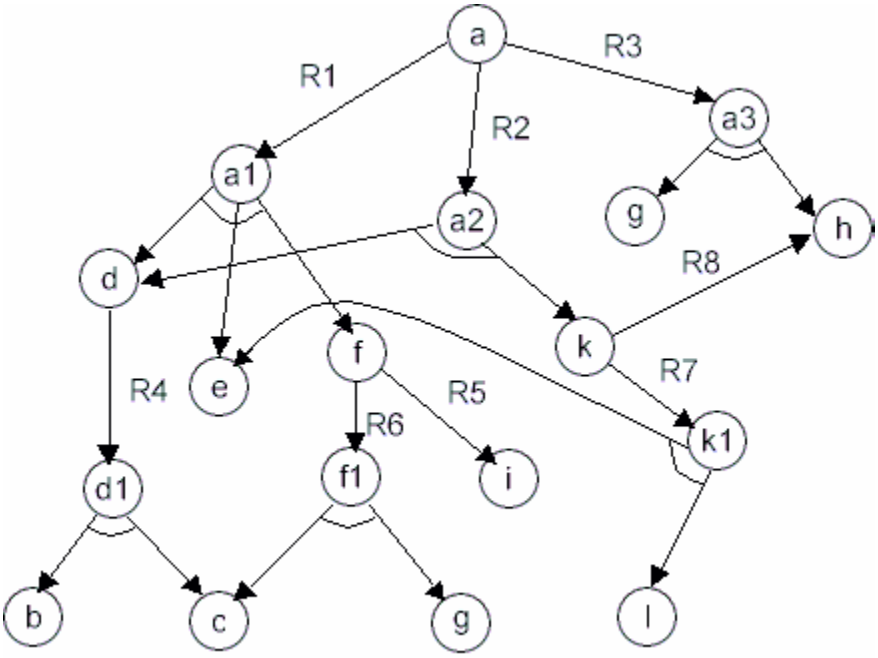
Ví dụ: Giả sử chúng ta có không gian trạng thái sau:

- Trạng thái ban đầu (bài toán cần giải) là a.
- Tập các toán tử quy gồm:

R_1 : a →d, e, f
R_2 : a →d, k
R_3 : a →g, h
R_4 : d →b, c
R_5 : f →i
R_6 : f →c, j
R_7 : k →e, l

$R_8 : k \rightarrow h$

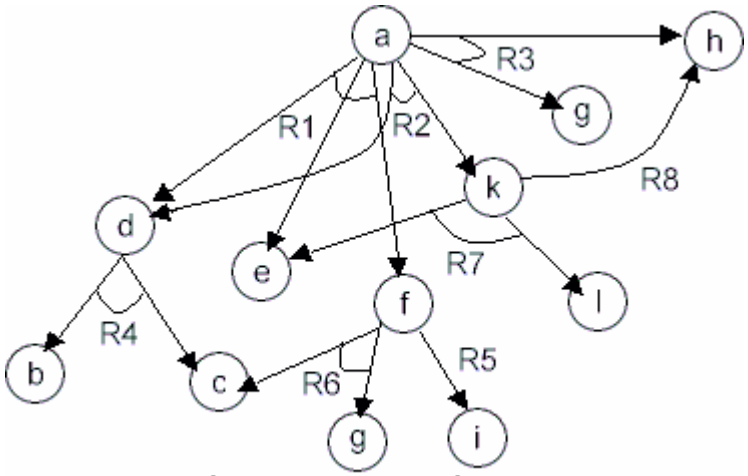
- Tập các trạng thái kết thúc (các bài toán sơ cấp) là T = {b, c, e, j, l}.



Hình 1.9 Một đồ thị và/hoặc

Không gian trạng thái trên có thể biểu diễn bởi đồ thị và/hoặc trong hình 1.9. Trong đồ thị đó, các đỉnh, chẳng hạn a_1, a_2, a_3 được gọi là đỉnh ***và***, các đỉnh chẳng hạn a, f, k được gọi là đỉnh ***hoặc***. Lý do là, đỉnh a_1 biểu diễn tập các bài toán {d, e, f} và a_1 được giải quyết nếu d và e và f được giải quyết. Còn tại đỉnh a, ta có các toán tử R_1, R_2, R_3 quy bài toán a về các bài toán con khác nhau, do đó a được giải quyết nếu hoặc a_1 = {d, e, f}, hoặc a_2 = {d, k}, hoặc a_3 = {g, h} được giải quyết.

Người ta thường sử dụng đồ thị và/hoặc ở dạng rút gọn. Chẳng hạn, đồ thị và/hoặc trong hình 1.9 có thể rút gọn thành đồ thị trong hình 1.10. Trong đồ thị rút gọn này, ta sẽ nói chẳng hạn d, e, f là các đỉnh kề đỉnh a theo toán tử R_1, còn d, k là các đỉnh kề a theo toán tử R_2.



Hình 1.10 Đồ thị rút gọn của đồ thị trong hình 1.9

Khi đã có các toán tử rút gọn vấn đề, thì bằng cách áp dụng liên tiếp các toán tử, ta có thể đưa bài toán cần giải về một tập các bài toán con. Chẳng hạn, trong ví dụ trên nếu ta áp dụng các toán tử R_1, R_4, R_6, ta sẽ quy bài toán a về tập các bài toán con {b, c, e, f}, tất cả các bài toán con này đều là sơ cấp. Từ các toán tử R_1, R_4 và R_6 ta xây dựng được một cây trong hình 1.11a, cây này được gọi là cây nghiệm. Cây nghiệm được định nghĩa như sau:

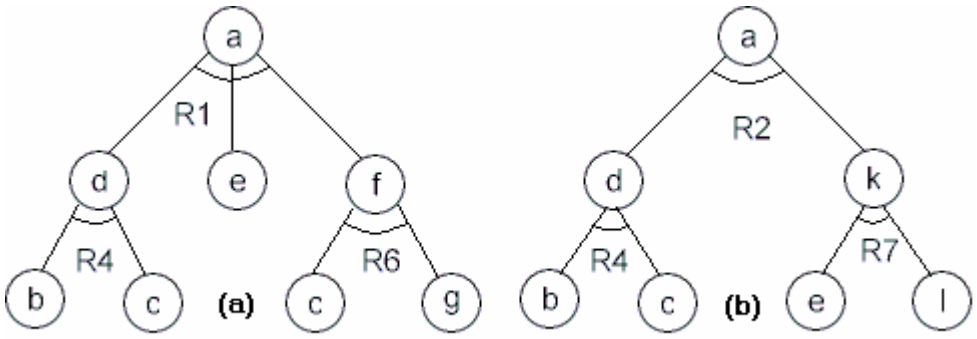
Cây nghiệm là một cây, trong đó:

- Gốc của cây ứng với bài toán cần giải.
- Tất cả các lá của cây là các đỉnh kết thúc (đỉnh ứng với các bài toán sơ cấp).
- Nếu u là đỉnh trong của cây, thì các đỉnh con của u là các đỉnh kề u theo một toán tử nào đó.

Các đỉnh của đồ thị và/hoặc sẽ được gắn nhãn giải được hoặc không giải được.

Các đỉnh ***giải được*** được xác định đệ quy như sau:

- Các đỉnh kết thúc là các đỉnh ***giải được*.**
- Nếu u không phải là đỉnh kết thúc, nhưng có một toán tử R sao cho tất cả các đỉnh kề u theo R đều giải được thì u ***giải được*.**



Hình 1.11 Các cây nghiệm

Các đỉnh ***không giải được*** được xác định đệ quy như sau:

- Các đỉnh không phải là đỉnh kết thúc và không có đỉnh kề, là các đỉnh ***không giải được*.**
- Nếu u không phải là đỉnh kết thúc và với mọi toán tử R áp dụng được tại u đều có một đỉnh v kề u theo R không giải được, thì u ***không giải được*.**

Ta có nhận xét rằng, nếu bài toán a ***giải được*** thì sẽ có một cây nghiệm gốc a, và ngược lại nếu có một cây nghiệm gốc a thì a ***giải được*.** Hiển nhiên là, một bài toán giải được có thể có nhiều cây nghiệm, mỗi cây nghiệm biểu diễn một cách giải bài toán đó. Chẳng hạn trong ví dụ đã nêu, bài toán a có hai cây nghiệm trong hình 1.11.

Thứ tự giải các bài toán con trong một cây nghiệm là như sau. Bài toán ứng với đỉnh u chỉ được giải sau khi tất cả các bài toán ứng với các đỉnh con của u đã được giải. Chẳng hạn, với cây nghiệm trong hình 1.11a, thứ tự giải các bài toán có thể là b, c, d, j, f, e, a. ta có thể sử dụng thủ tục sắp xếp topo (xem [ ]) để sắp xếp thứ tự các bài toán trong một cây nghiệm. Đương nhiên ta cũng có thể giải quyết đồng thời các bài toán con ở cùng một mức trong cây nghiệm.

Vấn đề của chúng ta bây giờ là, tìm kiếm trên đồ thị và/hoặc để xác định được đỉnh ứng với bài toán ban đầu là giải được hay không giải được, và nếu nó giải được thì xây dựng một cây nghiệm cho nó.

1.4.3 Tìm kiếm trên đồ thị và/hoặc

Ta sẽ sử dụng kỹ thuật tìm kiếm theo độ sâu trên đồ thị và/hoặc để đánh dấu các đỉnh. Các đỉnh sẽ được đánh dấu giải được hoặc không giải được theo định nghĩa đệ quy về đỉnh giải được và không giải được. Xuất phát từ đỉnh ứng với bài toán ban đầu, đi xuống theo độ sâu, nếu gặp đỉnh u là đỉnh kết thúc thì nó được đánh dấu giải được. Nếu gặp đỉnh u không phải là đỉnh kết thúc và từ u không đi tiếp được, thì u được đánh dấu không giải được. Khi đi tới đỉnh u, thì từ u ta lần lượt đi xuống các đỉnh v kề u theo một toán tử R nào đó. Nếu đánh dấu được một đỉnh v không giải được thì không cần đi tiếp xuống các đỉnh v còn lại. Tiếp tục đi xuống các đỉnh kề u theo một toán tử khác. Nếu tất cả các đỉnh kề u theo một toán tử nào đó được đánh dấu giải được thì u sẽ được đánh dấu giải được và quay lên cha của u. Còn nếu từ u đi xuống các đỉnh kề nó theo mọi toán tử đều gặp các đỉnh kề được đánh dấu không giải được, thì u được đánh dấu không giải được và quay lên cha của u.

Ta sẽ biểu diễn thủ tục tìm kiếm theo độ sâu và đánh dấu các đỉnh đã trình bày trên bởi hàm đệ quy Solvable(u). Hàm này nhận giá trị true nếu u giải được và nhận giá trị ***false*** nếu u không giải được. Trong hàm Solvable(u), ta sẽ sử dụng:

- Biến Ok. Với mỗi toán tử R áp dụng được tại u, biến Ok nhận giá trị ***true*** nếu tất cả các đỉnh v kề u theo R đều giải được, và Ok nhận giá trị ***false*** nếu có một đỉnh v kề u theo R không giải được.
- Hàm Operator(u) ghi lại toán tử áp dụng thành công tại u, tức là Operator(u) = R nếu mọi đỉnh v kề u theo R đều giải được.

function *Solvable(u)***;**

begin

1. **if** *u là đỉnh kết thúc* **then**

{*Solvable* ← **true; stop};**

2. **if** *u không là đỉnh kết thúc và không có đỉnh kề* **then**

{*Solvable(u)* ← **false; stop};**

3. **for** *mỗi toán tử R áp dụng được tại u* **do**

{*Ok* ← *true***;**

for *mỗi v kề u theo R* **do**

if *Solvable(v)* = **false then {***Ok* ← **false; exit};**

if *Ok* **then**

{*Solvable(u)* ← **true;** *Operator(u)* ← *R***; stop}}**

4. Solvable(u) ← **false;**

end;

Nhận xét

 Hoàn toàn tương tự như thuật toán tìm kiếm theo độ sâu trong không gian trạng thái (mục 1.3.2), thuật toán tìm kiếm theo độ sâu trên đồ thị và/hoặc sẽ xác định được bài toán ban đầu là giải được hay không giải được, nếu cây tìm kiếm không có nhánh vô hạn. Nếu cây tìm kiếm có nhánh vô hạn thì chưa chắc thuật toán đã dừng, vì có thể nó bị xa lầy khi đi xuống nhánh vô hạn. Trong trường hợp này ta nên sử dụng thuật toán tìm kiếm sâu lặp (mục 1.3.3).

Nếu bài toán ban đầu giải được, thì bằng cách sử dụng hàm Operator ta sẽ xây dựng được cây nghiệm.

Chương II

Các chiến lược tìm kiếm kinh nghiệm

Trong chương I, chúng ta đã nghiên cứu việc biểu diễn vấn đề trong không gian trạng thái và các kỹ thuật tìm kiếm mù. Các kỹ thuật tìm kiếm mù rất kém hiệu quả và trong nhiều trường hợp không thể áp dụng được. Trong chương này, chúng ta sẽ nghiên cứu các phương pháp tìm kiếm kinh nghiệm (tìm kiếm heuristic), đó là các phương pháp sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm.

Hàm đánh giá và tìm kiếm kinh nghiệm:

Trong nhiều vấn đề, ta có thể sử dụng kinh nghiệm, tri thức của chúng ta về vấn đề để đánh giá các trạng thái của vấn đề. Với mỗi trạng thái u, chúng ta sẽ xác định một giá trị số h(u), số này đánh giá "sự gần đích" của trạng thái u. Hàm h(u) được gọi là ***hàm đánh giá***. Chúng ta sẽ sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm. Trong quá trình tìm kiếm, tại mỗi bước ta sẽ chọn trạng thái để phát triển là trạng thái có giá trị hàm đánh giá nhỏ nhất, trạng thái này được xem là trạng thái có nhiều hứa hẹn nhất hướng tới đích.

Các kỹ thuật tìm kiếm sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm được gọi chung là các kỹ thuật tìm kiếm kinh nghiệm (heuristic search). Các giai đoạn cơ bản để giải quyết vấn đề bằng tìm kiếm kinh nghiệm như sau:

1. Tìm biểu diễn thích hợp mô tả các trạng thái và các toán tử của vấn đề.

2. Xây dựng hàm đánh giá.

3. Thiết kế chiến lược chọn trạng thái để phát triển ở mỗi bước.

Hàm đánh giá

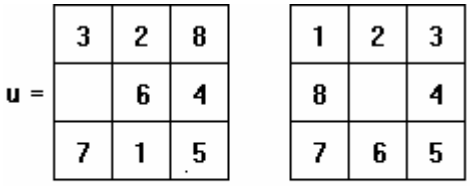
Trong tìm kiếm kinh nghiệm, hàm đánh giá đóng vai trò cực kỳ quan trọng. Chúng ta có xây dựng được hàm đánh giá cho ta sự đánh giá đúng các trạng thái thì tìm kiếm mới hiệu quả. Nếu hàm đánh giá không chính xác, nó có thể dẫn ta đi chệch hướng và do đó tìm kiếm kém hiệu quả.

Hàm đánh giá được xây dựng tùy thuộc vào vấn đề. Sau đây là một số ví dụ về hàm đánh giá:

- Trong bài toán tìm kiếm đường đi trên bản đồ giao thông, ta có thể lấy độ dài của đường chim bay từ một thành phố tới một thành phố đích làm giá trị của hàm đánh giá.

- Bài toán 8 số. Chúng ta có thể đưa ra hai cách xây dựng hàm đánh giá.

Hàm h_1: Với mỗi trạng thái u thì $h_1(u)$ là số quân không nằm đúng vị trí của nó trong trạng thái đích. Chẳng hạn trạng thái đích ở bên phải hình 2.1, và u là trạng thái ở bên trái hình 2.1, thì $h_1(u) = 4$, vì các quân không đúng vị trí là 3, 8, 6 và 1.



Hình 2.1 Đánh giá trạng thái u.

Hàm h_2: $h_2(u)$ là tổng khoảng cách giữa vị trí của các quân trong trạng thái u và vị trí của nó trong trạng thái đích. ở đây khoảng cách được hiểu là số ít nhất các dịch chuyển theo hàng hoặc cột để đưa một quân tới vị trí của nó trong trạng thái đích. Chẳng hạn với trạng thái u và trạng thái đích như trong hình 2.1, ta có:

$h_2(u) = 2 + 3 + 1 + 3 = 9$

Vì quân 3 cần ít nhất 2 dịch chuyển, quân 8 cần ít nhất 3 dịch chuyển, quân 6 cần ít nhất 1 dịch chuyển và quân 1 cần ít nhất 3 dịch chuyển.

Hai chiến lược tìm kiếm kinh nghiệm quan trọng nhất là tìm kiếm tốt nhất - đầu tiên (best-first search) và tìm kiếm leo đồi (hill-climbing search). Có thể xác định các chiến lược này như sau:

Tìm kiếm tốt nhất đầu tiên = Tìm kiếm theo bề rộng + Hàm đánh giá

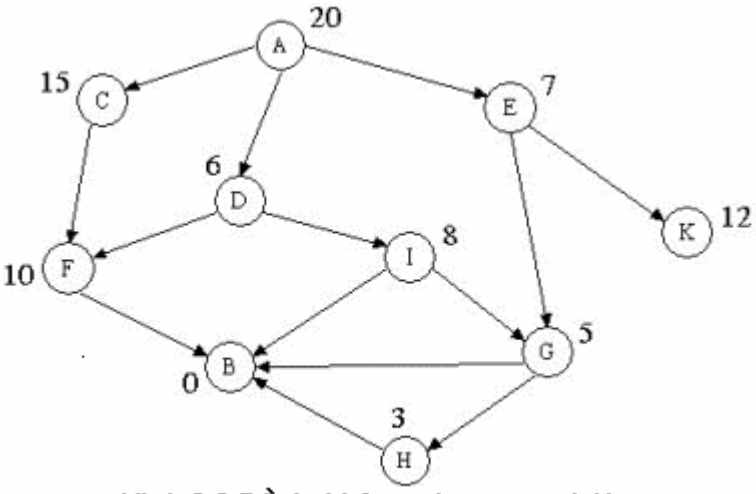
Tìm kiếm leo đồi = Tìm kiếm theo độ sâu + Hàm đánh giá

Chúng ta sẽ lần lượt nghiên cứu các kỹ thuật tìm kiếm này trong các mục sau.

Tìm kiếm tốt nhất - đầu tiên:

Tìm kiếm tốt nhất - đầu tiên (best-first search) là tìm kiếm theo bề rộng được hướng dẫn bởi hàm đánh giá. Nhưng nó khác với tìm kiếm theo bề rộng ở chỗ, trong tìm kiếm theo bề rộng ta lần lượt phát triển tất cả các đỉnh ở mức hiện tại để sinh ra các đỉnh ở mức tiếp theo, còn trong tìm kiếm tốt nhất - đầu tiên ta chọn

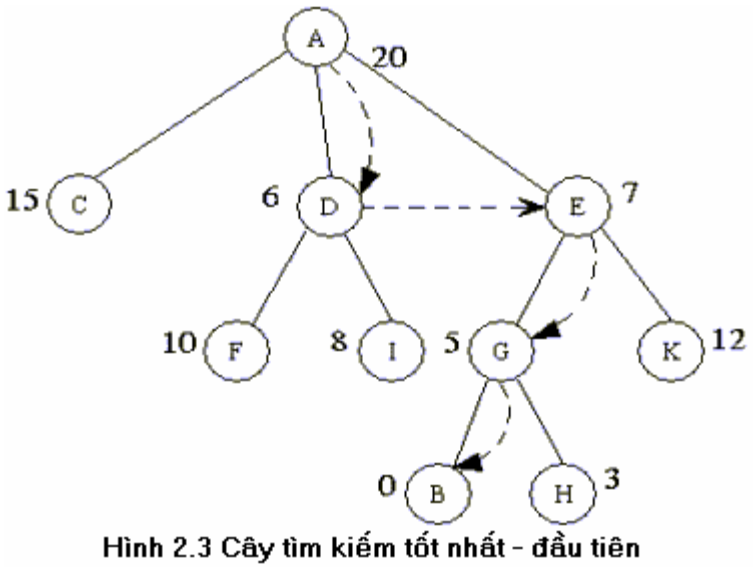
đỉnh để phát triển là đỉnh tốt nhất được xác định bởi hàm đánh giá (tức là đỉnh có giá trị hàm đánh giá là nhỏ nhất), đỉnh này có thể ở mức hiện tại hoặc ở các mức



Hình 2.2 Đồ thị không gian trạng thái

trên.

Ví dụ: Xét không gian trạng thái được biểu diễn bởi đồ thị trong hình 2.2, trong đó trạng thái ban đầu là A, trạng thái kết thúc là B. Giá trị của hàm đánh giá là các số ghi cạnh mỗi đỉnh. Quá trình tìm kiếm tốt nhất - đầu tiên diễn ra như sau: Đầu tiên phát triển đỉnh A sinh ra các đỉnh kề là C, D và E. Trong ba đỉnh này, đỉnh D có giá trị hàm đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra F, I. Trong số các đỉnh chưa được phát triển C, E, F, I thì đỉnh E có giá trị đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra các đỉnh G, K. Trong số các đỉnh chưa được phát triển thì G tốt nhất, phát triển G sinh ra B, H. Đến đây ta đã đạt tới trạng thái kết thúc. Cây tìm kiếm tốt nhất - đầu tiên được biểu diễn trong hình 2.3.



Hình 2.3 Cây tìm kiếm tốt nhất - đầu tiên

Sau đây là thủ tục tìm kiếm tốt nhất - đầu tiên. Trong thủ tục này, chúng ta sử dụng danh sách L để lưu các trạng thái chờ phát triển, danh sách được sắp theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái có giá trị hàm đánh giá nhỏ nhất ở đầu danh sách.

procedure *Best_First_Search***;**

begin

*1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu***;**

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại***; stop};**

*2.2 Loại trạng thái u ở đầu danh sách L***;**

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo thành công***; stop}**

2.4 **for** *mỗi trạng thái v kề u* **do**

*Xen v vào danh sách L sao cho L được sắp theo thứ tự tăng dần của hàm đánh giá***;**

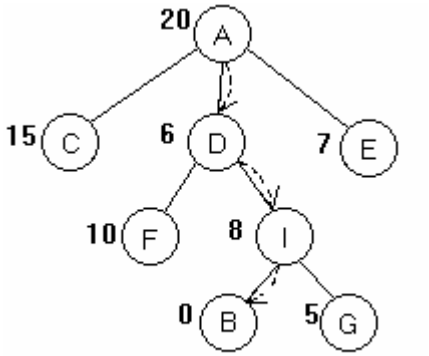
end;

Tìm kiếm leo đồi:

Tìm kiếm leo đồi (hill-climbing search) là tìm kiếm theo độ sâu được hướng dẫn bởi hàm đánh giá. Song khác với tìm kiếm theo độ sâu, khi ta phát triển một

đỉnh u thì bước tiếp theo, ta chọn trong số các đỉnh con của u, đỉnh có nhiều hứa hẹn nhất để phát triển, đỉnh này được xác định bởi hàm đánh giá.

Ví dụ: Ta lại xét đồ thị không gian trạng thái trong hình 2.2. Quá trình tìm kiếm leo đồi được tiến hành như sau. Đầu tiên phát triển đỉnh A sinh ra các đỉnh con C, D, E. Trong các đỉnh này chọn D để phát triển, và nó sinh ra các đỉnh con



Hình 2.4 Cây tìm kiếm leo đồi

B, G. Quá trình tìm kiếm kết thúc. Cây tìm kiếm leo đồi được cho trong hình 2.4.

Trong thủ tục tìm kiếm leo đồi được trình bày dưới đây, ngoài danh sách L lưu các trạng thái chờ được phát triển, chúng ta sử dụng danh sách L_1 để lưu giữ tạm thời các trạng thái kề trạng thái u, khi ta phát triển u. Danh sách L_1 được sắp xếp theo thứ tự tăng dần của hàm đánh giá, rồi được chuyển vào danh sách L sao trạng thái tốt nhất kề u đứng ở danh sách L.

procedure *Hill_Climbing_Search***;**

begin

*1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu***;**

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại***; stop};**

*2.2 Loại trạng thái u ở đầu danh sách L***;**

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo thành công***; stop};**

2.3 **for** *mỗi trạng thái v kề u do đặt v vào* L_1**;**

2.5 Sắp xếp L_1 *theo thứ tự tăng dần của hàm đánh giá***;**

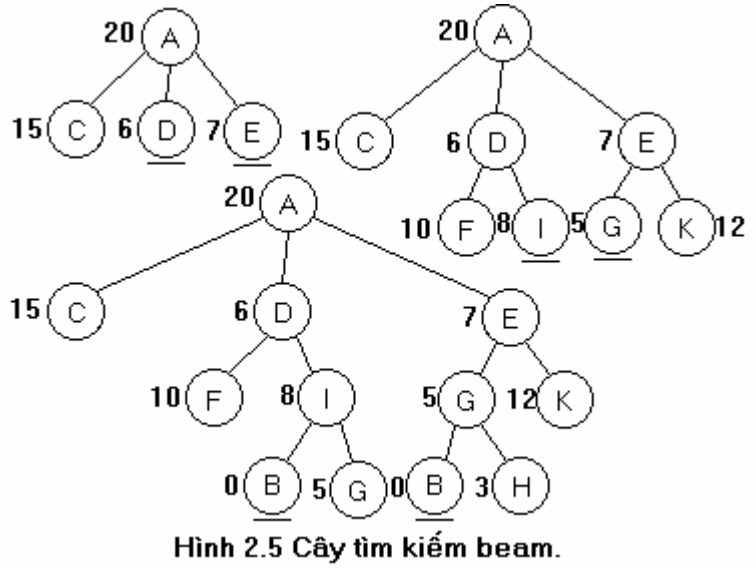
2.6 Chuyển danh sách L_1 *vào đầu danh sách L***;**

end;

Tìm kiếm beam

Tìm kiếm beam (beam search) giống như tìm kiếm theo bề rộng, nó phát triển các đỉnh ở một mức rồi phát triển các đỉnh ở mức tiếp theo. Tuy nhiên, trong tìm kiếm theo bề rộng, ta phát triển tất cả các đỉnh ở một mức, còn trong tìm kiếm beam, ta hạn chế chỉ phát triển k đỉnh tốt nhất (các đỉnh này được xác định bởi hàm đánh giá). Do đó trong tìm kiếm beam, ở bất kỳ mức nào cũng chỉ có nhiều nhất k đỉnh được phát triển, trong khi tìm kiếm theo bề rộng, số đỉnh cần phát triển ở mức d là b^d (b là nhân tố nhánh).

Ví dụ: Chúng ta lại xét đồ thị không gian trạng thái trong hình 2.2. Chọn k = 2. Khi đó cây tìm kiếm beam được cho như hình 2.5. Các đỉnh được gạch dưới là



Hình 2.5 Cây tìm kiếm beam.

các đỉnh được chọn để phát triển ở mỗi mức.

Chương III

Các chiến lược tìm kiếm tối ưu

Vấn đề tìm kiếm tối ưu, một cách tổng quát, có thể phát biểu như sau. Mỗi đối tượng x trong không gian tìm kiếm được gắn với một số đo giá trị của đối tượng đó f(x), mục tiêu của ta là tìm đối tượng có giá trị f(x) lớn nhất (hoặc nhỏ nhất) trong không gian tìm kiếm. Hàm f(x) được gọi là hàm mục tiêu. Trong chương này chúng ta sẽ nghiên cứu các thuật toán tìm kiếm sau:

- Các kỹ thuật tìm đường đi ngắn nhất trong không gian trạng thái: Thuật toán A*, thuật toán nhánh_và_cận.
- Các kỹ thuật tìm kiếm đối tượng tốt nhất: Tìm kiếm leo đồi, tìm kiếm gradient, tìm kiếm mô phỏng luyện kim.
- Tìm kiếm bắt chước sự tiến hóa: thuật toán di truyền.

1.1 Tìm đường đi ngắn nhất.

Trong các chương trước chúng ta đã nghiên cứu vấn đề tìm kiếm đường đi từ trạng thái ban đầu tới trạng thái kết thúc trong không gian trạng thái. Trong mục này, ta giả sử rằng, giá phải trả để đưa trạng thái a tới trạng thái b (bởi một toán tử nào đó) là một số $k(a,b) \geq 0$, ta sẽ gọi số này là độ dài cung (a,b) hoặc giá trị của cung (a,b) trong đồ thị không gian trạng thái. Độ dài của các cung được xác định tùy thuộc vào vấn đề. Chẳng hạn, trong bài toán tìm đường đi trong bản đồ giao thông, giá của cung (a,b) chính là độ dài của đường nối thành phố a với thành phố

b. Độ dài đường đí được xác định là tổng độ dài của các cung trên đường đi. Vấn đề của chúng ta trong mục này, tìm đường đi ngắn nhất từ trạng thái ban đầu tới trạng thái đích. Không gian tìm kiếm ở đây bao gồm tất cả các đường đi từ trạng thái ban đầu tới trạng thái kết thúc, hàm mục tiêu được xác định ở đây là độ dài của đường đi.

Chúng ta có thể giải quyết vấn đề đặt ra bằng cách tìm tất cả các đường đi có thể có từ trạng thái ban đầu tới trạng thái đích (chẳng hạn, sử sụng các ký thuật tìm kiếm mù), sau đó so sánh độ dài của chúng, ta sẽ tìm ra đường đi ngắn nhất. Thủ tục tìm kiếm này thường được gọi là thủ tục bảo tàng Anh Quốc (British Museum Procedure). Trong thực tế, kỹ thuật này không thể áp dụng được, vì cây tìm kiếm thường rất lớn, việc tìm ra tất cả các đường đi có thể có đòi hỏi rất nhiều thời gian. Do đó chỉ có một cách tăng hiệu quả tìm kiếm là sử dụng các hàm đánh giá đề hướng dẫn sử tìm kiếm. Các phương pháp tìm kiếm đường đi ngắn nhất mà chúng ta sẽ trình bày đều là các phương pháp tìm kiếm heuristic.

Giả sử u là một ***trạng thái đạt tới*** (có dường đi từ trạng thái ban đầu u_0 tới u). Ta xác định hai hàm đánh giá sau:

- g(u) là đánh giá độ dài đường đi ngắn nhất từ u_0 tới u (Đường đi từ u_0 tới trạng thái u không phải là trạng thái đích được gọi là ***đường đi một phần***, để phân biệt với ***đường đi đầy đủ***, là đường đi từ u_0 tới trạng thái đích).

- h(u) là đánh giá độ dài đường đi ngắn nhất từ u tới trạng thái đích.

Hàm h(u) được gọi là ***chấp nhận được*** (hoặc đánh giá thấp***)*** nếu với mọi trạng thái u, h(u) ≤ độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đích. Chẳng hạn trong bài toán tìm đường đi ngắn nhất trên bản đồ giao thông, ta có thể xác định h(u) là độ dài đường chim bay từ u tới đích.

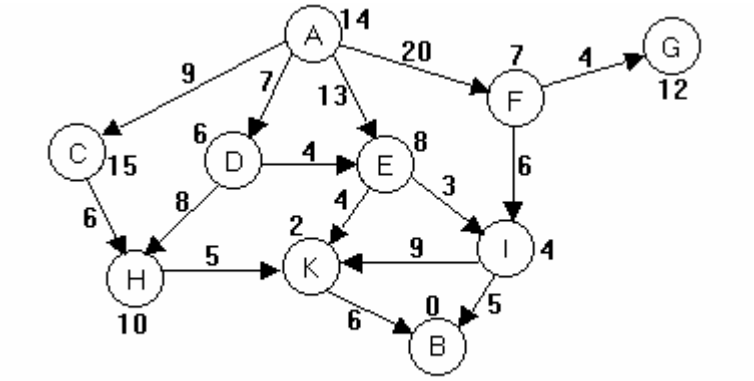
Ta có thể sử dụng kỹ thuật tìm kiếm leo đồi với hàm đánh giá h(u). Tất nhiên phương pháp này chỉ cho phép ta tìm được đường đi tương đối tốt, chưa chắc đã là đường đi tối ưu.

Ta cũng có thể sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u). Phương pháp này sẽ tìm ra đường đi ngắn nhất, tuy nhiên nó có thể kém hiệu quả.

Để tăng hiệu quả tìm kiếm, ta sử dụng hàm đánh giá mới :

f(u) = g(u) + h(u)

Tức là, f(u) là đánh giá độ dài đường đi ngắn nhất qua u từ trạng thái ban đầu tới trạng thái kết thúc.



Hình 3.1 Đồ thị không gian trạng thái với hàm đánh giá.

*1.1.1 Thuật toán A**

Thuật toán A* là thuật toán sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá f(u).

Để thấy được thuật toán A* làm việc như thế nào, ta xét đồ thị không gian trạng thái trong hình 3.1. Trong đó, trạng thái ban đầu là trạng thái A, trạng thái đích là B, các số ghi cạnh các cung là độ dài đường đi, các số cạnh các đỉnh là giá

trị của hàm h.Đầu tiên, phát triển đỉnh A sinh ra các đỉnh con C, D, E và F. Tính giá trị của hàm f tại các đỉnh này ta có:

g(C) = 9, f(C) = 9 + 15 = 24, g(D) = 7, f(D) = 7 + 6 = 13,

g(E) = 13, f(E) = 13 + 8 = 21, g(F) = 20, f(F) = 20 +7 = 27

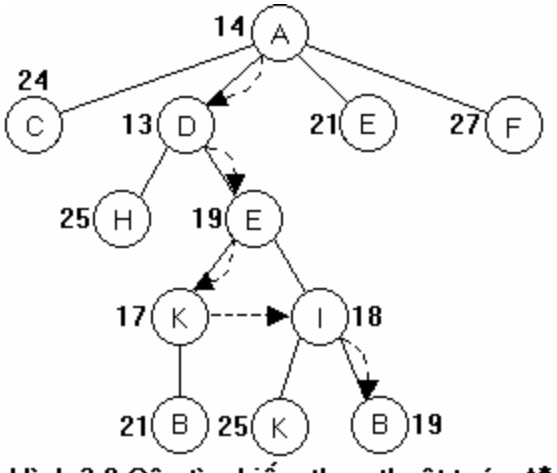
Như vậy đỉnh tốt nhất là D (vì f(D) = 13 là nhỏ nhất). Phát triển D, ta nhận được các đỉnh con H và E. Ta đánh giá H và E (mới):

g(H) = g(D) + Độ dài cung (D, H) = 7 + 8 = 15, f(H) = 15 + 10 = 25.

Đường đi tới E qua D có độ dài:

g(E) = g(D) + Độ dài cung (D, E) = 7 + 4 = 11.

Vậy đỉnh E mới có đánh giá là f(E) = g(E) + h(E) = 11 + 8 = 19. Trong số các đỉnh cho phát triển, thì đỉnh E với đánh giá f(E) = 19 là đỉnh tốt nhất. Phát triển đỉnh này, ta nhận được các đỉnh con của nó là K và I. Chúng ta tiếp tục quá trình trên cho tới khi đỉnh được chọn để phát triển là đỉnh kết thúc B, độ dài đường đi ngắn nhất tới B là g(B) = 19. Quá trình tìm kiếm trên được mô tả bởi cây tìm



Hình 3.2 Cây tìm kiếm theo thuật toán A*

kiếm trong hình 3.2, trong đó các số cạnh các đỉnh là các giá trị của hàm đánh giá f(u).

procedure *A**;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại*; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái đích* **then**

{*thông báo thành công*; **stop**}

2.4 **for** *mỗi trạng thái v kề u* **do**

{$g(v) \leftarrow g(u) + k(u,v)$;

$f(v) \leftarrow g(v) + h(v)$;

Đặt v vào danh sách L;}

2.5 Sắp xếp L theo thứ tự tăng dần của hàm f sao cho

trạng thái có giá trị của hàm f nhỏ nhất

ở đầu danh sách;

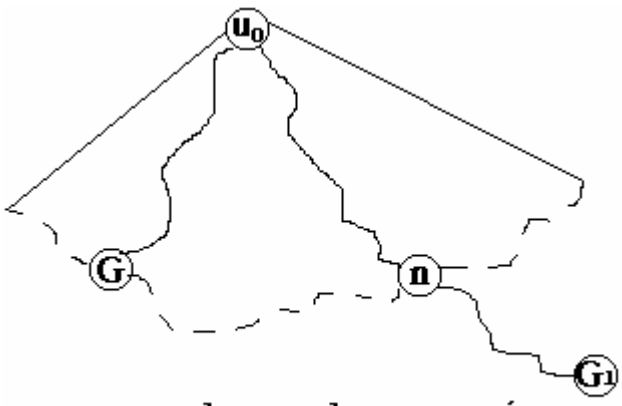
end;

Chúng ta đưa ra một số nhận xét về thuật toán A*.

 Người ta chứng minh được rằng, nếu hàm đánh giá h(u) là đánh giá thấp nhất (trường hợp đặc biệt, h(u) = 0 với mọi trạng thái u) thì thuật toán A* là thuật toán ***tối ưu,*** tức là nghiệm mà nó tìm ra là nghiệm tối ưu. Ngoài ra, nếu độ dài của các cung không nhỏ hơn một số dương δ nào đó thì thuật toán A* là thuật toán ***đầy đủ*** theo nghĩa rằng, nó luôn dừng và tìm ra nghiệm.

Chúng ta chứng minh tính tối ưu của thuật toán A*.

Giả sử thuật toán dừng lại ở đỉnh kết thúc G với độ dài đường đi từ trạng thái ban đầu u_0 tới G là g(G). Vì G là đỉnh kết thúc, ta có h(G) = 0 và f(G) = g(G) +



Hình 3.3 Đỉnh lá n của cây tìm kiếm nằm trên đường đi tối ưu.

h(G) = g(G). Giả sử nghiệm tối ưu là đường đi từ u_0 tới đỉnh kết thúc G_1 với độ dài l. Giả sử đường đi này "thoát ra" khỏi cây tìm kiếm tại đỉnh lá n (Xem hình 3.3). Có thể xẩy ra hai khả năng: n trùng với G_1 hoặc không. Nếu n là G_1 thì vì G được chọn để phát triển trước G_1, nên $f(G) \le f(G_1)$, do đó $g(G) \le g(G_1) = l$. Nếu $n \ne G_1$ thì do h(u) là hàm đánh giá thấp, nên $f(n) = g(n) + h(n) \le l$. Mặt khác, cũng do G được chọn để phát triển trước n, nên $f(G) \le f(n)$, do đó, $g(G) \le l$. Như vậy, ta đã

chứng minh được rằng độ dài của đường đi mà thuật toán tìm ra g(G) không dài hơn độ dài l của đường đi tối ưu. Vậy nó là độ dài đường đi tối ưu.

 Trong trường hợp hàm đánh giá h(u) = 0 với mọi u, thuật toán A* chính là thuật toán tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u) mà ta đã nói đến.

 Thuật toán A* đã được chứng tỏ là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho vấn đề tìm kiếm đường đi ngắn nhất.

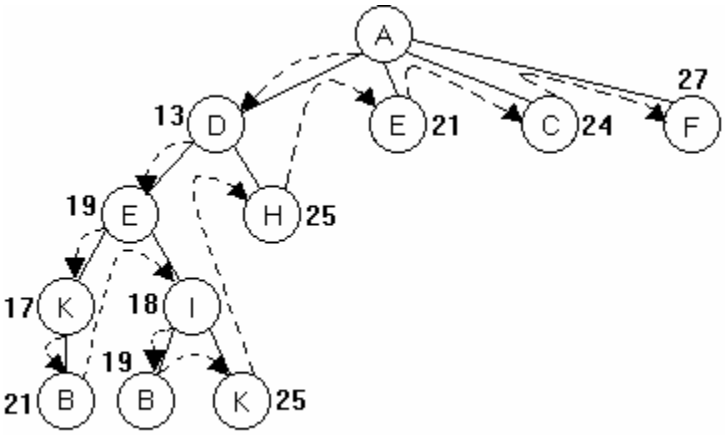
1.1.2 Thuật toán tìm kiếm nhánh-và-cận.

Thuật toán nhánh_và_cận là thuật toán sử dụng tìm kiếm leo đồi với hàm đánh giá f(u).

Trong thuật toán này, tại mỗi bước khi phát triển trạng thái u, thì ta sẽ chọn trạng thái tốt nhất v (f(v) nhỏ nhất) trong số các trạng thái kề u đề phát triển ở bước sau. Đi xuống cho tới khi gặp trạng thái v là đích, hoặc gặp trạng thái v không có đỉnh kề, hoặc gặp trạng thái v mà f(v) lớn hơn độ dài đường đi tối ưu tạm thời, tức là đường đi đầy đủ ngắn nhất trong số các đường đi đầy đủ mà ta đã tìm ra. Trong các trường hợp này, ta không phát triển đỉnh v nữa, hay nói cách khác, ta cắt đi các nhánh cây xuất phát từ v, và quay lên cha của v đề tiếp tục đi xuống trạng thái tốt nhất trong các trạng thái còn lại chưa được phát triển.

Ví dụ: Chúng ta lại xét không gian trạng thái trong hình 3.1. Phát triển đỉnh A, ta nhận được các đỉnh con C, D, E và F, f(C) = 24, f(D) = 13, f(E) = 21, f(F) = 27. Trong số này D là tốt nhất, phát triển D, sinh ra các đỉnh con H và E, f(H) = 25, f(E) = 19. Đi xuống phát triển E, sinh ra các đỉnh con là K và I, f(K) = 17, f(I) = 18. Đi xuống phát triển K sinh ra đỉnh B với f(B) = g(B) = 21. Đi xuống B, vì B là đỉnh đích, vậy ta tìm được đường đi tối ưu tạm thời với độ dài 21. Từ B quay lên K, rồi từ K quay lên cha nó là E. Từ E đi xuống J, f(J) = 18 nhỏ hơn độ dài đường đi tạm thời (là 21). Phát triển I sinh ra các con K và B, f(K) = 25, f(B) =

g(B) = 19. Đi xuống đỉnh B, vì đỉnh B là đích ta tìm được đường đi đầy đủ mới với độ dài là 19 nhỏ hơn độ dài đường đi tối ưu tạm thời cũ (21). Vậy độ dài đường đi tối ưu tạm thời bây giờ là 19. Bây giờ từ B ta lại quay lên các đỉnh còn lại chưa được phát triển. Song các đỉnh này đều có giá trị hàm đánh giá lớn hơn 19, do đó không có đỉnh nào được phát triển nữa. Như vậy, ta tìm được đường đi tối ưu với độ dài 19. Cây tìm kiếm được biểu diễn trong hình 3.4.



Hình 3.4 Cây tìm kiếm nhánh_và_cận.

Thuật toán nhánh_và_cận sẽ được biểu diễn bởi thủ tục Branch_and_Bound. Trong thủ tục này, biến cost được dùng để lưu độ dài đường đi ngắn nhất. Giá trị ban đầu của cost là số đủ lớn, hoặc độ dài của một đường đi đầy đủ mà ta đã biết.

procedure *Branch_and_Bound*;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

Gán giá trị ban đầu cho cost;

2. **loop do**

2.1 **if** *L rỗng* **then stop;**

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái kết thúc* **then**

if $g(u) \le y$ **then** {$y \leftarrow g(y)$; *Quay lại 2.1*};

2.4 **if** $f(u) > y$ **then** *Quay lại 2.1*;

2.5 **for** *mỗi trạng thái v kề u* **do**

{$g(v) \leftarrow g(u) + k(u,v)$;

$f(v) \leftarrow g(v) + h(v)$;

Đặt v vào danh sách L_1};

2.6 Sắp xếp L_1 *theo thứ tự tăng của hàm f*;

2.7 Chuyển L_1 *vào đầu danh sách L sao cho trạng thái*

ở đầu L_1 *trở thành ở đầu L*;

end;

Người ta chứng minh được rằng, thuật toán nhánh_và_cận cũng là thuật toán đầy đủ và tối ưu nếu hàm đánh giá h(u) là đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó.

1.2 Tìm đối tượng tốt nhất

Trong mục này chúng ta sẽ xét vấn đề tìm kiếm sau. Trên không gian tìm kiếm U được xác định hàm giá (hàm mục tiêu) cost, ứng với mỗi đối tượng $x \in U$ với một giá trị số cost(x), số này được gọi là giá trị của x. Chúng ta cần tìm một

đối tượng mà tại đó hàm giá trị lớn nhất, ta gọi đối tượng đó là ***đối tượng tốt nhất***. Giả sử không gian tìm kiếm có cấu trúc cho phép ta xác định được khái niệm lân cận của mỗi đối tượng. Chẳng hạn, U là không gian trạng thái thì lân cận của trạng thái u gồm tất cả các trạng thái v kề u; nếu U là không gian các vectơ thực n-chiều thì lân cận của vectơ $x = (x_1, x_2, ... x_n)$ gồm tất cả các vectơ ở gần x theo khoảng cách Ơcơlit thông thường.

Trong mục này, ta sẽ xét kỹ thuật tìm kiếm leo đồi để tìm đối tượng tốt nhất. Sau đó ta sẽ xét kỹ thuật tìm kiếm gradient (gradient search). Đó là kỹ thuật leo đồi áp dụng cho không gian tìm kiếm là không gian các vectơ thực n-chiều và hàm giá là là hàm khả vi liên tục. Cuối cùng ta sẽ nghiên cứu kỹ thuật tìm kiếm mô phỏng luyện kim(simulated annealing).

1.2.1 Tìm kiếm leo đồi

Kỹ thuật tìm kiếm leo đồi để tìm kiếm đối tượng tốt nhất hoàn toàn giống như kỹ thuật tìm kiếm leo đồi để tìm trạng thái kết thúc đã xét trong mục 2.3. Chỉ khác là trong thuật toán leo đồi ở mục 2.3, từ một trạng thái ta "leo lên" trạng thái kề tốt nhất (được xác định bởi hàm giá), tiếp tục cho tới khi đạt tới trạng thái đích; nếu chưa đạt tới trạng thái đích mà không leo lên được nữa, thì ta tiếp tục "tụt xuống" trạng thái trước nó, rồi lại leo lên trạng thái tốt nhất còn lại. Còn ở đây, từ một đỉnh u ta chỉ leo lên đỉnh tốt nhất v (được xác định bởi hàm giá cost) trong lân cận u nếu đỉnh này "cao hơn" đỉnh u, tức là cost(v) > cost(u). Quá trình tìm kiếm sẽ dừng lại ngay khi ta không leo lên đỉnh cao hơn được nữa.

Trong thủ tục leo đồi dưới đây, biến u lưu đỉnh hiện thời, biến v lưu đỉnh tốt nhất (cost(v) nhỏ nhất) trong các đỉnh ở lân cận u. Khi thuật toán dừng, biến u sẽ lưu trong đối tượng tìm được.

procedure *Hill_Climbing***;**

begin

*1. u ← một đối tượng ban đầu nào đó***;**

2. **if** *cost(v) > cost(u)* **then** *u ← v* **else stop;**

end;

Tối ưu địa phương và tối ưu toàn cục

Rõ ràng là, khi thuật toán leo đồi dừng lại tại đối tương u*, thì giá của nó cost(u*) lớn hơn giá của tất cả các đối tượng nằm trong lân cận của tất cả các đối tượng trên đường đi từ đối tượng ban đầu tới trạng thái u*. Do đó nghiệm u* mà thuật toán leo đồi tìm được là ***tối ưu địa phương***. Cần nhấn mạnh rằng không có gì đảm bảo nghiệm đó là ***tối ưu toàn cục*** theo nghĩa là cost(u*) là lớn nhất trên toàn bộ không gian tìm kiếm.

Để nhận được nghiệm tốt hơn bằng thuật toán leo đồi, ta có thể áp dụng lặp lại nhiều lần thủ tục leo đồi xuất phát từ một dãy các đối tượng ban đầu được chọn ngẫu nhiên và lưu lại nghiệm tốt nhất qua mỗi lần lặp. Nếu số lần lặp đủ lớn thì ta có thể tìm được nghiệm tối ưu.

Kết quả của thuật toán leo đồi phụ thuộc rất nhiều vào hình dáng của "mặt cong" của hàm giá. Nếu mặt cong chỉ có một số ít cực đại địa phương, thì kỹ thuật leo đồi sẽ tìm ra rất nhanh cực đại toàn cục. Song có những vấn đề mà mặt cong của hàm giá tựa như lông nhím vậy, khi đó sử dụng kỹ thuật leo đồi đòi hỏi rất nhiều thời gian.

1.2.2 Tìm kiếm gradient

Tìm kiếm gradient là kỹ thuật tìm kiếm leo đồi để tìm giá trị lớn nhất (hoặc nhỏ nhất) của hàm khả vi liên tục f(x) trong không gian các vectơ thực n-chiều.

$$\nabla f = \left(\frac{\partial f}{\partial \mathrm{x1}}, \frac{\partial f}{\partial \mathrm{x2}}, ..., \frac{\partial f}{\partial \mathrm{xn}} \right)$$

Như ta đã biết, trong lân cận đủ nhỏ của điểm $x = (x_1,...,x_n)$, thì hàm f tăng nhanh nhất theo hướng của vectơ gradient:

Do đó tư tưởng của tìm kiếm gradient là từ một điểm ta đi tới điểm ở lân cận nó theo hướng của vectơ gradient.

procedure *Gradient_Search***;**

begin

*x ← điểm xuất phát nào đó***;**

repeat

$x \leftarrow x + \alpha \nabla f(x)$**;**

until $|\nabla f| < \varepsilon$**;**

end;

Trong thủ tục trên, α là hằng số dương nhỏ nhất xác định tỉ lệ của các bước, còn ε là hằng số dương nhỏ xác định tiêu chuẩn dừng. Bằng cách lấy các bước đủ nhỏ theo hướng của vectơ gradient chúng ta sẽ tìm được điểm cực đại địa phương, đó là điểm mà tại đó $\nabla f = 0$, hoặc tìm được điểm rất gần với cực đại địa phương.

1.2.3 Tìm kiếm mô phỏng luyện kim:

Như đã nhấn mạnh ở trên, tìm kiếm leo đồi không đảm bảo cho ta tìm được nghiệm tối ưu toàn cục. Để cho nghiệm tìm được gần với tối ưu toàn cục, ta áp dụng kỹ thuật leo đồi lặp xuất phát từ các điểm được lựa chọn ngẫu nhiên. Bây giờ thay cho việc luôn luôn "leo lên đồi" xuất phát từ các điểm khác nhau, ta thực hiện

một số bước "tụt xuống" nhằm thoát ra khỏi các điểm cực đại địa phương. Đó chính là tư tưởng của kỹ thuật tìm kiếm mô phỏng luyện kim.

Trong tìm kiếm leo đồi, khi ở một trạng thái u ta luôn luôn đi tới trạng thái tốt nhất trong lân cận nó. Còn bây giờ, trong tìm kiếm mô phỏng luyện kim, ta chọn ngẫu nhiên một trạng thái v trong lân cận u. Nếu trạng thái v được chọn tốt hơn u (cost(v) > cost(u)) thì ta đi tới v, còn nếu không ta chỉ đi tới v với một xác suất nào đó. Xác suất này giảm theo hàm mũ của "độ xấu" của trạng thái v. Xác suất này còn phụ thuộc vào tham số nhiệt độ T. Nhiệt độ T càng cao thì bước đi tới trạng thái xấu càng có khả năng được thực hiện. Trong quá trình tìm kiếm, tham số nhiệt độ T giảm dần tới không. Khi T gần không, thuật toán hoạt động gần giống như leo đồi, hầu như nó không thực hiện bước tụt xuống. Cụ thể ta xác định xác suất đi tới trạng thái xấu v từ u là $e^{\Delta/T}$, ở đây Δ = cost(v) - cost(u).

Sau đây là thủ tục mô phỏng luyện kim.

procedure *Simulated_Anneaning***;**

begin

*t ← 0***;**

*u ← trạng thái ban đầu nào đó***;**

*T ← nhiệt độ ban đầu***;**

repeat

*v ← trạng thái được chọn nhẫu nhiên trong lân cận u***;**

if *cost(v) > cost(u)* **then** *u ← v*

else $u \leftarrow v$ *với xác suất* $e^{\Delta T}$**;**

$T \leftarrow g(T, t)$**;**

$t \leftarrow t + 1$**;**

until *T đủ nhỏ*

end;

Trong thủ tục trên, hàm g(T, t) thỏa mãn điều kiện g(T, t) < T với mọi t, nó xác định tốc độ giảm của nhiệt độ T. Người ta chứng minh được rằng, nếu nhiêt độ T giảm đủ chậm, thì thuật toán sẽ tìm được nghiệm tối ưu toàn cục. Thuật toán mô phỏng luyện kim đã được áp dụng thành công cho các bài toán tối ưu cỡ lớn.

1.3 Tìm kiếm mô phỏng sự tiến hóa. Thuật toán di truyền

Thuật toán di truyền (TTDT) là thuật toán bắt chước sự chọn lọc tự nhiên và di truyền. Trong tự nhiên, các cá thể khỏe, có khả năng thích nghi tốt với môi trường sẽ được tái sinh và nhân bản ở các thế hệ sau. Mỗi cá thể có cấu trúc gien đặc trưng cho phẩm chất của cá thể đó. Trong quá trình sinh sản, các cá thể con có thể thừa hưởng các phẩm chất của cả cha và mẹ, cấu trúc gien của nó mang một phần cấu trúc gien của cha và mẹ. Ngoài ra, trong quá trình tiến hóa, có thể xảy ra hiện tượng đột biến, cấu trúc gien của cá thể con có thể chứa các gien mà cả cha và mẹ đều không có.

Trong TTDT, mỗi cá thể được mã hóa bởi một cấu trúc dữ liệu mô tả cấu trúc gien của cá thể đó, ta sẽ gọi nó là ***nhiễm sắc thể (chroniosome).*** Mỗi nhiễm sắc thể được tạo thành từ các đơn vị được gọi là gien. Chẳng hạn, trong các TTDT cổ điển, các nhiễm sắc thể là các chuỗi nhị phân, tức là mỗi cá thể được biểu diễn bởi một chuỗi nhị phân.

TTDT sẽ làm việc trên các quần thể gồm nhiều cá thể. Một quần thể ứng với một giai đoạn phát triển sẽ được gọi là một ***thế hệ.*** Từ thế hệ ban đầu được tạo ra, TTDT bắt chước chọn lọc tự nhiên và di truyền để biến đổi các thế hệ. TTDT sử dụng các toán tử cơ bản sau đây để biến đổi các thế hệ.

- ***Toán tử tái sinh (reproduction)*** (còn được gọi là ***toán tử chọn lọc (selection)***). Các cá thể tốt được chọn lọc để đưa vào thế hệ sau. Sự lựa chọn này được thực hiện dựa vào độ thích nghi với môi trường của mỗi cá thể. Ta sẽ gọi hàm ứng mỗi cá thể với độ thích nghi của nó là ***hàm thích nghi (fitness function).***

- ***Toán tử lai ghép (crossover).*** Hai cá thể cha và mẹ trao đổi các gien để tạo ra hai cá thể con.

- ***Toán tử đột biến (mutation).*** Một cá thể thay đổi một số gien để tạo thành cá thể mới.

Tất cả các toán tử trên khi thực hiện đều mang tính ngẫu nhiên. Cấu trúc cơ bản của TTDT là như sau:

procedure *Genetic_Algorithm***;**

begin

t ← *0***;**

*Khởi tạo thế hệ ban đầu P(t)***;**

*Đánh giá P(t) (theo hàm thích nghi)***;**

repeat

t ← *t + 1***;**

Sinh ra thế hệ mới P(t) từ P(t-1) bởi

- *Chọn lọc*
- *Lai ghép*
- *Đột biến;*

*Đánh giá P(t)***;**

until *điều kiện kết thúc được thỏa mãn***;**

end;

Trong thủ tục trên, điều kiện kết thúc vòng lặp có thể là một số thế hệ đủ lớn nào đó, hoặc độ thích nghi của các cá thể tốt nhất trong các thế hệ kế tiếp nhau khác nhau không đáng kể. Khi thuật toán dừng, cá thể tốt nhất trong thế hệ cuối cùng được chọn làm nghiệm cần tìm.

Bây giờ ta sẽ xét chi tiết hơn toán tử chọn lọc và các toán tử di truyền (lai ghép, đột biến) trong các TTDT cổ điển.

1. Chọn lọc: Việc chọn lọc các cá thể từ một quần thể dựa trên độ thích nghi của mỗi cá thể. Các cá thể có độ thích nghi cao có nhiều khả năng được chọn. Cần nhấn mạnh rằng, hàm thích nghi chỉ cần là ***một hàm thực dương***, nó có thể không tuyến tính, không liên tục, không khả vi. Quá trình chọn lọc được thực hiện theo kỹ thuật quay bánh xe.

Giả sử thế hệ hiện thời P(t) gồm có n cá thể $\{x_1,..,x_n\}$. Số n được gọi là cỡ của quần thể. Với mỗi cá thể x_i, ta tính độ thích nghi của nó $f(x_i)$. Tính tổng các độ

$$F = \sum_{i=1}^{n} f(xi)$$

thích nghi của tất cả các cá thể trong quần thể:

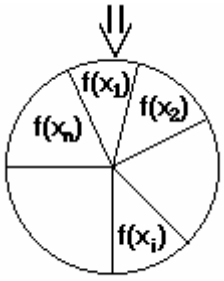
Mỗi lần chọn lọc, ta thực hiện hai bước sau:

- Sinh ra một số thực ngẫu nhiên q trong khoảng (0, F);

$$\sum_{i=1}^{k} f(xi) \geq 4$$

- x_k là cá thể được chọn, nếu k là số nhỏ nhất sao cho

Việc chọn lọc theo hai bước trên có thể minh họa như sau: Ta có một bánh xe được chia thành n phần, mỗi phần ứng với độ thích nghi của một cá thể (hình 3.5). Một mũi tên chỉ vào bánh xe. Quay bánh xe, khi bánh xe dừng, mũi tên chỉ vào phần nào, cá thể ứng với phần đó được chọn.



Hình 3.5 Kỹ thuật quay bánh xe.

Rõ ràng là với cách chọn này, các cá thể có thể có độ thích nghi càng cao càng có khả năng được chọn. Các cá thể có độ thích nghi cao có thể có một hay nhiều bản sao, các cá thể có độ thích nghi thấp có thể không có mặt ở thế hệ sau (nó bị chết đi).

2. Lai ghép: Trên cá thể được chọn lọc, ta tiến hành toán tử lai ghép. Đầu tiên ta cần đưa ra xác suất lai ghép p_c. xác suất này cho ta hy vọng có p_c.n cá thể được lai ghép (n là cỡ của quần thể).

Với mỗi cá thể ta thực hiện hai bước sau:

- Sinh ra số thực ngẫu nhiên r trong đoạn [0, 1];
- Nếu $r < p_c$ thì cá thể đó được chọn để lai ghép

Từ các cá thể được chọn để lai ghép, người ta cặp đôi chúng một cách ngẫu nhiên. Trong trường hợp các nhiễm sắc thể là các chuỗi nhị phân có độ dài cố định m, ta có thể thực hiện lai ghép như sau: Với mỗi cặp, sinh ra một số nguyên ngẫu nhiên p trên đoạn [0, m -1], p là vị trí điểm ghép. Cặp gồm hai nhiễm sắc thể

$$\mathbf{a = (a_1 , \dots , a_p , a_{p+1} , \dots , a_m)}$$

$$\mathbf{a = (b_1 , \dots , b_p , b_{p+1} , \dots , b_m)}$$

được thay bởi hai con là:

$$\mathbf{a' = (a_1 , \dots , a_p , b_{p+1} , \dots , b_m)}$$

$$\mathbf{b' = (b_1 , \dots , b_p , a_{p+1} , \dots , a_m)}$$

3. Đột biến: Ta thực hiện toán tử đột biến trên các cá thể có được sau quá trình lai ghép. Đột biến là thay đổi trạng thái một số gien nào đó trong nhiễm sắc thể. Mỗi gien chịu đột biến với xác suất p_m. Xác suất đột biến p_m do ta xác định và là xác suất thấp. Sau đây là toán tử đột biến trên các nhiễm sắc thể chuỗi nhị phân.

Với mỗi vị trí i trong nhiễm sắc thể:

$$\mathbf{a = (a_1 , \dots , a_i , \dots , a_m)}$$

Ta sinh ra một số thực nghiệm ngẫu nhiên p_i trong [0,1]. Qua đột biến a được biến thành a’ như sau:

$$\mathbf{a' = (a'_1 , \dots , a'_i , \dots , a'_m)}$$

Trong đó :

$$\begin{cases} \mathbf{a'_i = a_i} & \text{nếu } p_i \geq p_m \\ \mathbf{1 - a_i} & \text{nếu } p_i < p_m \end{cases}$$

Sau quá trình chọn lọc, lai ghép, đột biến, một thế hệ mới được sinh ra. Công việc còn lại của thuật toán di truyền bây giờ chỉ là lặp lại các bước trên.

Ví dụ: Xét bài toán tìm max của hàm $f(x) = x^2$ với x là số nguyên trên đoạn [0,31]. Để sử dụng TTDT, ta mã hoá mỗi số nguyên x trong đoạn [0,31] bởi một số nhị phân độ dài 5, chẳng hạn, chuỗi 11000 là mã của số nguyên 24. Hàm thích nghi được xác định là chính hàm $f(x) = x^2$. Quần thể ban đầu gồm 4 cá thể (cỡ của quần thể là n = 4). Thực hiện quá trình chọn lọc, ta nhận được kết quả trong bảng sau. Trong bảng này, ta thấy cá thể 2 có độ thích nghi cao nhất (576) nên nó được chọn 2 lần, cá thể 3 có độ thích nghi thấp nhất (64) không được chọn lần nào. Mỗi cá thể 1 và 4 được chọn 1 lần.

Bảng kết quả chọn lọc

Số liệu cá thể	Quần thể ban đầu	x	Độ thích nghi $f(x) = x^2$	Số lần được chọn
1	0 1 1 0 1	13	169	1
2	1 1 0 0 0	24	576	2
3	0 1 0 0 0	8	64	0
4	1 0 0 1 1	19	361	1

Thực hiện qúa trình lai ghép với xác suất lai ghép $p_c = 1$, cả 4 cá thể sau chọn lọc đều được lai ghép. Kết quả lai ghép được cho trong bảng sau. Trong bảng

này, chuỗi thứ nhất được lai ghép với chuỗi thứ hai với điểm ghép là 4, hai chuỗi còn lại được lai ghép với nhau với điểm ghép là 2.

Bảng kết quả lai ghép

Quần thể sau chọn lọc	Điểm ghép	Quần thể sau lai ghép	x	Độ thích nghi f(x) = x2
0 1 1 0 | 1	4	0 1 1 0 0	2	144
1 1 0 0 | 0	4	1 1 0 0 1	5	625
1 1 | 0 0 0	2	1 1 0 1 1	7	729
1 0 | 0 1 1	2	1 0 0 0 0	6	256

Để thực hiện quá trình đột biến, ta chọn xác suất đột biến p_m= 0,001, tức là ta hy vọng có 5.4.0,001 = 0,02 bit được đột biến. Thực tế sẽ không có bit nào được đột biến. Như vậy thế hệ mới là quần thể sau lai ghép. Trong thế hệ ban đầu, độ thích nghi cao nhất là 576, độ thích nghi trung bình 292. Trong thế hệ sau, độ thích nghi cao nhất là 729, trung bình là 438. Chỉ qua một thế hệ, các cá thể đã “tốt lên” rất nhiều.

Thuật toán di truyền khác với các thuật toán tối ưu khác ở các điểm sau:

- TTDT chỉ sử dụng hàm thích để hướng dẫn sự tìm kiếm, hàm thích nghi chỉ cần là hàm thực dương. Ngoài ra, nó không đòi hỏi không gian tìm kiếm phải có cấu trúc nào cả.

- TTDT làm việc trên các nhiễm sắc thể là mã của các cá thể cần tìm.

- TTDT tìm kiếm từ một quần thể gồm nhiều cá thể.

- Các toán tử trong TTDT đều mang tính ngẫu nhiên.

Để giải quyết một vấn đề bằng TTDT, chúng ta cần thực hiện các bước sau đây:

- Trước hết ta cần mã hóa các đối tượng cần tìm bởi một cấu trúc dữ liệu nào đó. Chẳng hạn, trong các TTDT cổ điển, như trong ví dụ trên, ta sử dụng mã nhị phân.

- Thiết kế hàm thích nghi. Trong các bài toán tối ưu, hàm thích nghi được xác định dựa vào hàm mục tiêu.

- Trên cơ sở cấu trúc của nhiễm sắc thể, thiết kế các toán tử di truyền (lai ghép, đột biến) cho phù hợp với các vấn đề cần giải quyết.

- Xác định cỡ của quần thể và khởi tạo quần thể ban đầu.

- Xác định xác suất lai ghép pc và xác suất đột biến. Xác suất đột biến cần là xác suất thấp. Người ta (Goldberg, 1989) khuyên rằng nên chọn xác suất lai ghép là 0,6 và xác suất đột biến là 0,03. Tuy nhiên cần qua thử nghiệm để tìm ra các xác suất thích hợp cho vấn đề cần giải quyết.

Nói chung thuật ngữ TTDT là để chỉ TTDT cổ điển, khi mà cấu trúc của các nhiễm sắc thể là các chuỗi nhị phân với các toán tử di truyền đã được mô tả ở trên. Song trong nhiều vấn đề thực tế, thuận tiện hơn, ta có thể biểu diễn nhiễm sắc thể bởi các cấu trúc khác, chẳng hạn vectơ thực, mảng hai chiều, cây,... Tương ứng với cấu trúc của nhiễm sắc thể, có thể có nhiều cách xác định các toán tử di truyền. Quá trình sinh ra thế hệ mới P(t) từ thế hệ cũ P(t - 1) cũng có nhiều cách chọn lựa.

Người ta gọi chung các thuật toán này là thuật toán tiến hóa (evolutionary algorithms) hoặc chương trình tiến hóa (evolution program).

Thuật toán tiến hóa đã được áp dụng trong các vấn đề tối ưu và học máy. Để hiểu biết sâu sắc hơn về thuật toán tiến hoá, bạn đọc có thể tìm đọc [ ], [ ] và [ ] . [ ] và [ ] được xem là các sách hay nhất viết về TTDT. [ ] cho ta cái nhìn tổng quát về sự phát triển gần đây của TTDT.

Chương IV

Tìm kiếm có đối thủ

Nghiên cứu máy tính chơi cờ đã xuất hiện rất sớm. Không lâu sau khi máy tính lập trình được ra đời vào năm 1950, Claude Shannon đã viết chương trình chơi cờ đầu tiên. các nhà nghiên cứu **Trí Tuệ Nhân Tạo** đã nghiên cứu việc chơi cờ, vì rằng máy tính chơi cờ là một bằng chứng rõ ràng về khả năng máy tính có thể làm được các công việc đòi hỏi trí thông minh của con người. Trong chương này chúng ta sẽ xét các vấn đề sau đây:

- Chơi cờ có thể xem như vấn đề tìm kiếm trong không gian trạng thái.
- Chiến lược tìm kiếm nước đi Minimax.
- Phương pháp cắt cụt α-β, một kỹ thuật để tăng hiệu quả của tìm kiếm Minimax.

1.1 Cây trò chơi và tìm kiếm trên cây trò chơi.

Trong chương này chúng ta chỉ quan tâm nghiên cứu các trò chơi có hai người tham gia, chẳng hạn các loại cờ (cờ vua, cờ tướng, cờ ca rô...). Một người chơi được gọi là Trắng, đối thủ của anh ta được gọi là Đen. Mục tiêu của chúng ta là nghiên cứu chiến lược chọn nước đi cho Trắng (Máy tính cầm quân Trắng).

Chúng ta sẽ xét các trò chơi hai người với các đặc điểm sau. Hai người chơi thay phiên nhau đưa ra các nước đi tuân theo các luật đi nào đó, các luật này là như nhau cho cả hai người. Điển hình là cờ vua, trong cờ vua hai người chơi có

thể áp dụng các luật đi con tốt, con xe, ... để đưa ra nước đi. Luật đi con tốt Trắng xe Trắng, ... cũng như luật đi con tốt Đen, xe Đen, ... Một đặc điểm nữa là hai người chơi đều được biết thông tin đầy đủ về các tình thế trong trò chơi (không như trong chơi bài, người chơi không thể biết các người chơi khác còn những con bài gì). Vấn đề chơi cờ có thể xem như vấn đề tìm kiếm nước đi, tại mỗi lần đến lượt mình, người chơi phải tìm trong số rất nhiều nước đi hợp lệ (tuân theo đúng luật đi), một nước đi tốt nhất sao cho qua một dãy nước đi đã thực hiện, anh ta giành phần thắng. Tuy nhiên vấn đề tìm kiếm ở đây sẽ phức tạp hơn vấn đề tìm kiếm mà chúng ta đã xét trong các chương trước, bởi vì ở đây có đối thủ, người chơi không biết được đối thủ của mình sẽ đi nước nào trong tương lai. Sau đây chúng ta sẽ phát biểu chính xác hơn vấn đề tìm kiếm này.

Vấn đề chơi cờ có thể xem như vấn đề tìm kiếm trong không gian trạng thái. Mỗi trạng thái là một tình thế (sự bố trí các quân của hai bên trên bàn cờ).

- Trạng thái ban đầu là sự sắp xếp các quân cờ của hai bên lúc bắt đầu cuộc chơi.

- Các toán tử là các nước đi hợp lệ.

- Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường được xác định bởi một số điều kiện dừng nào đó.

- Một hàm kết cuộc (payoff function) ứng mỗi trạng thái kết thúc với một giá trị nào đó. Chẳng hạn như cờ vua, mỗi trạng thái kết thúc chỉ có thể là thắng, hoặc thua (đối với Trắng) hoặc hòa. Do đó, ta có thễ xác định hàm kết cuộc là hàm nhận giá trị 1 tại các trạng thái kết thúc là thắng (đối với Trắng), -1 tại các trạng thái kết thúc là thua (đối với Trắng) và 0 tại các trạng thái kết thúc hòa. Trong một số trò chơi khác, chẳng hạn trò chơi tính điểm, hàm kết cuộc có thể nhận giá trị nguyên trong khoảng [-k, k] với k là một số nguyên dương nào đó.

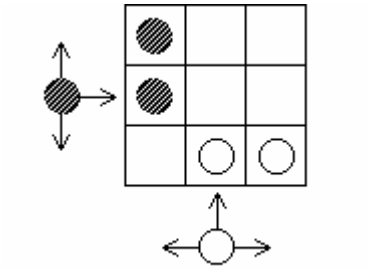
Như vậy vấn đề của Trắng là, tìm một dãy nước đi sao cho xen kẽ với các nước đi của Đen tạo thành một đường đi từ trạng thái ban đầu tới trạng thái kết thúc là thắng cho Trắng.

Để thuận lợi cho việc nghiên cứu các chiến lược chọn nước đi, ta biểu diễn không gian trạng thái trên dưới dạng cây trò chơi.

Cây trò chơi

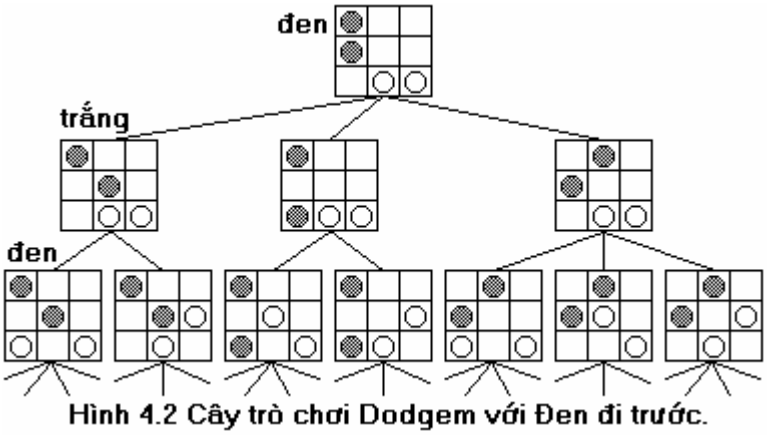
Cây trò chơi được xây dựng như sau. Gốc của cây ứng với trạng thái ban đầu. Ta sẽ gọi đỉnh ứng với trạng thái mà Trắng (Đen) đưa ra nước đi là đỉnh Trắng (Đen). Nếu một đỉnh là Trắng (Đen) ứng với trạng thái u, thì các đỉnh con của nó là tất cả các đỉnh biểu diễn trạng thái v, v nhận được từ u do Trắng (Đen) thực hiện nước đi hợp lệ nào đó. Do đó, trên cùng một mức của cây các đỉnh đều là Trắng hặc đều là Đen, các lá của cây ứng với các trnạg thái kết thúc.

Ví dụ: Xét trò chơi Dodgen (được tạo ra bởi Colin Vout). Có hai quân Trắng và hai quân Đen, ban đầu được xếp vào bàn cờ 3*3 (Hình vẽ). Quân Đen có thể đi tới ô trống ở bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi tới trống ở bên trái, bên phải, ở trên. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa hai quân của mình ra khỏi bàn cờ trước sẽ thắng, hoặc tạo ra tình thế bắt đối phương không đi được cũng sẽ thắng.



Hình 4.1 Trò chơi Dodgem.

Giả sử Đen đi trước, ta có cây trò chơi được biểu diễn như trong hình 4.2.



Hình 4.2 Cây trò chơi Dodgem với Đen đi trước.

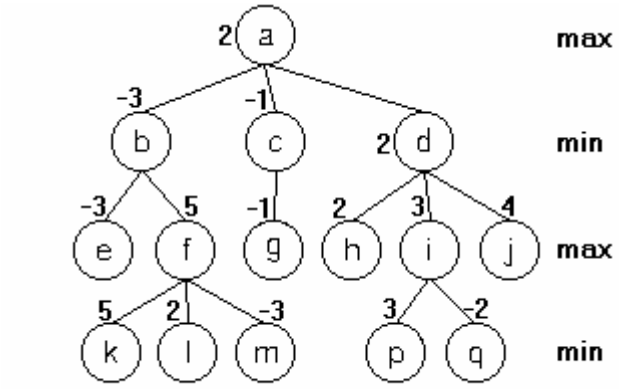
1.2 Chiến lược Minimax

Quá trình chơi cờ là quá trình Trắng và Đen thay phiên nhau đưa ra quyết định, thực hiện một trong số các nước đi hợp lệ. Trên cây trò chơi, quá trình đó sẽ tạo ra đường đi từ gốc tới lá. Giả sử tới một thời điểm nào đó, đường đi đã dẫn tới đỉnh u. Nếu u là đỉnh Trắng (Đen) thì Trắng (Đen) cần chọn đi tới một trong các đỉnh Đen (Trắng) v là con của u. Tại đỉnh Đen (Trắng) v mà Trắng (Đen) vừa chọn, Đen (Trắng) sẽ phải chọn đi tới một trong các đỉnh Trắng (Đen) w là con của v. Quá trình trên sẽ dừng lại khi đạt tới một đỉnh là lá của cây.

Giả sử Trắng cần tìm nước đi tại đỉnh u. Nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con của v là đỉnh tốt nhất (cho Trắng) trong số các đỉnh con của u. Ta cần giả thiết rằng, đến lượt đối thủ chọn nước đi từ v, Đen cũng sẽ chọn nước đi tốt nhất cho anh ta. Như vậy, để chọn nước đi tối ưu cho Trắng tại đỉnh u, ta cần phải xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá (ứng

với các trạng thái kết thúc) là giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất lên gốc u. Giả sử v là đỉnh trong của cây và giá trị các đỉnh con của nó đã được xác định. Khi đó nếu v là đỉnh Trắng thì giá trị của nó được xác định là giá trị lớn nhất trong các giá trị của các đỉnh con. Còn nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các giá trị của các đỉnh con.

Ví dụ: Xét cây trò chơi trong hình 4.3, gốc a là đỉnh Trắng. Giá trị của các đỉnh là số ghi cạnh mỗi đỉnh. Đỉnh i là Trắng, nên giá trị của nó là max(3,-2) = 3,



Hình 4.3 Gán giá trị cho các đỉnh của cây trò chơi.

đỉnh d là đỉnh Đen, nên giá trị của nó là min(2, 3, 4) = 2.

Việc gán giá trị cho các đỉnh được thực hiện bởi các hàm đệ qui MaxVal và MinVal. Hàm MaxVal xác định giá trị cho các đỉnh Trắng, hàm MinVal xác định giá trị cho các đỉnh Đen.

function *MaxVal(u)***;**

begin

if *u là đỉnh kết thúc* **then** *MaxVal(u)* ← *f(u)*

else *MaxVal(u)* ← *max***{***MinVal(v)* **|** *v là đỉnh con của u***}**

end;

function *MinVal(u)***;**

begin

if *u là đỉnh kết thúc* **then** *MinVal(u)* ← *f(u)*

else *MinVal(u)* ← *min***{***MaxVal(v)* | *v là đỉnh con của u***}**

end;

Trong các hàm đệ quy trên, f(u) là giá trị của hàm kết cuộc tại đỉnh kết thúc u. Sau đây là thủ tục chọn nước đi cho trắng tại đỉnh u. Trong thủ tục Minimax(u,v), v là biến lưu lại trạng thái mà Trắng đã chọn đi tới từ u.

procedure *Minimax(u, v)***;**

begin

val ← *-∞***;**

for *mỗi w là đỉnh con của u* **do**

if *val* <= *MinVal(w)* **then**

{*val* ← *MinVal(w)***;** *v* ← *w***}**

end;

Thủ tục chọn nước đi như trên gọi là chiến lược Minimax, bởi vì Trắng đã chọ được nước đi dẫn tới đỉnh con có giá trị là max của các giá trị các đỉnh con, và Đen đáp lại bằng nước đi tới đỉnh có giá trị là min của các giá trị các đỉnh con.

Thuật toán Minimax là thuật toán tìm kiếm theo độ sâu, ở đây ta đã cài đặt thuật toán Minimax bởi các hàm đệ quy. Bạn đọc hãy viết thủ tục không đệ quy thực hiện thuật toán này.

Về mặt lí thuyết, chiến lược Minimax cho phép ta tìm được nước đi tối ưu cho Trắng. Song nó không thực tế, chúng ta sẽ không có đủ thời gian để tính được nước đi tối ưu. Bởi vì thuật toán Minimax đòi hỏi ta phải xem xét toàn bộ các đỉnh của cây trò chơi. Trong các trò chơi hay, cây trò chơi là cực kỳ lớn. Chẳng hạn, đối với cờ vua, chỉ tính đến độ sâu 40, thì cây trò chơi đã có khoảng 10^{120} đỉnh! Nếu cây có độ cao m, và tại mỗi đỉnh có b nước đi thì độ phức tạp về thời gian của thuật toán Minimax là $O(b^m)$.

Để có thể tìm ra nhanh nước đi tốt (không phải là tối ưu) thay cho việc sử dụng hàm kết cuộc và xem xét tất cả các khả năng dẫn tới các trạng thái kết thúc, chúng ta sẽ sử dụng hàm đánh giá và chỉ xem xét một bộ phận của cây trò chơi.

Hàm đánh giá

Hàm đánh giá eval ứng với mỗi trạng thái u của trò chơi với một giá trị số eval(u), giá trị này là sự đánh giá "độ lợi thế" của trạng thái u. Trạng thái u càng thuận lợi cho Trắng thì eval(u) là số dương càng lớn; u càng thuận lợi cho Đen thì eval(u) là số âm càng nhỏ; eval(u) ≈ 0 đối với trạng thái không lợi thế cho ai cả.

Chất lượng của chương trình chơi cờ phụ thuộc rất nhiều vào hàm đánh giá. Nếu hàm đánh giá cho ta sự đánh giá không chính xác về các trạng thái, nó có thể hướng dẫn ta đi tới trạng thái được xem là tốt, nhưng thực tế lại rất bất lợi cho ta. Thiết kế một hàm đánh giá tốt là một việc khó, đòi hỏi ta phải quan tâm đến nhiều

nhân tố: các quân còn lại của hai bên, sự bố trí của các quân đó, ... ở đây có sự mâu thuẫn giữa độ chính xác của hàm đánh giá và thời gian tính của nó. Hàm đánh giá chính xác sẽ đòi hỏi rất nhiều thời gian tính toán, mà người chơi lại bị giới hạn bởi thời gian phải đưa ra nước đi.

Ví dụ 1: Sau đây ta đưa ra một cách xây dựng hàm đánh giá đơn giản cho cờ vua. Mỗi loại quân được gán một giá trị số phù hợp với "sức mạnh" của nó. Chẳng hạn, mỗi tốt Trắng (Đen) được cho 1 (-1), mã hoặc tượng Trắng (Đen) được cho 3 (-3), xe Trắng (Đen) được cho 5 (-5) và hoàng hậu Trắng (Đen) được cho 9 (-9). Lấy tổng giá trị của tất cả các quân trong một trạng thái, ta sẽ được giá trị đánh giá của trạng thái đó. Hàm đánh giá như thế được gọi là hàm tuyến tính có trọng số, vì nó có thể biểu diễn dưới dạng:

$$s_1w_1 + s_2w_2 + \ldots + s_nw_n.$$

Trong đó, w_i là giá trị mỗi loại quân, còn s_i là số quân loại đó. Trong cách đánh giá này, ta đã không tính đến sự bố trí của các quân, các mối tương quan giữa chúng.

Ví dụ 2: Bây giờ ta đưa ra một cách đánh giá các trạng thái trong trò chơi Dodgem. Mỗi quân Trắng ở một vị trí trên bàn cờ được cho một giá trị tương ứng trong bảng bên trái hình 4.4. Còn mỗi quân Đen ở một vị trí sẽ được cho một giá trị tương ứng trong bảng bên phải hình 4.4:

30	35	40
15	20	25
0	5	10

Giá trị quân Trắng.

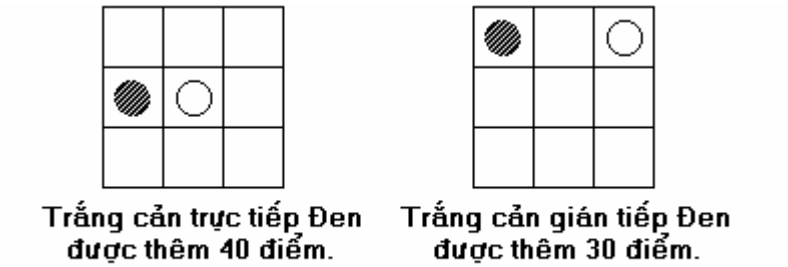
-10	-25	-40
-5	-20	-35
0	-15	-30

Giá trị quân Đen.

Hình 4.4 Đánh giá các quân trong trò chơi Dodgem.

Ngoài ra, nếu quân Trắng cản trực tiếp một quân Đen, nó được thêm 40 điểm, nếu cản gián tiếp nó được thêm 30 điểm (Xem hình 4.5). Tương tự, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, còn cản gián tiếp nó được thêm -30 điểm.

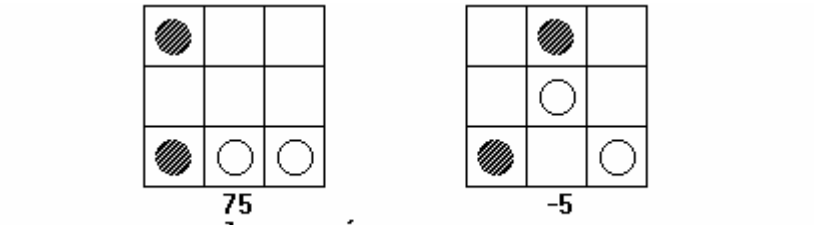
áp dụng các qui tắc trên, ta tính được giá trị của trạng thái ở bên trái hình 4.6 là 75, giá trị của trạng thái bên phải hình vẽ là -5.



Hình 4.5 Đánh giá sự tương quan giữa quân Trắng và Đen.

Trong cánh đánh giá trên, ta đã xét đến vị trí của các quân và mối tương quan giữa các quân.

Một cách đơn giản để hạn chế không gian tìm kiếm là, khi cần xác định nước đi cho Trắng tại u, ta chỉ xem xét cây trò chơi gốc u tới độ cao h nào đó. áp dụng thủ tục Minimax cho cây trò chơi gốc u, độ cao h và sử dụng giá trị của hàm đánh



Hình 4.6 Giá trị của một số trạng thái trong trò chơi Dodgem.

giá cho các lá của cây đó, chúng ta sẽ tìm được nước đi tốt cho Trắng tại u.

1.3 Phương pháp cắt cụt alpha - beta

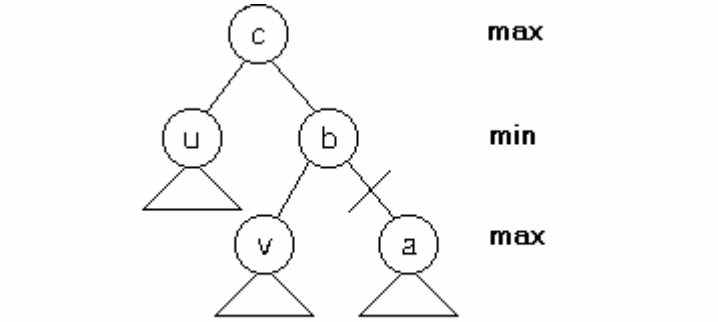
Trong chiến lược tìm kiếm Minimax, để tìm kiếm nước đi tốt cho Trắng tại trạng thái u, cho dù ta hạn chế không gian tìm kiếm trong phạm vi cây trò chơi gốc u với độ cao h, thì số đỉnh của cây trò chơi này cũng còn rất lớn với $h \geq 3$. Chẳng hạn, trong cờ vua, nhân tố nhánh trong cây trò chơi trung bình khoảng 35, thời gian đòi hỏi phải đưa ra nước đi là 150 giây, với thời gian này trên máy tính thông thường chương trình của bạn chỉ có thể xem xét các đỉnh trong độ sâu 3 hoặc 4. Một người chơi cờ trình độ trung bình cũng có thể tính trước được 5, 6 nước hoặc hơn nữa, và do đó chương trình của bạn mới đạt trình độ người mới tập chơi!

Khi đánh giá đỉnh u tới độ sâu h, một thuật toán Minimax đòi hỏi ta phải đánh giá tất cả các đỉnh của cây gốc u tới độ sâu h. Song ta có thể giảm bớt số đỉnh cần phải dánh giá mà vẫn không ảnh hưởng gì đến sự đánh giá đỉnh u. Phương pháp cắt cụt alpha-beta cho phép ta cắt bỏ các nhánh không cần thiết cho sự đánh giá đỉnh u.

Tư tưởng của kỹ thuật cắt cụt alpha-beta là như sau: Nhớ lại rằng, chiến lược tìm kiếm Minimax là chiến lược tìm kiếm theo độ sâu. Giả sử trong quá trình tìm kiếm ta đi xuống đỉnh a là đỉnh Trắng, đỉnh a có người anh em v đã được đánh giá. Giả sử cha của đỉnh a là b và b có người anh em u dã được đánh giá, và giả sử cha của b là c (Xem hình 4.7). Khi đó ta có giá trị đỉnh c (đỉnh Trắng) ít nhất là giá trị của u, giá trị của đỉnh b (đỉnh Đen) nhiều nhất là giá trị v. Do đó, nếu eval(u) > eval(v), ta không cần đi xuống để đánh giá đỉnh a nữa mà vẫn không ảnh hưởng gì dến đánh giá đỉnh c. Hay nói cách khác ta có thể cắt bỏ cây con gốc a. Lập luận tương tự cho trường hợp a là đỉnh Đen, trong trường hợp này nếu eval(u) < eval(v) ta cũng có thể cắt bỏ cây con gốc a.

Để cài đặt kỹ thuật cắt cụt alpha-beta, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá

trị của các đỉnh con đã đánh giá của một đỉnh Trắng, còn tham số β ghi lại giá trị nhỏ nhất trong các đỉnh con đã đánh giá của một đỉnh Đen. Giá trị của α và β sẽ được cập nhật trong quá trình tìm kiếm. α và β được sử dụng như các biến địa phương trong các hàm MaxVal(u, α, β) (hàm xác định giá trị của đỉnh Trắng u) và



Hình 4.7 Cắt bỏ cây con gốc a, nếu eval(u) > eval(v).

Minval(u, α, β) (hàm xác định giá trị của đỉnh Đen u).

function *MaxVal(u, α, β)***;**

begin

if *u là lá của cây hạn chế hoặc u là đỉnh kết thúc*

then *MaxVal* ← *eval(u)*

else for *mỗi đỉnh v là con của u* **do**

{$\alpha \leftarrow max[\alpha, MinVal(v, \alpha, \beta)]$**;**

// Cắt bỏ các cây con từ các đỉnh v còn lại

if $\alpha \geq \beta$ **then exit};**

MaxVal ← α**;**

end;

function *MinVal(u, α, β)***;**

begin

if *u là lá của cây hạn chế hoặc u là đỉnh kết thúc*

then *MinVal $\leftarrow$ eval(u)*

else for *mỗi đỉnh v là con của u* **do**

{$\beta \leftarrow min[\beta, MaxVal(v, \alpha, \beta)]$**;**

// Cắt bỏ các cây con từ các đỉnh v còn lại

if $\alpha \geq \beta$ **then exit};**

MinVal $\leftarrow \beta$**;**

end;

Thuật toán tìm nước đi cho Trắng sử dụng kỹ thuật cắt cụt alpha-beta, được cài đặt bởi thủ tục Alpha_beta(u,v), trong đó v là tham biến ghi lại đỉnh mà Trắng cần đi tới từ u.

procedure *Alpha_beta(u,v)***;**

begin

$\alpha \leftarrow -\infty$**;**

$\beta \leftarrow \infty$**;**

for *mỗi đỉnh w là con của u* **do**

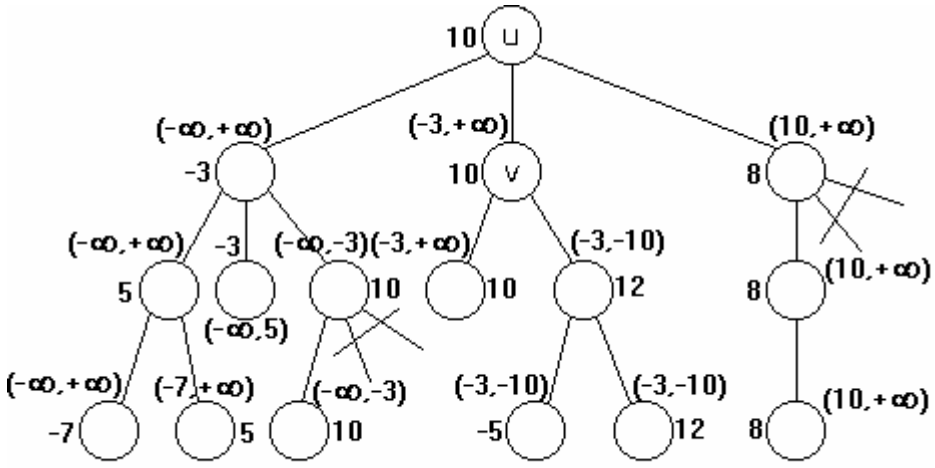
if $\alpha \le MinVal(w, \alpha, \beta)$ **then**

{$\alpha \leftarrow MinVal(w, \alpha, \beta)$**;**

$v \leftarrow w$**;}**

end;

Ví dụ. Xét cây trò chơi gốc u (đỉnh Trắng) giới hạn bởi độ cao h = 3 (hình 4.8). Số ghi cạnh các lá là giá trị của hàm đánh giá. áp dụng chiến lược Minimax và kỹ thuật cắt cụt, ta xác định được nước đi tốt nhất cho Trắng tại u, đó là nước đi dẫn tới đỉnh v có giá trị 10. Cạnh mỗi đỉnh ta cũng cho giá trị của cặp tham số (α, β). Khi gọi các hàm MaxVal và MinVal để xác định giá trị của đỉnh đó. Các nhánh bị cắt bỏ được chỉ ra trong hình:



Hình 4.8 Xác định giá trị các đỉnh bằng kỹ thuật cắt cụt.

Phần II:

Tri thức và lập luận

Chương V:

Logic mệnh đề

Trong chương này chúng ta sẽ trình bày các đặc trưng của ngôn ngữ biểu diễn tri thức. Chúng ta sẽ nghiên cứu logic mệnh đề, một ngôn ngữ biểu diễn tri thức rất đơn giản, có khả năng biểu diễn hẹp, nhưng thuận lợi cho ta làm quen với nhiều khái niệm quan trọng trong logic, đặc biệt trong logic vị từ cấp một sẽ được nghiên cứu trong các chương sau.

5.1. Biểu diễn tri thức

Con người sống trong môi trường có thể nhận thức được thế giới nhờ các giác quan (tai, mắt và các bộ phận khác), sử dụng các tri thức tích luỹ được và nhờ khả năng lập luận, suy diễn, con người có thể đưa ra các hành động hợp lý cho công việc mà con người đang làm. Một mục tiêu của Trí tuệ nhân tạo ứng dụng là thiết kế các *tác nhân thông minh* (intelligent agent) cũng có khả năng đó như con người. Chúng ta có thể hiểu tác nhân thông minh là bất cứ cái gì có thể nhận thức được môi trường thông qua các bộ cảm nhận (sensors) và đưa ra hành động hợp lý đáp ứng lại môi trường thông qua bộ phận hành động (effectors). Các robots, các softbot (software robot), các hệ chuyên gia,... là các ví dụ về tác nhân thông minh. Các tác nhân thông minh cần phải có tri thức về thế giới hiện thực mới có thể đưa ra các quyết định đúng đắn.

Thành phần trung tâm của các *tác nhân dựa trên tri thức* (knowledge-based agent), còn được gọi là *hệ dựa trên tri thức* (knowledge-based system) hoặc đơn giản là hệ tri thức, là cơ sở tri thức. Cơ sở tri thức (CSTT) là một tập hợp các tri thức được biểu diễn dưới dạng nào đó. Mỗi khi nhận được các thông tin đưa vào, tác nhân cần có khả năng suy diễn để đưa ra các câu trả lời, các hành động hợp lý, đúng đắn. Nhiệm vụ này được thực hiện bởi bộ suy diễn. Bộ suy diễn là thành phần cơ bản khác của các hệ tri thức. Như vậy hệ tri thức bảo trì một CSTT và được trang bị một thủ tục suy diễn. Mỗi khi tiếp nhận được các sự kiện từ môi trường, thủ tục suy diễn thực hiện quá trình liên kết các sự kiện với các tri thức trong CSTT để rút ra các câu trả lời, hoặc các hành động hợp lý mà tác nhân cần thực hiện. Đương nhiên là, khi ta thiết kế một tác nhân giải quyết một vấn đề nào đó thì CSTT sẽ chứa các tri thức về miền đối tượng cụ thể đó. Để máy tính có thể

sử dụng được tri thức, có thể xử lý tri thức, chúng ta cần biểu diễn tri thức dưới dạng thuận tiện cho máy tính. Đó là mục tiêu của biểu diễn tri thức.

Tri thức được mô tả dưới dạng các câu trong *ngôn ngữ biểu diễn tri thức*. Mỗi câu có thể xem như sự mã hóa của một sự hiểu biết của chúng ta về thế giới hiện thực. Ngôn ngữ biểu diễn tri thức (cũng như mọi ngôn ngữ hình thức khác) gồm hai thành phần cơ bản là *cú pháp* và *ngữ nghĩa*.

 Cú pháp của một ngôn ngữ bao gồm các ký hiệu về các quy tắc liên kết các ký hiệu (các luật cú pháp) để tạo thành các câu (công thức) trong ngôn ngữ. Các câu ở đây là biểu diễn ngoài, cần phân biệt với biểu diễn bên trong máy tính. Các câu sẽ được chuyển thành các cấu trúc dữ liệu thích hợp được cài đặt trong một vùng nhớ nào đó của máy tính, đó là biểu diễn bên trong. Bản thân các câu chưa chứa đựng một nội dung nào cả, chưa mang một ý nghĩa nào cả.

 Ngữ nghĩa của ngôn ngữ cho phép ta xác định ý nghĩa của các câu trong một miền nào đó của thế giới hiện thực. Chẳng hạn, trong ngôn ngữ các biểu thức số học, dãy ký hiệu (x+y)*z là một câu viết đúng cú pháp. Ngữ nghĩa của ngôn ngữ này cho phép ta hiểu rằng, nếu x, y, z, ứng với các số nguyên, ký hiệu + ứng với phép toán cộng, còn * ứng với phép chia, thì biểu thức (x+y)*z biểu diễn quá trình tính toán: lấy số nguyên x cộng với số nguyên y, kết quả được nhân với số nguyên z.

 Ngoài hai thành phần cú pháp và ngữ nghĩa, ngôn ngữ biểu diễn tri thức cần được cung cấp *cơ chế suy diễn*. Một luật suy diễn (rule of inference) cho phép ta suy ra một công thức từ một tập nào đó các công thức. Chẳng hạn, trong logic mệnh đề, luật modus ponens từ hai công thức A và $A \Rightarrow B$ suy ra công thức B. Chúng ta sẽ hiểu *lập luận* hoặc *suy diễn* là một quá trình áp dụng các luật suy diễn để từ các tri thức trong cơ sở tri thức và các sự kiện ta nhận được các tri thức mới. Như vậy chúng ta xác định:

1.1 Ngôn ngữ biểu diễn tri thức = Cú pháp + Ngữ nghĩa + Cơ chế suy diễn.

1.2 Một ngôn ngữ biểu diễn tri thức tốt cần phải có khả năng biểu diễn rộng, tức là có thể mô tả được mọi điều mà chúng ta muốn nói. Nó cần phải hiệu quả theo nghĩa là, để đi tới các kết luận, thủ tục suy diễn đòi hỏi ít thời gian tính toán và ít không gian nhớ. Người ta cũng mong muốn ngôn ngữ biểu diễn tri thức gần với ngôn ngữ tự nhiên.

1.3 Trong sách này, chúng ta sẽ tập trung nghiên cứu logic vị từ cấp một (first-order predicate logic hoặc first-order predicate calculus) - một ngôn ngữ biểu diễn tri thức, bởi vì logic vị từ cấp một có khả năng biểu diễn tương đối tốt, và hơn nữa nó là cơ sở cho nhiều ngôn ngữ biểu diễn tri thức khác, chẳng hạn toán hoàn cảnh (situation calculus) hoặc logic thời gian khoảng cấp một (first-order interval tempral logic). Nhưng trước hết chúng ta sẽ nghiên cứu logic mệnh đề (propositional logic hoặc propositional calculus). Nó là ngôn ngữ rất đơn giản, có khả năng biểu diễn hạn chế, song thuận tiện cho ta đưa vào nhiều khái niệm quan trọng trong logic.

5.2. Cú pháp và ngữ nghĩa của logic mệnh đề.

5.2.1 Cú pháp:

Cú pháp của logic mệnh đề rất đơn giản, nó cho phép xây dựng nên các công thức. Cú pháp của logic mệnh đề bao gồm tập các ký hiệu và tập các luật xây dựng công thức.

1. Các ký hiệu

- Hai hằng logic True và False.
- Các ký hiệu mệnh đề (còn được gọi là các biến mệnh đề): P, Q,...
- Các kết nối logic $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$.
- Các dấu mở ngoặc (và đóng ngoặc).

2. Các quy tắc xây dựng các công thức

- Các biến mệnh đề là công thức.
- Nếu A và B là công thức thì:

($A \wedge B$) (đọc "A hội B" hoặc "A và B")

($A \vee B$) (đọc "A tuyển B" hoặc "A hoặc B")

($\neg A$) (đọc "phủ định A")

($A \Rightarrow B$) (đọc "A kéo theo B" hoặc "nếu A thì B")

($A \Leftrightarrow B$) (đọc "A và B kéo theo nhau")

là các công thức.

Sau này để cho ngắn gọn, ta sẽ bỏ đi các cặp dấu ngoặc không cần thiết. Chẳng hạn, thay cho ((A∨B)∧C) ta sẽ viết là (A∨B)∧C.

Các công thức là các ký hiệu mệnh đề sẽ được gọi là các *câu đơn* hoặc *câu phân tử*. Các công thức không phải là câu đơn sẽ được gọi là câu phức hợp. Nếu P là ký hiệu mệnh đề thì P và TP được gọi là *literal*, P là *literal dương*, còn TP là *literal âm*. Câu phức hợp có dạng $A_1 \vee ... \vee A_m$ trong đó A_i là các literal sẽ được gọi là *câu tuyển* (clause).

5.2.2 **Ngữ nghĩa:**

Ngữ nghĩa của logic mệnh đề cho phép ta *xác định* thiết lập ý nghĩa của các công thức trong thế giới hiện thực nào đó. Điều đó được thực hiện bằng cách kết hợp mệnh đề với sự kiện nào đó trong thế giới hiện thực. Chẳng hạn, ký hiệu mệnh đề P có thể ứng với sự kiện "Paris là thủ đô nước Pháp" hoặc bất kỳ một sự kiện nào khác. Bất kỳ một sự kết hợp các kí hiệu mệnh đề với các sự kiện trong thế giới thực được gọi là một *minh họa* (interpretation). Chẳng hạn minh họa của kí hiệu mệnh đề P có thể là một sự kiện (mệnh đề) "Paris là thủ đô nước Pháp ". Một sự kiện chỉ có thể đúng hoặc sai. Chẳng hạn, sự kiện "Paris là thủ đô nước Pháp " là đúng, còn sự kiện "Số Pi là số hữu tỉ " là sai.

Một cách chính xác hơn, cho ta hiểu một minh họa là một cách gán cho mỗi ký hiệu mệnh đề một giá trị chân lý **True** hoặc **False.** Trong một minh họa, nếu kí hiệu mệnh đề P được gán giá trị chân lý **True/False** (P <-**True**/ P<-**False**) thì ta nói mệnh đề P **đúng/sai** trong minh họa đó. Trong một minh họa, ý nghĩa của các câu phức hợp được xác định bởi ý nghĩa của các kết nối logic. Chúng ta xác định ý nghĩa của các kết nối logic trong các bảng chân lý (xem hình 5.1)

P	Q	lP	P∧ Q	P v Q	P=> Q	P<= >Q
Fals e	Fals e	Tru e	Fals e	Fals e	Tru e	Tru e
Fals e	Tru e	Tru e	Fals e	Tru e	Tru e	Fals e
Tru e	Fals e	Fals e	Fals e	Tru e	Fals e	Fals e
Tru e	Tru e	Fals e	Tru e	Tru e	Tru e	Tru e

Hình 5.1 Bảng chân lý của các kết nối logic

ý nghĩa của các kết nối logic ∧, v và l được xác định như các từ "**và**","**hoặc là**" và "**phủ định**" trong ngôn ngữ tự nhiên. Chúng ta cần phải giải thích thêm về ý nghĩa của phép kéo theo P => Q (P kéo theo Q), P là giả thiết, còn Q là kết luận. Trực quan cho phép ta xem rằng, khi P là đúng và Q là đúng thì câu "P kéo theo Q " là đúng, còn khi P là đúng Q là sai thì câu "P kéo theo Q" là sai.

Nhưng nếu P sai và Q đúng , hoặc P sai Q sai thì "P kéo theo Q" là đúng hay sai ? Nếu chúng ta xuất phát từ giả thiết sai, thì chúng ta không thể khảng định gì về kết luận. Không có lý do gì để nói rằng, nếu P sai và Q đúng hoặc P sai và Q sai thì "P kéo theo Q" là sai. Do đó trong trường hợp P sai thì "P kéo theo Q " là đúng dù Q là đúng hay Q là sai.

Bảng chân lý cho phép ta xác định ngẫu nhiên các câu phức hợp. Chẳng hạn ngữ nghĩa của các câu P∧Q trong minh họa {P <- **True** , Q<- **False** } là **False**. Việc xác định ngữ nghĩa của một câu (P v Q) ∧ lS trong một minh họa được tiến hành như sau: đầu tiên ta xác định giá trị chân lý của P v Q và l S , sau đó ta sử dụng bảng chân lý ∧ để xác định giá trị (PvQ) ∧lS

- Một công thức được gọi là *thoả được* (satisfiable) nếu nó đúng trong một minh họa nào đó. Chẳng hạn công thức (PvQ) ∧lS là thoả được, vì nó có giá trị **True** trong minh họa {P <- **True**, Q<-**False**, S<- **True**}.
- Một công thức được gọi là *vững chắc*_(valid hoặc tautology) nếu nó đúng trong mọi minh họa chẳng hạn câu P vlP là vững chắc
- Một công thức được gọi là *không thoả được* , nếu nó là sai trong mọi minh họa. Chẳng hạn công thức P ∧ lP.

Chúng ta sẽ gọi một *mô hình* (modul) của một công thức là một minh họa sao cho công thức là đúng trong minh họa này. Như vậy một công thức *thoả được* là công thức có một mô hình. Chẳng hạn, minh họa {P <- **False** , Q <- **False** , S<- **True** } là một mô hình của công thức (P =>Q) ∧ S .

Bằng cách lập bảng chân lý (*phương pháp bảng chân lý*) là ta có thể xác định được một công thức có *thoả được* hay không. Trong bảng này, mỗi biến mệnh đề đứng đầu với một cột, công thức cần kiểm tra đứng đầu một cột, mỗi dòng tương ứng với một minh họa. Chẳng hạn hình 5.2 là bảng chân lý cho công thức (P=>Q) ∧S. Trong bảng chân lý này ta cần đưa vào các cột phụ ứng với các công thức con của các công thức cần kiểm tra để việc tính giá trị của công thức này được dễ dàng. Từ bảng chân lý ta thấy rằng công thức (P=>Q) ∧S là *thoả được* nhưng không *vững chắc* .

P	Q	S	P=>Q	(P=>Q) ∧S
False	False	False	True	False
False	False	True	True	True
False	True	False	True	False
False	True	True	True	True
True	False	False	False	False

True	False	True	False	False
True	True	False	True	False
True	True	True	True	True

Hình 5.2 Bảng chân lý cho công thức (P=>Q) ∧S

Cần lưu ý rằng, một công thức chứa n biến, thì số các minh họa của nó là 2^n , tức là bảng chân lý có 2^n dòng. Như vậy việc kiểm tra một công thức có *thoả được* hay không bằng phương pháp bảng chân lý, đòi hỏi thời gian mũ. Cook (1971) đã chứng minh rằng, vấn đề kiểm tra một công thức trong logic mệnh đề có *thoả được* hay không là vấn đề NP-đầy đủ.

Chúng ta sẽ nói rằng (*thoả được, không thoả được*) nếu hội của chúng $G_1 \wedge$ .....$\wedge G_m$ là *vững chắc* (*thoả được, không thoả được*). Một mô hình của tập công thức G là mô hình của tập công thức $G_1 \wedge$.....$\wedge G_m$.

5.3 Dạng chuẩn tắc

Trong mục này chúng ta sẽ xét việc chuẩn hóa các công thức, đưa các công thức về dạng thuận lợi cho việc lập luận, suy diễn. Trước hết ta sẽ xét các phép biến đổi tương đương. Sử dụng các phép biển đổi này, ta có thể đưa một công thức bất kỳ về các dạng chuẩn tắc.

5.3.1 Sự tương đương của các công thức

Hai công thức A và B được xem là *tương đương* nếu chúng có cùng một giá trị chân lý trong mọi minh họa. Để chỉ A tương đương với B ta viết A≡ B bằng phương pháp bảng chân lý, dễ dàng chứng minh được sự tương đương của các công thức sau đây :

- A=>B ≡ lA v B
- A< = > B ≡ (A=>B) ∧ (B=>A)
- l(lA) ≡ A

1.4 Luật De Morgan

- l(A v B) ≡ lA ∧ lB
- l(A ∧ B) ≡ lA v lB

1.5 Luật giao hoán

- A v B ≡ B v A

- A ∧ B ≡ B ∧ A

1.6 Luật kết hợp

- (A v B) v C ≡ Av(B v C)
- (A ∧ B) ∧ C ≡ A∧ (B ∧ C)

1.7 Luật phân phối

- A ∧ (B v C) ≡ (A ∧ B) v (A ∧ C)
- A v (B ∧ C) ≡ (A v B) ∧ (A v C)

5.3.2 *Dạng chuẩn tắc :*

Các công thức tương đương có thể xem như các biểu diễn khác nhau của cùng một sự kiện. Để dễ dàng viết các chương trình máy tính thao tác trên các công thức, chúng ta sẽ chuẩn hóa các công thức, đưa chúng về dạng biểu diễn chuẩn được gọi là *dạng chuẩn hội.* Một công thức ở dạng chuẩn hội, có dạng A_1 v ... .v A_m trong đó các A_i là *literal* . Chúng ta có thể biến đổi một công thức bất kỳ về công thức ở dạng chuẩn hội bằng cách áp dụng các thủ tục sau.

- Bỏ các dấu kéo theo (=>) bằng cách thay (A=>B) bởi (lAvB).
- Chuyển các dấu phủ định (l) vào sát các kết hiệu mệnh đề bằng cách áp dụng luật De Morgan và thay l(lA) bởi A .

- áp dụng luật phân phối, thay các công thức có dạng Av(B∧C) bởi (A v B) ∧ (A v B) .

Ví dụ : Ta chuẩn hóa công thức (P => Q) v l(R v lS) :

(P => Q) v l(R v lS) ≡ (lP v Q) v (lR ∧ S) ≡ ((lP v Q)vlR) ∧ ((lP v Q) v S) ≡ (l P v Q v lR) ∧ (lP v Q v S). Như vậy công thức (P=> Q) v l(R v lS) được đưa về dạng chuẩn hội (lP v Q v lR) ∧ (lP v Q v S).

Khi biểu diễn tri thức bởi các công thức trong logic mệnh đề, cơ sở tri thức là một tập nào đó các công thức. Bằng cách chuẩn hoá các công thức, cơ sở tri thức là một tập nào đó các câu tuyển.

Các câu Horn:

ở trên ta đã chỉ ra, mọi công thức đều có thể đưa về dạng chuẩn hội, tức là các hội của các tuyển, mỗi câu tuyển có dạng

lP_1 v.....v lP_m v Q_1 v.....v Q_m

trong đó P_i , Q_i là các ký hiệu mệnh đề (literal dương) câu này tương đương với câu

lP_1 v.....v lP_m => v Q_1 v.....v Q_m **????** p1^ .... ^ pm => **Q**

Dạng câu này được gọi là *câu Kowalski* (do nhà logic Kowalski đưa ra năm 1971).

Khi n <=1, tức là câu Kowalski chỉ chứa nhiều nhất một literal dương ta có dạng một câu đặc biệt quan trọng được gọi là *câu Horn* (mang tên nhà logic Alfred Horn năm 1951).

Nếu m>0, n=1, câu Horn có dạng :

$P_1 \wedge ..... \wedge P_m \Rightarrow Q$

Trong đó P_i , Q là các literal dương. Các P_i được gọi là các điều kiện (hoặc giả thiết), còn Q được gọi là kết luận (hoặc hệ quả). Các câu Horn dạng này còn được gọi là các luật ***if ... then*** và được biểu diễn như sau :

If P_1 and ....and P_m *then* Q .

Khi m=0, n=1 câu Horn trở thành câu đơn Q, hay sự kiện Q. Nếu m>0, n=0 câu Horn trở thành dạng $\neg P_1 \vee ..... \vee \neg P_m$ hay tương đương $\neg(P_1 \wedge ... \wedge P_m)$. Cần chú ý rằng, không phải mọi công thức đều có thể biểu diễn dưới dạng hội của các câu Horn. Tuy nhiên trong các ứng dụng, cơ sở tri thức thường là một tập nào đó các câu Horn (tức là một tập nào đó các luật if-then).

5.4 Luật suy diễn

Một công thức H được xem là *hệ qủa logic* (logical consequence) của một tập công thức $G = \{G_1,.....,G_m\}$ nếu trong bất kỳ minh họa nào mà $\{G_1,.....,G_m\}$ đúng thì H cũng đúng, hay nói cách khác bất kỳ một mô hình nào của *G* cũng là mô hình của H.

Khi có một cơ sở tri thức, ta muốn sử dụng các tri thức trong cơ sở này để suy ra tri thức mới mà nó là hệ quả logic của các công thức trong cơ sở tri thức. Điều đó được thực hiện bằng các thực hiện *các luật suy diễn* (rule of inference). Luật suy diễn giống như một thủ tục mà chúng ta sử dụng để sinh ra một công thức mới từ các công thức đã có. Một luật suy diễn gồm hai phần : một tập các điều kiện và một kết luận. Chúng ta sẽ biểu diễn các luật suy diễn dưới dạng "phân số ", trong đó tử số là danh sách các điều kiện, còn mẫu số là kết luận của luật, tức là mẫu số là công thức mới được suy ra từ các công thức ở tử số.

Sau đây là một số luật suy diễn quan trọng trong logic mệnh đề. Trong các luật này α, α_i , β, γ là các công thức :

1. Luật Modus Ponens

$$\frac{\alpha => \beta, \alpha}{\beta}$$

Từ một kéo theo và giả thiết của kéo theo, ta suy ra kết luận của nó.

2. Luật Modus Tollens

$$\frac{\alpha => \beta, l\beta}{l\alpha}$$

Từ một kéo theo và phủ định kết luận của nó, ta suy ra phủ định giả thiết của kéo theo.

3. Luật bắc cầu

$\alpha => \beta, \beta => \gamma$

$\alpha => \gamma$

Từ hai kéo theo, mà kết luận của nó là của kéo theo thứ nhất trùng với giả thiết của kéo theo thứ hai, ta suy ra kéo theo mới mà giả thiết của nó là giả thiết của kéo theo thứ nhất, còn kết luận của nó là kết luận của kéo theo thứ hai.

4. Luật loại bỏ hội

$\alpha_1 \wedge ..... \wedge \alpha_i \wedge ..... \wedge \alpha_m$

α_i

Từ một hội ta đưa ra một nhân tử bất kỳ của hội .

5. Luật đưa vào hội

$\alpha_1, ..., \alpha_i, ... \alpha_m$

$\alpha_1 \wedge ..... \wedge \alpha_i \wedge ..... \wedge \alpha_m$

Từ một danh sách các công thức, ta suy ra hội của chúng.

6. Luật đưa vào tuyển

α_i

$\alpha_1 v ..... v \alpha_i . v ..... v \alpha_m$

Từ một công thức, ta suy ra một tuyển mà một trong các hạng tử của các tuyển là công thức đó.

7. Luật giải

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Từ hai tuyển, một tuyển chứa một hạng tử đối lập với một hạng tử trong tuyển kia, ta suy ra tuyển của các hạng tử còn lại trong cả hai tuyển.

Một luật suy diễn được xem là *tin cậy* (secured) nếu bất kỳ một mô hình nào của giả thiết của luật cũng là mô hình kết luận của luật. Chúng ta chỉ quan tâm đến các luật suy diễn tin cậy.

Bằng phương pháp bảng chân lý, ta có thể kiểm chứng được các luật suy diễn nêu trên đều là tin cậy. Bảng chân lý của luật giải được cho trong hình 5.3. Từ bảng này ta thấy rằng , trong bất kỳ một minh họa nào mà cả hai giả thiết $\alpha \vee \beta$, $\neg\beta \vee \gamma$ đúng thì kết luận $\alpha \vee \gamma$ cũng đúng. Do đó luật giải là luật suy điễn tin cậy.

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	False	False	False	True	False
False	False	True	False	True	True

False	True	False	True	False	False
False	True	True	True	True	True
True	False	False	True	True	True
True	False	True	True	True	True
True	True	False	True	False	True
True	True	True	True	True	True

Hình 5.3 Bảng chân lý chứng minh tính tin cậy của luật giải.

Ta có nhận xét rằng, luật giải là một luật suy diễn tổng quát, nó bao gồm luật Modus Ponens, luật Modus Tollens, luật bắc cầu như các trường hợp riêng. (Bạn đọc dễ dàng chứng minh được điều đó).

Tiên đề định lý chứng minh.

Giả sử chúng ta có một tập nào đó các công thức. Các luật suy diễn cho phép ta từ các công thức đã có suy ra công thức mới bằng một dãy áp dụng các

luật suy diễn. Các công thức đã cho được gọi là các *tiên đề* . Các công thức được suy ra được gọi là các *định lý*. Dãy các luật được áp dụng để dẫn tới định lý được gọi là một *chứng minh* của định lý. Nếu các luật suy diễn là tin cậy, thì các định lý là hệ quả logic của các tiên đề.

Ví dụ: Giả sử ta có các công thức sau :

$Q \wedge S \Rightarrow G \vee H$ (1)

$P \Rightarrow Q$ (2)

$R \Rightarrow S$ (3)

P (4)

R (5)

Từ công thức (2) và (4), ta suy ra Q (Luật Modus Ponens) . Lại áp dụng luật Modus Ponens, từ (3) và (5) ta suy ra S . Từ Q, S ta suy ra $Q \wedge S$ (luật đưa vào hội). Từ (1) và $Q \wedge S$ ta suy ra G v H. Công thức G v H đã được chứng minh.

Trong các hệ tri thức, chẳng hạn các hệ chuyên gia, hệ lập trình logic,..., sử dụng các luật suy diễn người ta thiết kế lên các *thủ tục suy diễn*_ (còn được gọi là *thủ tục chứng minh*) để từ các tri thức trong cơ sở tri thức ta suy ra các tri thức mới đáp ứng nhu cầu của người sử dụng.

Một *hệ hình thức* (formal system) bao gồm một tập các tiên đề và một tập các luật suy diễn nào đó (trong ngôn ngữ biểu diễn tri thức nào đó).

Một tập luật suy diễn được xem là *đầy đủ*, nếu mọi hệ quả logic của một tập các tiên đề đều chứng minh được bằng cách chỉ sử dụng các luật của tập đó.

Phương pháp chứng minh bác bỏ

Phương pháp chứng minh bác bỏ (refutation proof hoặc proof by contradiction) là một phương pháp thường xuyên được sử dụng trong các chứng minh toán học. Tư tưởng của phương pháp này là như sau : Để chứng minh P đúng, ta giả sử P sai (thêm ⌉P vào các giả thiết) và dẫn tới một mâu thuẫn. Sau đây ta sẽ trình bầy cơ sở này.

Giả sử chúng ta có một tập hợp các công thức $G = \{G_1,.....,G_m\}$ ta cần chứng minh công thức H là hệ quả logic của G . Điều đó tương đương với chứng minh công thức $G_1 \wedge .... \wedge G_m \rightarrow H$ là vững chắc. Thay cho chứng minh $G_1 \wedge ..... \wedge G_m \Rightarrow H$ là vững chắc, ta chứng minh $G_1 \wedge .... \wedge G_m \wedge \neg H$ là không thỏa mãn được. Tức là ta chứng minh tập G'‘= ($G_1,.....,G_m, \neg H$) là không thỏa được nếu từ G‘ta suy ra hai mệnh đề đối lập nhau. Việc chứng minh công thức H là hệ quả logic của tập các tiêu đề G bằng cách chứng minh tính không thỏa được của tập các tiêu đề được thêm vào phủ định của công thức cần chứng minh, được gọi là chứng minh bác bỏ.

5.5 Luật giải, chứng minh bác bỏ bằng luật giải

Để thuận tiện cho việc sử dụng luật giải, chúng ta sẽ cụ thể hoá luật giải trên các dạng câu đặc biệt quan trọng.

*Luật giải trên các câu tuyển

A1 v. . ..... vAm v C

⌉C v B1 v.. ..... v Bn

A1 v.. ..... v Am v B1 v.... v Bn

trong đó Ai, Bj và C là các literal.

* Luật giải trên các câu Horn:

Giả sử Pi, Rj, Q và S là các literal. Khi đó ta có các luật sau :

P1 ^. .....^Pm ^ S => Q,

R1 ^. .....^ Rn => S

P1 ^.....^Pm ^ R1 ^..... ^ Rn =>Q

Một trường hợp riêng hay được sử dụng của luật trên là :

P1 ^.....^ Pm ^ S => Q,

S

P1 ^.....^Pm => Q

Khi ta có thể áp dụng luật giải cho hai câu, thì hai câu này được gọi là *hai câu giải được* và kết quả nhận được khi áp dụng luật giải cho hai câu đó được gọi là *giải thức* của chúng. Giải thức của hai câu A và B được kí hiệu là res(A,B). Chẳng hạn, hai câu tuyển giải được nếu một câu chứa một literal đối lập với một

literal trong câu kia. Giải thức của hai literal đối lập nhau (P và ⌝P) là câu rỗng, chúng ta sẽ ký hiệu câu rỗng là [] , câu rỗng không thoả được.

Giả sử G là một tập các câu tuyển (Bằng cách chuẩn hoá ta có thể đưa một tập các công thức về một tập các câu tuyển). Ta sẽ ký hiệu R(G) là tập câu bao gồm các câu thuộc G và tất cả các câu nhận được từ G bằng một dãy áp dụng luật giải.

Luật giải là luật đầy đủ để chứng minh một tập câu là không thỏa được. Điều này được suy từ định lý sau :

Định lý giải:

Một tập câu tuyển là không thỏa được nếu và chỉ nếu câu rỗng [] ∈ R(G).

Định lý giải có nghĩa rằng, nếu từ các câu thuộc G , bằng cách áp dụng luật giải ta dẫn tới câu rỗng thì G là không thỏa được, còn nếu không thể sinh ra câu rỗng bằng luật giải thì G thỏa được. Lưu ý rằng, việc dẫn tới câu rỗng có nghĩa là ta đã dẫn tới hai literal đối lập nhau P và ⌝P (tức là dẫn tới mâu thuẫn).

Từ định lý giải, ta đưa ra thủ tục sau đây để xác định một tập câu tuyển G là thỏa được hay không . Thủ tục này được gọi là thủ tục giải.

procedure Resolution ;

Input : tập G các câu tuyển ;

begin

1.*Repeat*

1.1 Chọn hai câu A và B thuộc G ;

1.2 *if* A và B giải được *then* tính Res (A,B) ;

1.3 *if* Res (A,B) là câu mới *then* thêm Res (A,B) vào G ;

until

nhận được [] hoặc không có câu mới xuất hiện ;

2. *if* nhận được câu rỗng *then* thông báo G không thoả được

e lse thông báo G thoả được ;

end;

Chúng ta có nhận xét rằng, nếu G là tập hữu hạn các câu thì các literal có mặt trong các câu của G là hữu hạn. Do đó số các câu tuyển thành lập được từ các literal đó là hữu hạn. Vì vậy chỉ có một số hữu hạn câu được sinh ra bằng luật giải. Thủ tục giải sẽ dừng lại sau một số hữu hạn bước.

Chỉ sử dụng luật giải ta không thể suy ra mọi công thức là hệ quả logic của một tập công thức đã cho. Tuy nhiên, sử dụng luật giải ta có thể chứng minh được một công thức bất kì có là hệ quả của một tập công thức đã cho hay không

bằng phương pháp chứng minh bác bỏ. Vì vậy luật giải được xem là *luật đầy đủ cho bác bỏ.*

Sau đây là thủ tục chứng minh bác bỏ bằng luật giải

Procedure Refutation_Proof ;

input : Tập G các công thức ;

Công thức cần chứng minh H;

Begin

1. Thêm ¬H vào G ;

2. Chuyển các công thức trong G về dạng chuẩn hội ;

3. Từ các dạng chuẩn hội ở bước hai, thành lập tạp các câu tuyển g' ;

4. áp dụng thủ tục giải cho tập câu G' ;

5. *if* G' không thoả được *then* thông báo H là hệ quả logic

else thông báo H không là hệ quả logic của G ;

end;

Ví dụ: Giả giử G là tập hợp các câu tuyển sau

¬A v ¬B v P (1)

¬C v ¬D v P (2)

¬E v C (3)

A (4)

E (5)

D (6)

Giả sử ta cần chứng minh P. Thêm vào G câu sau:

¬P (7)

áp dụng luật giải cho câu (2) và (7) ta được câu:

¬C v ¬D (8)

Từ câu (6) và (8) ta nhận được câu:

¬C (9)

Từ câu (3) và (9) ta nhận được câu:

¬E (10)

Tới đây đã xuất hiện mâu thuẫn, vì câu (5) và (10) đối lập nhau. Từ câu (5) và (10) ta nhận được câu rỗng []. Vậy P là hệ quả logic của các câu (1) --(6).

Trí tuệ Nhân tạo

Các cấu trúc & chiến lược giải quyết vấn đề

Tập 1

Artificial Intelligence

George F. Luger & William A. Stubblefield

NHÀ XUẤT BẢN THỐNG KÊ

ĐẠI HỌC QUỐC GIA
TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ
KHOA ĐIỆN VÀ ĐIỆN TỬ
BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

BÀI GIẢNG MÔN HỌC :
Trí Tuệ Nhân Tạo Và Hệ Chuyên Gia

Thành phố Hồ Chí Minh Ngày 7 Tháng 01 Năm 2006

Biên soạn : Tiến sĩ Nguyễn Thiện Thành

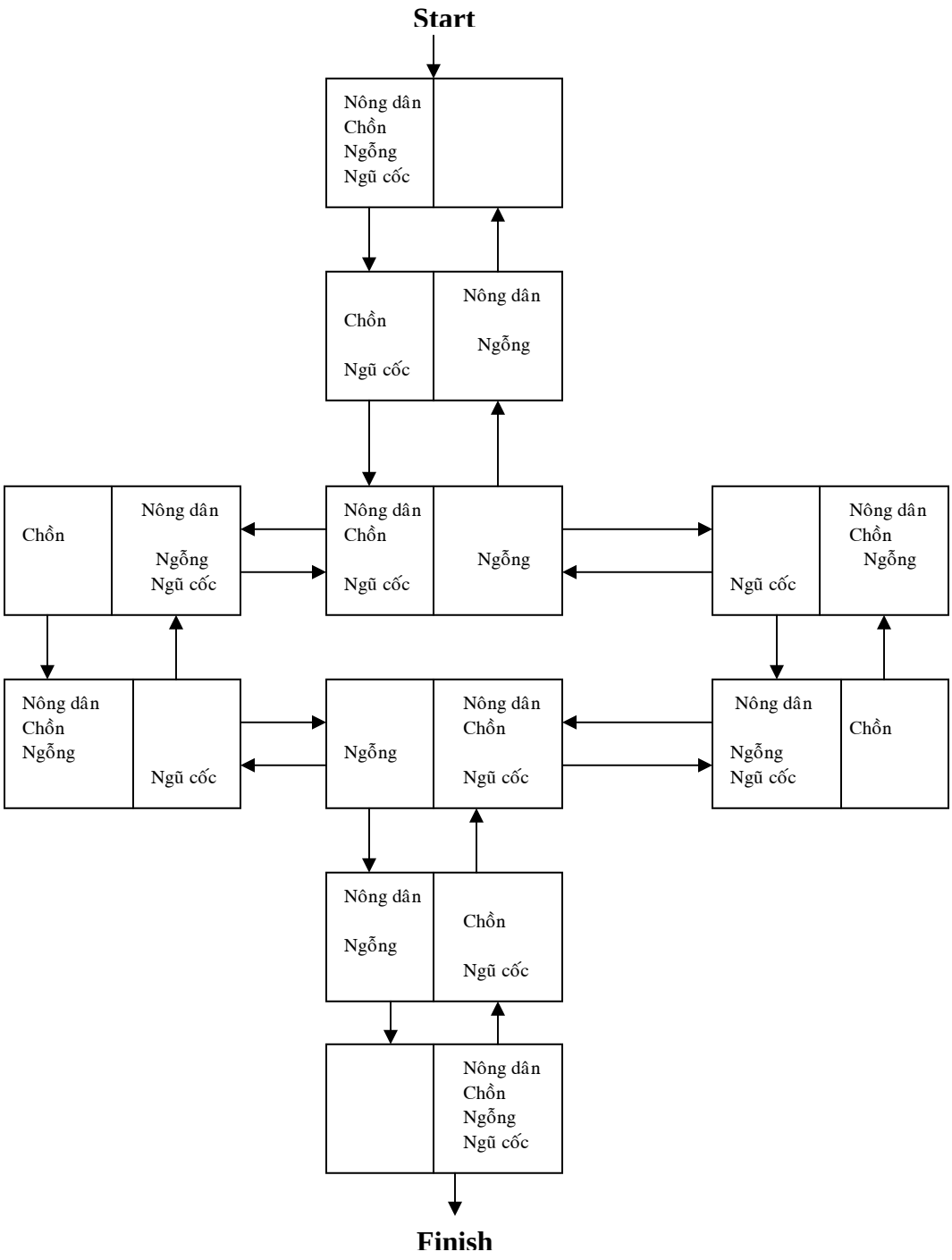
Nội dung bài giảng:

Sau khi tri thức của bài tóan đã được biểu diễn, kỹ thuật giải bài tóan trong lĩnh vực trí tuệ nhân tạo là các phương pháp tìm kiếm trong miền đặc trưng tri thức về bài tóan đó.

Ví dụ : Xét bài tóan người nông dân, chồn, ngỗng và ngũ cốc. Bài tóan đặt ra là người nông dân muốn mang theo với mình một con chồn, một con ngỗng và một số ngũ cốc qua bên kia sông bằng một chiếc thuyền. Tuy nhiên, thuyền của ông ta quá bé chỉ có thể mang theo một thứ duy nhất với ông ta trên mỗi chuyến thuyền sang sông. Nếu ông ta để lại chồn và ngỗng bên này sông thì chồn sẽ ăn ngỗng và nếu ông ta để lại ngỗng và ngũ cốc thì ngỗng sẽ ăn hết số ngũ cốc. Hãy sắp xếp các chuyến thuyền sao cho người nông dân mang mọi thứ sang bên kia sông an tòan?

Với bài tóan này, ta có thể biểu diễn nhờ thông qua các phát biểu ngôn ngữ tự nhiên, tuy nhiên cách biểu diễn này không giúp ta vạch trần ra các ràng buộc vốn sẵn có trong bài tóan. Cách biểu diễn tốt nhất giúp ta có thể vạch trần các ràng buộc vốn sẵn có trong bài tóan là xây dựng một biểu đồ với các nút có đánh nhãn người nông dân mang theo thứ mà ông ta cần phải mang theo trên mỗi chuyến thuyền và các cạnh liên kết giữa các nút là các đường mũi tên chỉ các chuyến thuyền qua lại sông.

Cách biểu diễn này hàm chứa các thành phần như ngữ từ học, cấu trúc, thủ tục và ngữ nghĩa.
+ Ngữ từ học (Lexical) : là các từ vựng hợp lệ được sử dụng như là các ký hiệu trong biểu diễn.
+ Cấu trúc (Structure) : là các đường mũi tên liên kết giữa các nút chỉ định các chuyến thuyền qua lại sông.
+ Thủ tục (Procedure) : là mô tả cách giải bài tóan từ nút này đến nút kia nhờ thông các đường chỉ định mũi tên.
+ Ngữ nghĩa (Semantic) : là ý nghĩa của các nút và các cạnh liên kết thông qua cách giải bài tóan.
Biểu đồ biểu diễn bài tóan người nông dân, chồn, ngỗng và ngũ cốc được mô tả như hình

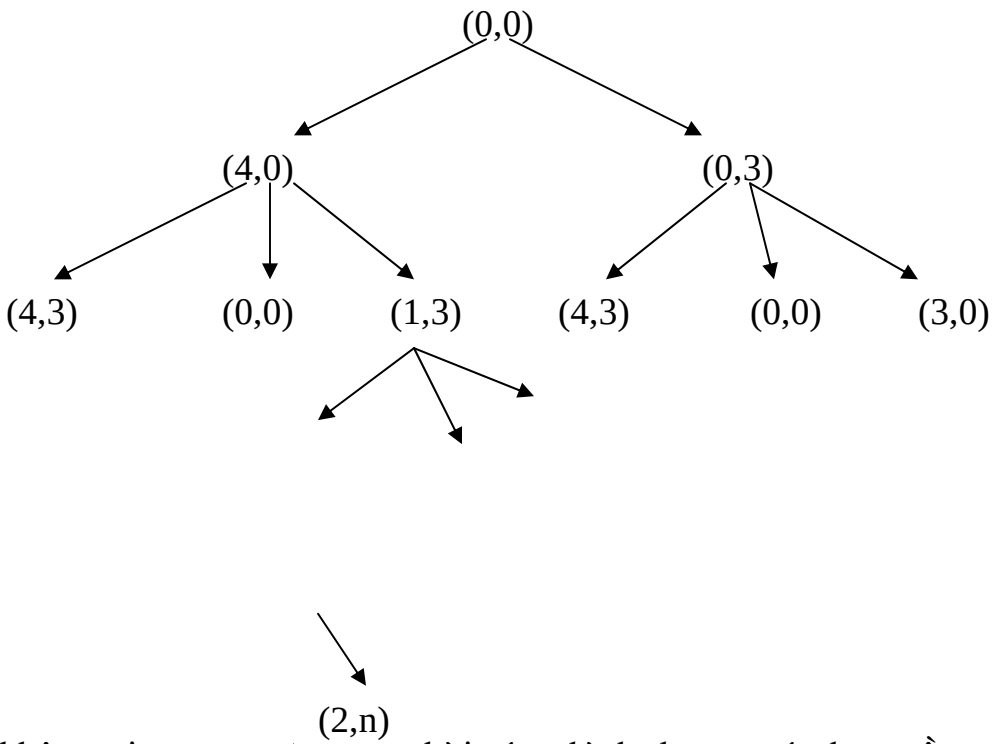

Start
Nông dân
Chồn
Ngỗng
Ngũ cốc
Chồn
Ngũ cốc
Nông dân
Ngỗng
Chồn
Nông dân
Ngỗng
Ngũ cốc
Nông dân
Chồn
Ngỗng
Ngũ cốc
Nông dân
Chồn
Ngỗng
Nông dân
Chồn
Ngỗng
Ngũ cốc
Nông dân
Chồn
Ngỗng
Ngũ cốc
Nông dân
Ngỗng
Chồn
Ngũ cốc
Nông dân
Chồn
Ngỗng
Ngũ cốc
Finish

+ Trạng thái khác của bài tóan : đó là cặp số nguyên (x,y) mô tả các trạng thái trong không gian bài tóan.

+ Trạng thái chuyển tiếp của bài tóan : đó la' bước chuyển tiếp từ trạng thái hiện có đến trạng thái mới nhờ thông luật áp dụng của bài tóan. Luật áp dụng là luật mà vế điều kiện của nó hợp với trạng thái hiện hữu để vế kết luận của nó phát sinh ra trạng thái mới. Tập các luật giải bài tóan bình đựng nước được liệt kê là

Luật 1 : $(x,y/\ x < 4) \rightarrow (4,y)$.
Luật 2 : $(x,y/\ y < 3) \rightarrow (x,3)$.
Luật 3 : $(x,y/\ x > 0) \rightarrow (0,y)$.
Luật 4 : $(x,y/\ y > 0) \rightarrow (x,0)$.
Luật 5 : $(x,y/\ x + y >= 4$ và $y > 0) \rightarrow (4,y - (4 - x))$.
Luật 6 : $(x,y/\ x + y >= 3$ và $x > 0) \rightarrow (x - (3 - y),3)$.
Luật 7 : $(x,y/\ x + y < 4$ và $y > 0) \rightarrow (x + y,0)$.
Luật 8 : $(x,y/x + y < 3$ và $x > 0) \rightarrow (0,x + y)$

Không gian trạng thái cho bài tóan này được biểu diễn bằng đồ thị như hình



Vậy, không gian trạng thái cho bài tóan bình đựng nước bao gồm trạng thái ban đầu, tất cả các trạng thái khác đạt được từ trạng thái ban đầu nhờ thông qua các luật ứng dụng (các trạng thái chuyển tiếp) và trạng thái đích của bài tóan.

+ Trạng thái ban của bài tóan : Tất cả mọi người và thuyền ở bên này sông với cấu hình là (MMM, CCC, B), trong đó M là tu sĩ, C là kẻ ăn thịt người và B là thuyền.

+ Trạng thái đích của bài tóan : Tất cả mọi người và thuyền đều được qua bên kia kia sông an tòan, vì thế cấu hình đích bên này sông là (_, _, _).

+ Ràng buộc của bài tóan : Số tu sĩ phải là luôn luôn lớn hơn hoặc bằng số kẻ ăn thịt người ở bất cứ nơi nào bên này sông, trên thuyền hoặc bên kia sông.

+ Trạng thái khác của bài tóan : cấu hình số tu sĩ, số kẻ ăn thịt người và thuyền ở bên này sông hoặc ở bên kia sông.

+ Trạng thái chuyển tiếp của bài tóan : bước dịch chuyển thuyền đưa một vài thành viên qua lại sông.

Bài tóan ba tu sĩ và ba kẻ ăn thịt người được giải gồm các bước như sau :

	Bên này sông	**Bên kia sông**
0. Trạng thái ban đầu	**(MMM,CCC,B)**	**(_ , _ , _)**
1. Hai kẻ ăn thịt người qua bên kia sông.	**(MMM, C, _)**	**(_ , CC, B)**
2. Một kẻ ăn thịt người qua lại bên này sông.	(MMM, CC, B)	(_ , C, _)
3. Hai kẻ ăn thịt người qua bên kia sông.	(MMM, _, _)	(_, CCC, B)
4. Một kẻ ăn thịt người qua lại bên này sông.	(MMM, CC,B)	(_ , CC, _)
5. Hai kẻ tu sĩ qua bên kia sông.	(M, CC, _)	(MM, CC,B)
6. Một tu sĩ và một kẻ ăn thịt người qua lại bên này sông.	(MM,CC,B)	(M, C, -)
7. Hai tu sĩ qua bên kia sông	(_, CC, _)	(MMM, C, B)
8. Một kẻ ăn thịt người qua bên này sông.	(_ , CCC, B)	(MMM, _ , _)
9. Hai kẻ ăn thị người qua bên kia sông.	(_ , C , _ B)	(MMM, CC, B)
10. Một kẻ ăn thịt người qua qua lại bên này sông.	(_, CC, B)	(MMM, C, -)
11. Hai kẻ ăn thịt người qua bên kia sông.	(_ , _ , _) Đích.	(MMM,CCC,B)

được lặp lại cho tất cả các nút viếng thăm cho đến khi nào trạng thái đích của bài tóan được tìm thấy. Cách tìm kiếm này tạo ra một đường liên kết bám theo các trạng thái có thông tin heuristic nỏ nhất đó là các trạng thái được đánh giá là tốt nhất để đạt đến đích.

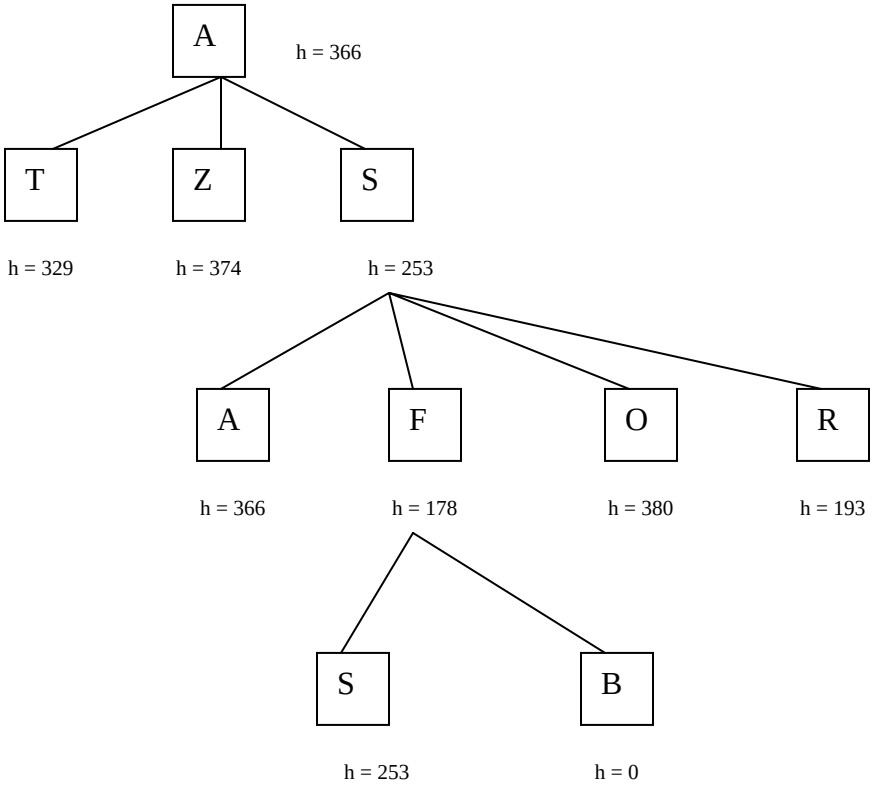
Giải thuật được mô tả như sau :

```
Procedure      best_first_search
Begin
        Open = [Start];
        Closed = [ ];
    While Open ≠ [ ]
    Begin
        + Lấy nút đầu tiên của danh sách Open , gọi nút này là X và loại bỏ X
        khỏi danh sách Open.
        + If X = đích Then return(success)
        Else
        Begin
        + Khai triển các thừa kế của X nhờ thông qua các luật ứng dụng.
        Cho mỗi thừa kế của X thực hiện một trong các trường hợp như sau :
        Case : Thừa kế chưa xuất hiện trên danh sách Open hoặc Closed.
                + Ước lượng heuristic cho thừa kế.
                + Đặt thừa kế vào danh sách Open.
        Case : Thừa kế đã có mặt trên danh sách Open.
                + Ước lượng heuristic cho thừa kế.
                + Nếu trạng thái mới xuất hiện là tốt hơn trạng thái cũ đã xuất
        hiện trên Open thì loại bỏ cũ khỏi danh sách Open và đặt mới vào danh
        sách Open; mặt khác giữ lại trạng thái cũ ở danh sách Open và loại bỏ
        trạng thái mới xuất hiện.
        Case : Thừa kế đã có mặt trên danh sách Closed.
                + Ước lượng heuristic cho thừa kế.
                + Nếu trạng thái mới xuất hiện là tốt hơn trạng thái cũ đã có mặt
        sẵn trên Closed thì loại bỏ cũ khỏi danh sách Closed và đặt mới vào
        danh sách Open; mặt khác giữ lại trạng thái cũ ở danh sách Closed và
        loại bỏ trạng thái mới xuất hiện.
        End
    + Đặt X vào danh sách Closed.
```

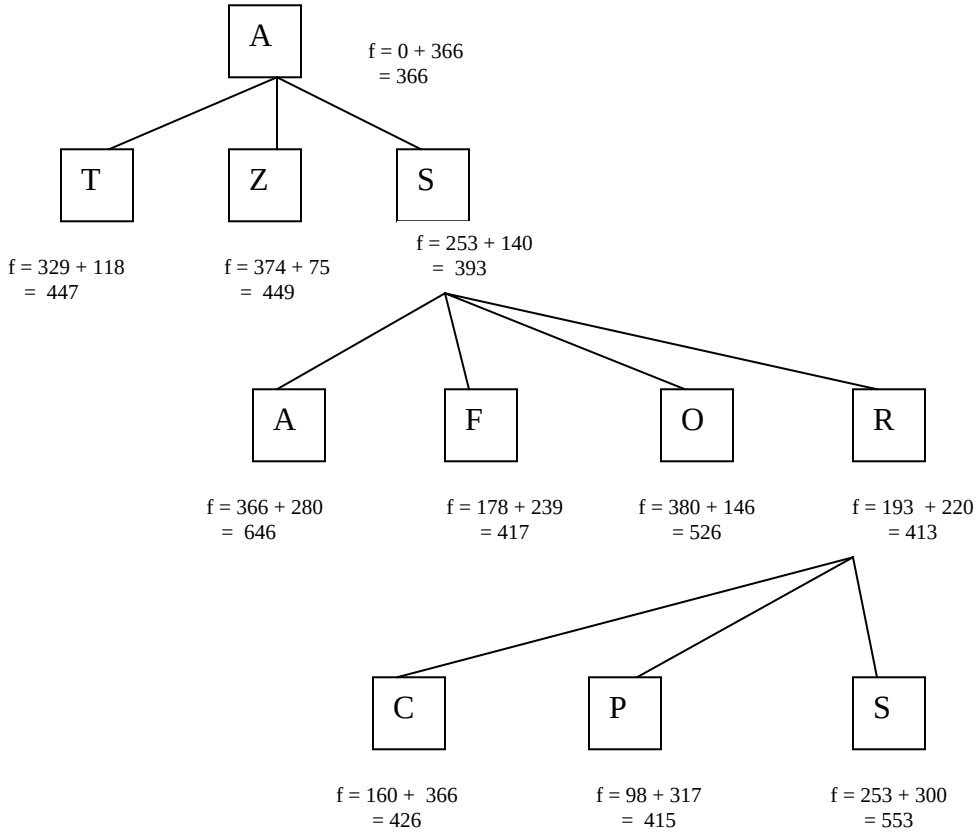
Trong đó, khỏang cách thực sự giữa các thành phố được đánh nhãn trên bản đồ và khỏang cách đường chim bay giữa các thành phố đến thành phố B được liệt kê như bảng

Khỏang các đường chim bay từ các thành phố đến thành phố B	
A	366 Km
B	0 Km
C	160 Km
D	242 Km
E	161 Km
F	178 Km
G	77 Km
H	151 Km
I	226 Km
L	244 Km
M	241 Km
N	234 Km
O	380 Km
P	98 Km
R	193 Km
S	253 Km
T	329 Km
U	80 Km
V	199 Km
Z	374 Km

Giải thuật tìm kiếm sử dụng thông tin heuristic tìm đường đi ngắn nhất từ thành phố A đến thành phố B được mô tả như hình vẽ



Giải thuật tìm kiếm sử dụng thông tin hàm đánh giá heuristic tìm đường đi ngắn nhất từ thành phố A đến thành phố B được mô tả như hình vẽ



Lời giải của bài tóan có thể được tìm thấy là

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array} \qquad \begin{array}{r} 9567 \\ +1085 \\ \hline 10652 \end{array}$$

Bước này đưa hệ thống vào thou nghiệm trong các trường hợp thực kế để rút ra kết luận đánh giá chất lượng vận hành của hệ thống đáng tin cậy hoặc chưa đáng tin cậy.

Ví dụ 1 : Thiết kế hệ chuyên gia suy diễn tiến cố vấn sinh viên học tập.

+ Định nghĩa vấn đề : Bài tóan đặt ra ra là thiết kế hệ chuyên gia suy diễn tiến cố vấn sinh viên học tập giải quyết các vấn đề như sau :

1- Giải quyết các môn học mà sinh viên đã thi đậu cho qua.
2- Xử lý các môn học mà sinh viên được đặt cách cho qua.
3- Xử lý các môn học có các môn học tiên quyết.
4- Xử lý các môn học mà sinh viên được phép đăng ký học trong mỗi học kỳ.

+ Định nghĩa dữ liệu vào : Dữ liệu vào của bài tóan gồm có

1- Các môn học bắt buộc.
2- Các môn học tự chọn.
3- Các môn học có các môn học tiên quyết.
4- Các môn học mà sinh viên đã học xong.
5- Các môn học cho phép sinh viên được đăng ký trong mỗi học kỳ.

+ Cấu trúc điều khiển dữ liệu suy diễn tiến của hệ thống : Để xử lý số liệu vào ra của hệ thống, cơ sở luật của hệ thống được thiết lập gồm các luật là

Luật 1 : Nếu X là môn học mà sinh viên đã thi đậu cho qua thì sinh viên đã học xong môn học với X.

Luật 2 : Nếu X là môn học mà sinh viên đã được đặt cách cho qua thì sinh viên đã học xong môn học với X.

Luật 3 : Nếu sinh viên đã học xong môn học với X và Q là danh sách chứa các môn học mà sinh viên đã học xong thì Q chứa X.

Luật 4 : Nếu X có môn học tiên quyết Y thì môn học tiên quyết của X là Y.

Luật 5 : Nếu X có môn học tiên quyết Y và Y có môn học tiên quyết Z thì môn học tiên quyết của X là Z.

Luật 6 : Nếu môn học tiên quyết của X là Y và P là danh sách chứa các môn học tiên quyết thì của X P phải chứa Y.

Luật 7 : Nếu Q là danh sách chứa các môn học mà sinh viên đã học xong với X, P là danh sách chứa các môn học tiên quyết của X và P là tập con của Q thì sinh viên đã học xong tất cả với các môn học học tiên quyết của X.

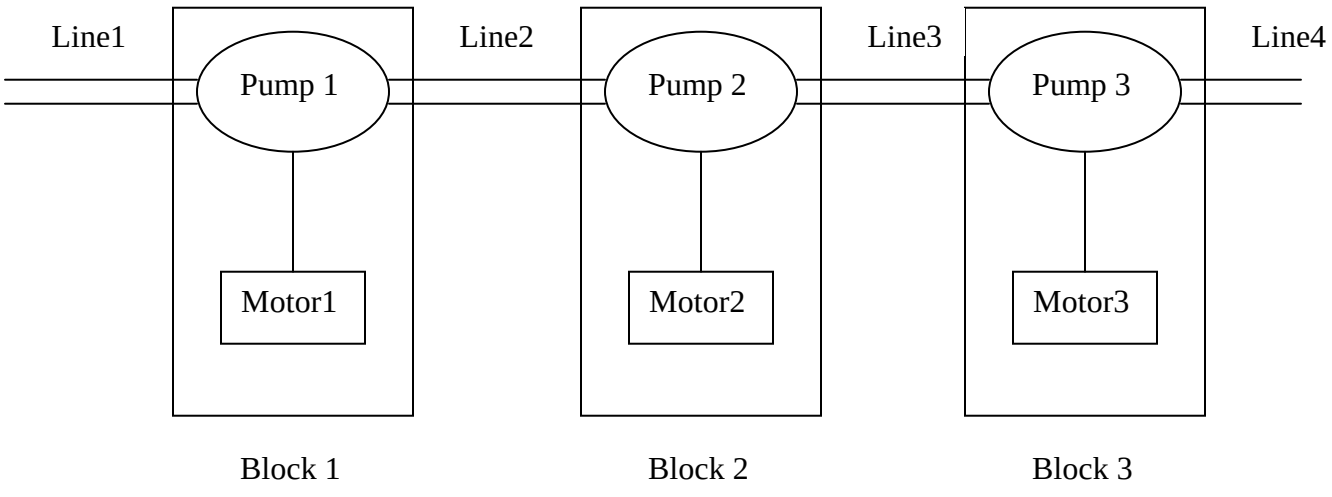
Luật 8 : Nếu X là môn học bắt buộc, sinh viên chưa học xong với X, sinh viên đã học xong tất cả với các môn học tiên quyết của X và X là môn học cho phép sinh viên đăng ký học trong học kỳ thì cho phép sinh viên đăng ký môn học với X.
Luật 9 : Nếu X là môn học tự chọn, sinh viên chưa học xong với X , sinh viên đã học xong tất cả với các môn học tiên quyết của X và X là môn học cho phép sinh viên đăng ký học trong học kỳ thì cho phép sinh viên đăng ký môn học với X.
+ Mã hóa cơ sở tri thức : sau đây là một ví dụ điển hình mã hóa cơ sở tri thức gồm cơ sở dữ liệu và cơ sở luật.

- Các môn học bắt buộc được mã hóa bằng ký hiệu điển hình là
  req("intro to computing").
  req("data structures").
  req("assembler").
  req("operating systems").
  Vân và vân vân.
- Các môn học tự chọn được mã hóa bằng ký hiệu điển hình là
  elec("information systems").
  elec("compilers").
  elec("algorithm analysis").
  Vân và vân vân.
- Các môn học tiên quyết được mã hóa bằng ký hiệu điển hình là
  impreq("data structures","intro to computing").
  impreq("calculus 2","calculus 1").
  impreq("operating systems","assempler").
  Vân và vân vân.
- Các môn học cho phép sinh viên được phép đăng ký học trong học kỳ được mã hóa bằng ký hiệu điển hình là
  given_now("intro to computing").
  given_now("calculus 2").
  Vân và vân vân.
- Các môn học mà sinh viên đã được đặt cách cho qua và thi đậu cho được mã hóa bằng ký hiệu điển hình là
  waived("intro to computing").
  waived("calculus 1").
  passed("data structures").
  passed("assempler").
  passed("calculus 2").

Vân và vân vân.

- Luật 1 được mã hóa bằng ký hiệu điển hình là
  if passed(X) then done_with(X).
- Luật 2 được mã hóa bằng ký hiệu điển hình là
  if waived(X) then done_with(X).
- Luật 3 được mã hóa bằng ký hiệu điển hình là
  if findall(Y, done_with(Y),X)) then all_done_with(X).
- Luật 4 được mã hóa bằng ký hiệu điển hình là
  if impreq(X,Y) then preq(X,Y).
- Luật 5 được mã hóa bằng ký hiệu điển hình là
  if impreq(X,Y) and preq(Y,Z) then preq(X,Z).
- Luật 6 được mã hóa bằng ký hiệu điển hình là
  if findall(Y,preq(X,Y),Z) then all_preq_for(X,Z).
- luật 7 được mã hóa bằng ký hiệu điển hình là
  if all_preq_for(X,Z) and all_done_with(Q) and subset(Z,Q)
  then have_preq_for(X).
- Luật 8 được mã hóa bằng ký hiệu điển hình là
  if req(X) and not(done_with(X)) and given_now(X) and
  have_preq_for(X) then pos_req_course(X).
- Luật 9 được mã hóa bằng ký hiệu điển hình là
  if elec(X) and not(done_with(X)) and given_now(X) and
  have_preq_for(X) then pos_elec_course(X).

Ví dụ 2 : Thiết kế hệ chuyên gia suy diễn tiến để giải quyết bài tóan chẩn đóan sự cố trên một trạm bơm nước như hình vẽ



Trạm bơm nước gồm có ba khối liên kết nhau qua các đường ống, trong đó mỗi khối có một máy bơm và một motor.

Vấn đề đặt ra của bài tóan là phát hiện sự cố vận hành nước trên trạm, nhận dạng khối có sự cố và chẩn đóan các thành phần bị hỏng gây ra sự cố như máy bơm, motor hoặc đường ống bị rỉ nức.

+ Điều kiện phát hiện sự cố vận hành nước trên trạm đó là áp suất ra trên các đường ống của trạm là thấp hơn áp suất vận hành nước bình thường.

+ Điều kiện khối có sự cố vận hành nước trên trạm là áp suất vào trên các đường ống của khối là bình thường trong khi đó áp suất ra trên các đường ống của khối là thấp hơn áp suất bình thường.

+ Các thành phần của khối là motor, máy bơm và đường ống. Khi đã xác định được khối có sự cố vận hành nước thì một trong các thành phần này của khối là nhân tố quyết định gây ra sự cố.

- Điều kiện motor vận hành yếu đó là nguyên nhân gây ra sự cố thì khi đó chỉ số vận hành motor được đọc về từ cảm biến là thấp.
- Điều kiện máy bơm bị hỏng đó cũng là nguyên nhân gây ra sự cố thì khi đó áp suất được đọc về từ cảm biến là áp suất vào của khối bằng áp suất ra của khối.
- Điều kiện đường ống bị rỉ nức đó là nguyên nhân gây ra sự cố thì khi đó áp suất được đọc về từ cảm biến là áp suất vào của khối phải là nhỏ hơn áp suất ra của khối.

Từ việc phân tích bài tóan nói trên, dữ liệu vào của hệ thống là áp suất và chỉ số vận hành của motor.

Giả sử áp suất vận hành nước bình thường của các đường ống là

+ Line 1 = 50 psi
+ Line 2 = 100 psi
+ Line 3 = 150 psi
+ Line 4 = 200 psi

và ta cũng giả sử rằng chỉ số vận hành bình thường của các motor là motor1 = motor2 = motor3 = 1.

Trên cơ sở đó, cấu trúc điều khiển dữ liệu vào ra của hệ thống xử lý hai khối trên trạm gồm các luật được thiết lập như sau :

Luật 1 : if line1 < 50 then line1 = low.

Luật 2 : if line1 >= 50 then line1 = normal.

Ví dụ : Thiết kế hệ chuyên gia suy diễn lùi tư vấn tài chánh bao gồm các công việc được mô tả như sau :

1) Định nghĩa vấn đề : Bài tóan tư vấn khách hàng về tài chánh nên đầu tư số tiền của họ vào một trong các loại phần vốn đầu tư như tiết kiệm, chứng khóan, công trái và tiết kiệm và thị trường chứng khóan điều đó phải phụ thuộc vào tình trạng bản thân và tình trạng tài chánh của khách hàng. Tình trạng bản thân của khách hàng phụ thuộc vào độ tuổi, công việc làm và tình trạng gia đình. Tình trạng tài chánh của khách hàng phụ thuộc vào tài sản hiện có và tình trạng gia đình. Như vậy, hai nhân tố quan trọng nhất để dẫn đến đích của bài tóan tư vấn khách hàng đầu tư phần tiền của họ vào các khỏan đầu tư đó là tình trạng bản thân và tình trạng tài chánh của khách hàng là ổn định hoặc không ổn định.
2) Định nghĩa các đích của bài tóan : Với bài tóan này, các đích của bài tóan được định nghĩa là
   + Phần vốn đầu tư loại 1 : 100% đầu tư vào tiết kiệm
   + Phần vốn đầu tư loại 2 : 60% thị trường chứng khóan, 30% thị trường công trái và 10% tiết kiệm.
   + Phần vốn đầu tư loại 3 : 20% thị trường chứng khóan, 40% thị trường công trái và 40% tiết kiệm.
   + Phần vốn đầu tư loại 4 : 100% đầu tư vào thị trường chứng khóan.
3) Thiết kế các luật suy diễn đích : Luật suy diễn đích của hệ chuyên gia suy diễn lùi là luật có cấu trúc nhìn từ đích lùi về dữ liệu. Điều đó có nghĩa là đích cuối cùng của bài tóan phải đượcđịnh nghĩa, từ đó nhìn về các nhân tố quyết định để dẫn đến đích và các nhân tố quyết định này được xem như là các đích mới để nhìn về các nhân tố quyết định khác dẫn đến đích mới này. Thủ tục thiết kế luật này được lặp lại cho đến khi nhân tố quyết định là ngõ vào từ dữ liệu ban đầu của bài tóan.

Hệ thống luật móc xích suy diễn đích để giải bài tóan tư vấn khách hàng về tài chánh được thiết lập là :

Luật 1 : Nếu số tiền của khách hàng là nhỏ hơn 1000 dollars thì tư vấn khách hàng nên đầu tư 100% số tiền của họ vào phần vốn đầu tư tiết kiệm.

Luật 2 : Nếu tình trạng bản thân của khách hàng là không ổn định và tình trạng thài chánh của khách hàng là không ổn định thì tư vấn khách hàng nên đầu tư 100% số tiền của họ vào phần vốn đầu tư tiết kiệm.

Luật 3 : Nếu tình trạng bả thân của khách hàng là không ổn định và tình trạng thài chánh của khách hàng là ổn định thì tư vấn khách hàng nên đầu 60% số tiền của họ

vào phần vốn đầu tư chứng khóan, 30% số tiền của họ vào phần vốn đầu tư công trái và 10% số tiền của họ vào phần vốn đầu tư tiết kiệm.

Luật 4 : Nếu tình trạng bản thân của khách hàng là ổn định và tình trạng tài chánh của khách hàng là không ổn định thì tư vấn khách hàng đầu tư 20% số tiền của họ vào phần vốn đầu tư chứng khóan, 40% số tiền của họ vào phần vốn đầu tư công trái và 40% số tiền của họ vào phần vốn đầu tư tiết kiệm.

Luật 5 : Nếu tình trạng bản thân của khách hàng là ổn định và tình trạng thài chánh của khách hàng là ổn định thì tư vấn khách hàng nên đầu tư 100% số tiền của họ vào phần vốn đầu tư chứng khóan.

Luật 6 : Nếu tuổi của khách hàng là lớn tuổi hoặc việc làm của khách hàng là không ổn định thì tình trạng bản thân của khách hàng là không ổn định.

Luật 7 : Nếu tuổi của khách hàng là trẻ tuổi vàviệc làm của khách hàng là ổn định và khách hàng có trẻ con thì tình trạng bản thân của khách hàng là không ổn định.

Luật 8 : Nếu tuổi của khách hàng là trẻ và việc làm của khách hàng là ổn định và khách hàng không có trẻ con thì tình trạng bản thân của khách hàng là ổn định.

Luật 9 : Nếu tuổi của khách hàng là lớn hơn 40 thì tuổi của khách hàng là lớn tuổi.

Luật 10 : Nếu tuổi của khách hàng là nhỏ hơn 40 thì tuổi của khách hàng là trẻ tuổi.

Luật 11 : Nếu thời gian hợp đồng làm việc của khách hàng là hơn 10 năm thì việc làm của khách hàng là ổn định.

Luật 12 : Nếu thời gian hợp đồng làm việc của khách hàng là từ 3 năm đến 10 năm và mức độ sa thải là thấp thì việc làm của khách hàng là ổn định.

Luật 13 : Nếu thời gian hợp đồng làm việc của khách hàng là từ 3 năm đến 10 năm và mức độ sa thải là cao thì việc làm của khách hàng là không ổn định.

Luật 14 : Nếu thời gian hợp đồng làm việc của khách hàng là ít hơn 3 năm thì việc làm của khách hàng là không ổn định.

Luật 15 : Nếu tổng số tài sản của khách hàng là nhỏ hơn tổng số nợ của khách hàng thì tình trạng tài chánh của khách hàng là không ổn định.

Luật 16 : Nếu tổng số tài sản của khách hàng là lớn hơn tổng số nợ của khách hàng và nhỏ hơn 2 lần tổng số nợ của khách hàng và khách hàng có trẻ con thì tình trạng tài chánh của khách hàng là không ổn định.

Luật 17 : Nếu tổng số tài sản của khách hàng là lớn hơn tổng số nợ của khách hàng thì tình trạng tài chánh của khách hàng là ổn định.

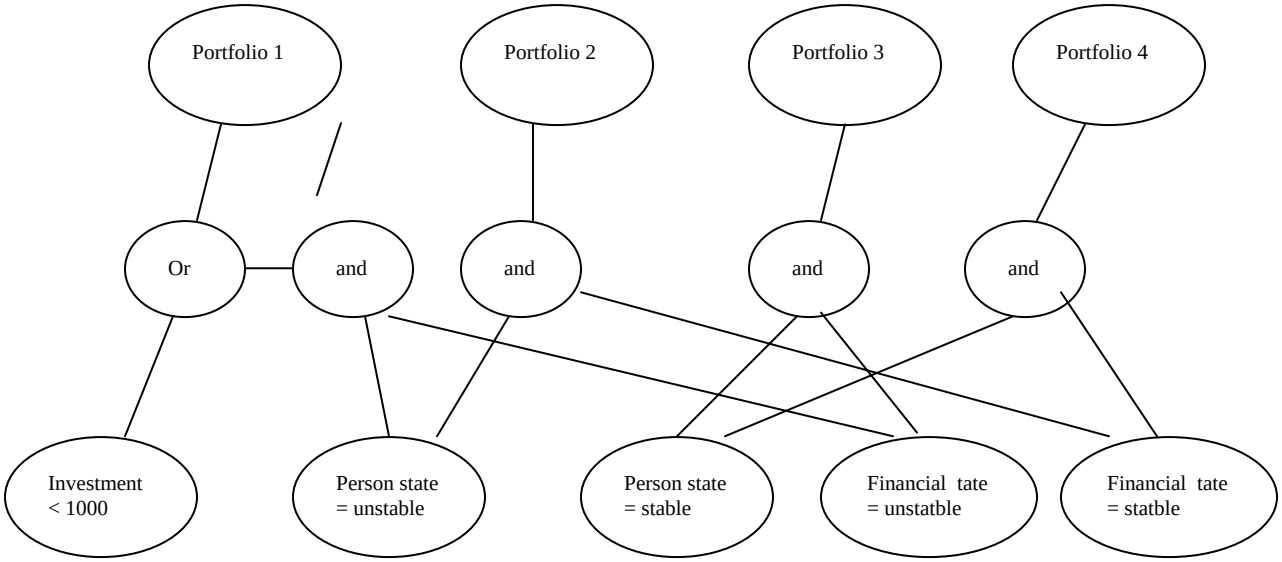
Chạy hệ chuyên gia này với các số liệu vào là

Số tiền đầu tư : 5000 dollars

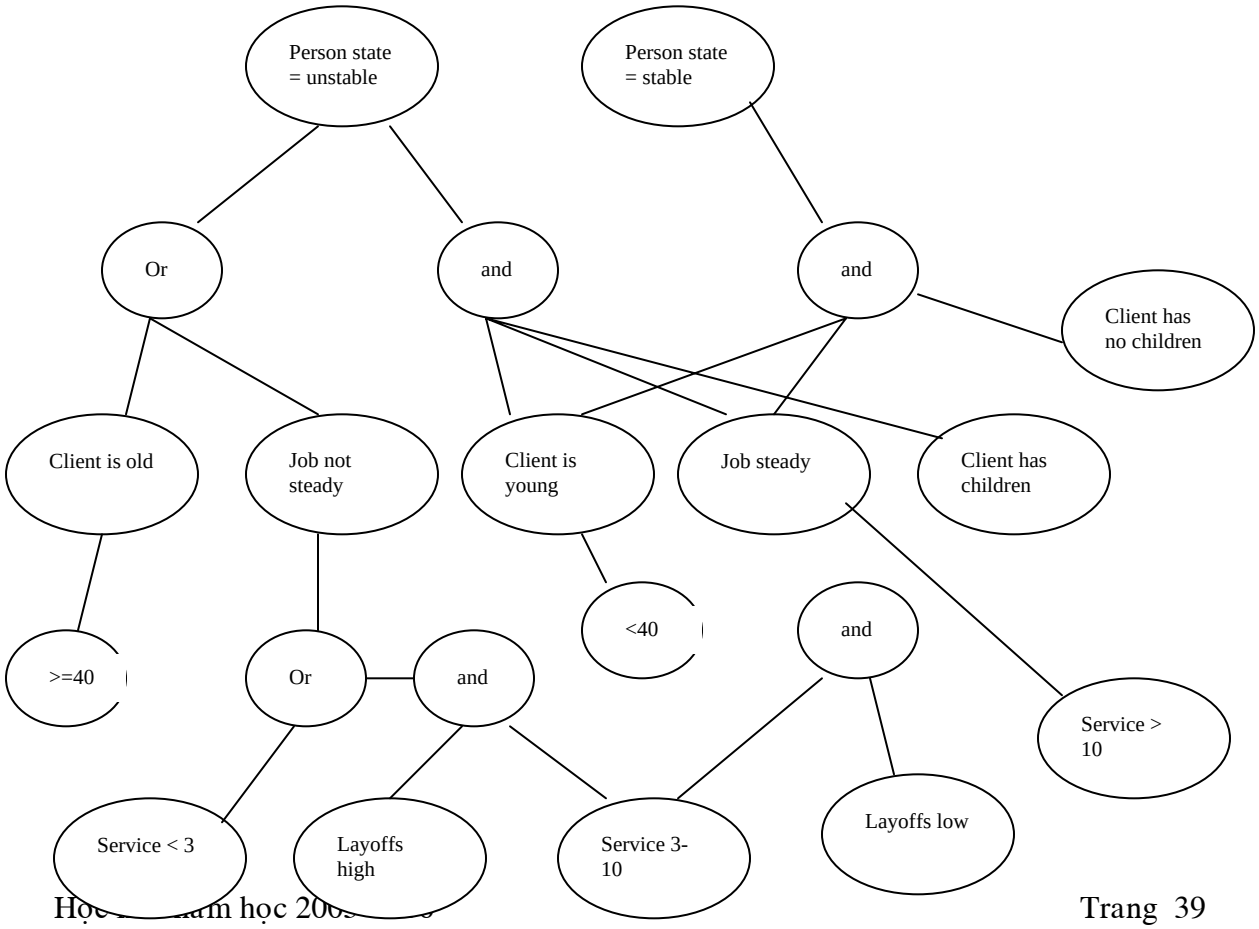
Tuổi của khách hàng : 30

Thời gian hợp đồng làm việc : 5 năm
Có trẻ con không : Có
Tổng số tài sản : 100000 dollars
Tổng số nợ : 20000 dollars.

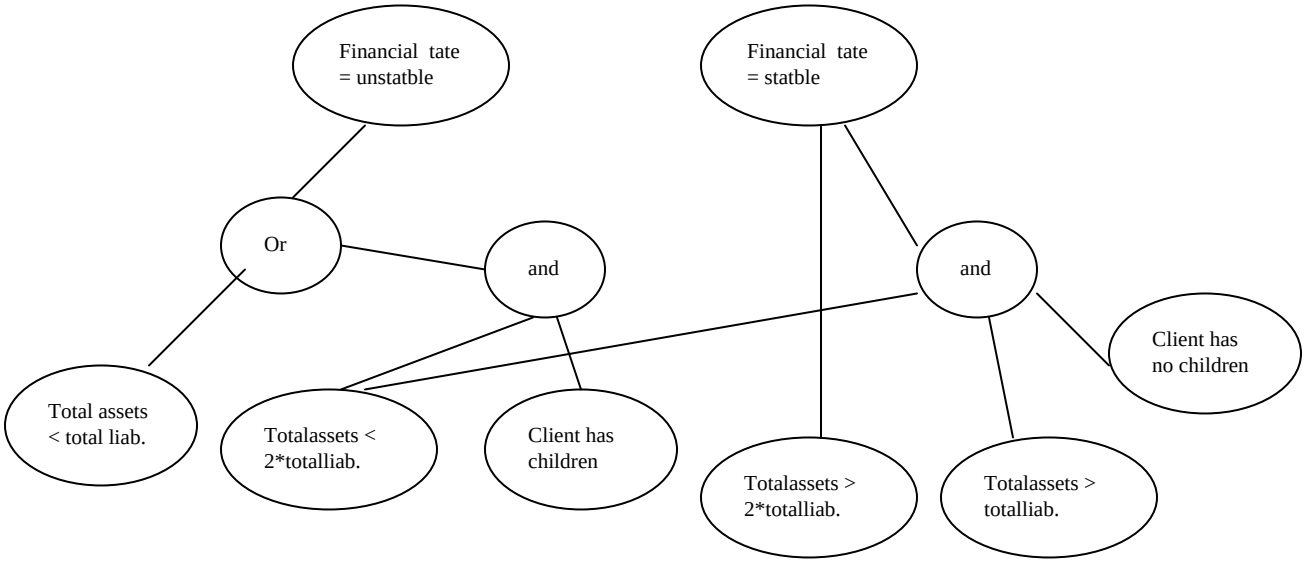
Mạng suy diễn luật đích của hệ chuyên gia suy diễn lùi tư vấn tài chánh được mô tả như hình



Mạng suy diễn tình trạng bản thân của khách hàng được mô tả như hình



Mạng suy diễn tình trạng trạng tài chánh của khách hàng được mô tả như hình



- Ở biểu thức dạng P∨Q, trong P và Q được gọi là các giới từ.
- Ở biểu thức dạng P→Q, trong đó P được gọi là tiền điều kiện và Q được gọi là kết luận.

Trong câu logic đề xuất, các ký hiệu ( ) và [ ] được sử dụng để nhóm các biểu thức con trong câu.

Ví dụ : Cho công thức ((P∧Q) →R) = ¬P∨¬Q∨R đó là một dạng câu hòan thiện, bởi vì :

- P, Q và R là các đề xuất và vì thế chúng là các câu hòan thiện.
- P∧Q là liên từ của hai câu và vì thế nó là một câu hòan thiện.
- (P∧Q) →R là kéo theo của một câu cho một câu khác và vì thế nó là một câu.
- ¬P và ¬Q là phủ đinh của câu và vì thế chúng là câu.
- ¬P∨¬Q là giới từ của hai câu và vì thế nó là một câu hòan thiện.
- ¬P∨¬Q∨R là giới từ của hai câu và vì thế nó cũng là một câu hòan thiện.
- ((P∧Q) →R) = ¬P∨¬Q∨R là sự tương của hai câu và vì thế nó là một câu hòan thiện.

+ Ngữ nghĩa của logic đề xuất : Ngữ nghĩa của logic đề xuất đó chính là giá trị chân lý của các ký hiệu đề xuất. Giá trị chân lý đúng của một đề xuất được ký hiệu là T và giá trị chân lý sai của một đề xuất được ký hiệu là F.

- Giá trị chân lý của phủ định ¬, ¬P là F nếu P là T và ¬P là T nếu P là F.
- Giá trị chân lý của liên từ ∧ , là T chỉ khi nào giá trị chân lý của cả hai thành phần của nó là T; mặt khác giá trị chân lý của nó là F.
- Giá trị chân lý của giới từ ∨ , là F chỉ khi nào giá trị chân lý của cả hai thành phần của nó là F; mặt khác giá trị chân lý của nó là T.
- Giá trị chân lý của phép kéo theo →, là F nếu giá trị chân lý của vế tiền điều kiện là T và giá trị chân lý của vế kết luận là F; mặt khác giá trị chân lý của nó là T.
- Giá trị chân lý của phép tương đương ↔, là T chỉ khi nào hai thành phần của nó là có cùng giá trị chân lý; mặt khác giá trị chân lý của nó là F.

Cho P, Q và R là các biểu thức đề xuất, các biểu thức sau đây là các biểu thức logic tương đương :

- ¬(¬P) = P.
- (P∨Q) = (¬P→Q).

phép tóan logic của logic vị từ và logic đề xuất là giống nhau. Sự khác nhau giữa hai lọai logic này là ký hiệu vị từ và ký hiệu đề xuất.

Ký hiệu vị từ gồm có hằng vị từ, biến vị từ, hàm vị từ, vị từ và vị từ định lượng.

- Hằng vị từ : là chuỗi của các chữ cái in thường dùng để biểu diễn tên riêng hoặc thuộc tính riêng của đối tượng.

Ví dụ : john, tree, tall, blue là các ký hiệu hằng vị từ hợp lệ.

- Biến vị từ : là chuỗi của các chữ cái với ít nhất chữ cái đầu tiên của chuỗi phải là chữ cái in hoa dùng để biểu diễn lớp của các đối tượng.

Ví dụ : X là biến vị từ dùng để biểu diễn lớp của các đối tượng ngày trong tuần hoặc Breaker là biến vị từ dùng để biểu diễn lớp của các đối tượng máy cắt điện.

- Hàm vị từ : là ánh xạ từ một hoặc nhiều phần tử của tập hợp này đến một phần tử duy nhất trong một tập hợp khác. Hàm phải có tên riêng và các đối số vào của nó. Tên của hàm vị từ được qui ước là chuỗi của các chữ cái in thường. Cú pháp tổng quát của hàm là

Tên_hàm(Các đối số vào của hàm).

Hàm nhận các đối số vào từ một tập hợp này và trả về duy nhất một đối số ra trong một tập hợp khác.

Ví dụ : Cho đề xuất là

George is father of David.

Đề xuất này có thể được biểu diễn bằng hàm vị từ father là

father(david).

Hàm trả về giá trị ra của nó là george.

Cho một đề xuất khác là

2 plus 3 is equal to 5.

Đề xuất này có thể được biểu diễn bằng hàm vị từ plus là

plus(2,3).

Hàm sẽ trả về giá trị ra là 5.

Các đối số của hàm vị từ có thể là hằng vị từ hoặc biến vị từ.

- Vị từ : Vị từ là một dạng đặc biệt của hàm vị từ. Vị từ cũng phải có tên riêng và các đối số vào của nó. Tên vị từ thường là mối quan hệ giữa hai hoặc nhiều đối tượng trong một đề xuất. Qui ước đặt tên cho vị từ cũng

giống như hàm vị từ. Vị từ nhận các đối số vào từ một tập hợp và trả về đối số ra trong tập hợp giá trị chân lý đúng T hoặc sai F.

Ví dụ : Cho đề xuất là

George likes Kate.

Đề xuất này có thể được biểu diễn bằng vị từ likes là

likes(george,kate).

Vị từ likes sẽ trả về logic true(T) nếu George thích Kate; mặt khác vị từ trả về giá trị logic false(F).

Các đối số của vị từ có thể là hằng vị từ, biến vị từ hoặc hàm vị từ.

Ví dụ : Cho đề xuất là

X likes Kate, trong đó X là lớp của các đối tượng đàn ông.

Đề xuất này có thể được biểu diễn bằng vị từ likes là

likes(X, kate).

Vị từ sẽ trả về giá trị logic true (T) nếu that sự tất cả những người đàn ông thích Kate; mặt khác vị từ trả về giá trị logic false (F).

- Vị từ định lượng : Khi các đối số vào của hàm vị từ hoặc vị từ là biến vị từ khi đó để xác định phạm vi giá trị của biến, hai đại lượng đứng trước biến đó là $\forall$ và $\exists$, hai đại lượng này được gọi là các vị từ định lượng.

Vị từ định lượng $\forall$ đứng trước biến để xác định biểu thức là đúng cho mọi giá trị của biến.

Vị từ định lượng $\exists$, đứng trước biến để xác định biểu thức là đúng co một vài giá trị của biến.

Ví dụ : Cho đề xuất là

All humans are mortal.

Đề xuất này có thể được biểu diễn là

$(\forall X)$ (human(X) $\rightarrow$ mortal(X)).

Cho một đề xuất khác là

There is a student who is smart.

Đề xuất này có thể được biểu diễn là

$(\exists X)$(student(X) $\wedge$smart(X)).

Ví dụ ứng dụng : Sau đây là ví dụ minh chứng dùng logic vị từ biểu diễn cơ sở tri thức của bài tóan tư vấn tài chánh.

1. saving_account(inadequate) $\rightarrow$ investment(savings).
2. savings_count(adequate)$\wedge$income(adequate)$\rightarrow$investment(stocks).

$\neg(P \wedge Q) = \neg P \vee \neg Q$
$\neg(P \vee Q) = \neg P \wedge \neg Q.$
$\neg(\exists X)\ P(X) = (\forall X)\neg P(X).$
$\neg(\forall X)\ P(X) = (\exists X)\neg P(X).$

Bước 2 : Để chứng minh đề xuất P là đúng, ta giả sử P là sai, điều đó có nghĩa là phủ định của P là đúng và cộng phủ định này vào tập các tiên đề.

Bước 3 : Đưa tất cả các vị từ định lượng về đứng trước các mệnh đề và loại bỏ chúng khỏi tập các tiên đề.

Bước 4 : Chọn cặp mệnh đề, một có chứa P và một mệnh đề khác chứa ¬P, khử bỏ cặp phân tử này, vì chúng có mâu thuẩn. Các phần tử còn lại của hai mệnh đề hợp nhau tạo thành mệnh đề mới.

Bước 5 : Lặp lại bước 4 cho đến khi nào có mệnh đề rỗng xuất hiện thì dừng thủ tục chứng minh. Mệnh đề rỗng là mệnh đề không chứa bất kỳ một phân tử nào. Điều này chứng tỏ rằng có sự mâu thuẩn giữa tiên đề đã giả sử và tập các tiên đề, do đó ta đi đến kết luận rằng đề xuất P là tương thích với tập các tiên đề và phủ định của đề xuất P là không tương thích với tập các tiên đề.

Ví dụ : Cho tập các tiên đề là

1) All dogs are animals.
2) Fido is a dog.
3) All animals will die.

Hãy chứng minh rằng Fido will die bằng phương pháp hợp gải ?

Giải

Các tiên đề được biểu diễn bằng logic vị từ ở dạng hòan thiện là

1) $\forall(X)\ (dog(x) \rightarrow animal(X))$.
2) dog(fido).
3) $\forall(Y)(animal(Y) \rightarrow die(Y))$.

Để chứng minh đề xuất Fido will die là đúng, giả sử Fido wil die là sai và do đó phủ định của nó là đúng đó là ¬die(fido). Cho tiên đề này là tiên đề 4 và cộng nó vào tập các tên đề.

1) $\forall(X)\ (dog(x) \rightarrow animal(X))$.

nhãn để chỉ các quan hệ hoặc các tính chất giữa các đối tượng. Với cách biểu diễn này được xem như một cấu trúc dữ liệu mà các ràng buộc vốn sẵn có trong bài tóan có thể được vạch ra và hướng giải quyết vấn đề của bài tóan bằng các luật suy diễn cũng có thể được thiết lập nhờ thông qua các đường mũi tên liên kết giữa các đối tượng. Vì thế mạng còn được gọi là mạng suy diễn tri thức.

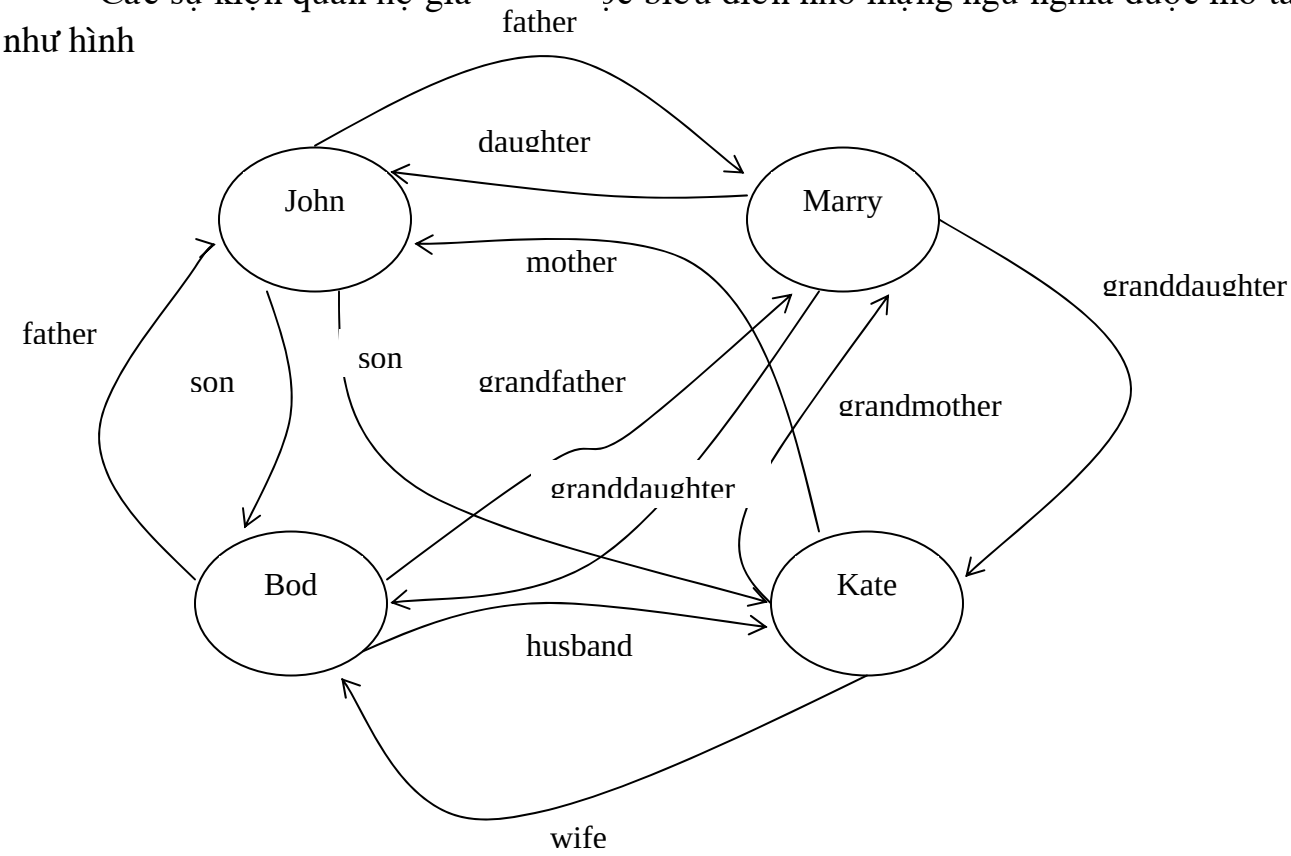
Ví dụ : Cho bài tóan quan hệ gia đình với các sự kiện là

1) John is father of Marry.
2) Mary is daughter of John.
3) Bod is father of John.
4) John is son of Bod.
5) Bod is husband of Kate.
6) Kate is wife of Bod.
7) John is son of Kate.
8) Bod is grandfather of Marry.
9) Kate is grandmother of Marry.
10) Marry is granddaughter of Bod and Kate.

Hãy biểu diễn các sự kiện quan hệ gia đình này nhờ mạng ngữ nghĩa ?

Giải :

Các sự kiện quan hệ gia đình được biểu diễn nhờ mạng ngữ nghĩa được mô tả như hình



Value : value of property_name_2>

Slot : <property_name_N>
Value : <value of property_name_N>.

Các slot trong mỗi frame chứa các thông tin như sau :

1) Thông tin nhận dạng frame.
2) Thông tin quan hệ của frame này với frame khác.
3) Các thành phần mô tả của frame.
4) Thông tin thủ thủ tục.
5) Thông tin mặc định frame.
6) Thông tin đề xuất mới.

Biểu diễn tri thức nhờ Frame cung cấp ý tưởng lập trình hướng đối tượng trong ngôn ngữ lập trình Prolog hoặc các ngôn ngữ lập trình khác như C++ và Visual Basic. Frame cho phép truy cập các thành phần của Frame đó là các slot và cho phép hưởng quyền thừa kế giữa Frame dữ liệu này và Frame dữ liệu khác.

Ví dụ : Cho mạng ngữ nghĩa biểu diễn các sự kiện về động vật như hình trên, các sự kiện này được tổ chức trong các khung như sau :

Frame animal
Slot : can
Value : breathe, move
Slot : has
Value : wings

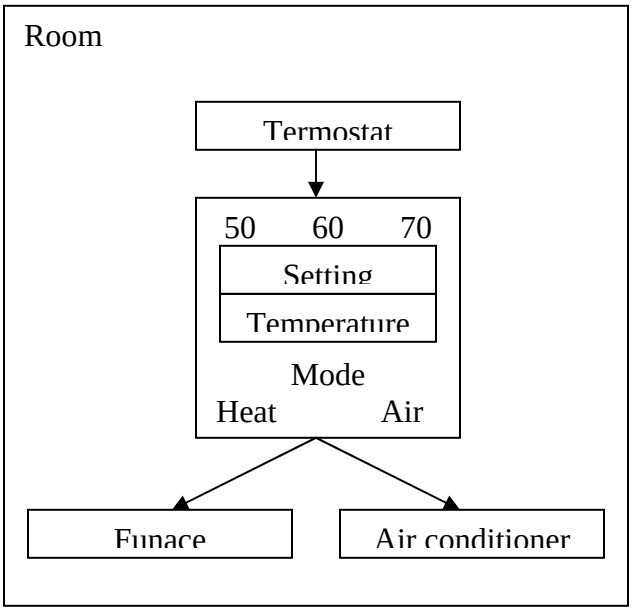
Frame bird
Slot : is_a
Value : animal
Slot : can
Value : fly
Slot : has
Value : wings, feathers

Frame canary
Slot : is_a
Value : bird
Slot : can
Value : sing
Slot : is
Value : yellow

Frame ostrich
Slot : is_a
Value : bird
Slot : can_not
Value : fly
Slot : is
Value : tail

Ví dụ ứng dụng : Thiết kế hệ chuyên gia điều khiển nhiệt độ môi trường trên cơ sở hệ thống Frame.

Xét hệ thống điều khiển nhiêt độ môi trường trong một căn nhà nhỏ gồm có ba phòng đó là phòng khách, phòng ngũ và phòng ăn, trong đó mỗi phòng có một nhiệt kế, một lò sưởi và một máy điều hòa có cấu trúc như hình.



Hệ thống gồm các frame như room, thermostat, funace và air conđioner được mô tả như sau :

Frame room
Slot : funace
Value : < funac1,funace2,funace3 >
Slot : air_conditioner
Value : < air_conditioner1,air_conditioner2, air_conditioner3 >
Slot : thermostat
Value : < thermostat1, thermost2, thermostat3 >
Slot : occupany
Value : < yes,no >

Frame thermostat
Slot : air_conditioner
Value : < air_conditioner1,air_conditioner2, air_conditioner3 >
Slot : funace
Value : < funac1,funace2,funace3 >
Slot : mode
Value : <heat,air>
Slot : setting
Value : 60
Slot : temperature
Value : 65
Slot : room
Value : <livingroom, bedroom, kitchenroom >

Frame air_conditioner
Slot : room
Value : <livingroom, bedroom, kitchenroom >
Slot : state
Value : <on,off>
Slot : thermostat
Value : < thermostat1,thermostat2, thermostat3 >

Frame funace
Slot : room
Value : <livingroom, bedroom, kitchenroom >
Slot : state
Value : <on,off>
Slot : thermostat
Value : < thermostat1,thermostat2, thermostat3 >

Cơ sở luật điều khiển nhiệt độ môi trường trong nhà trên cơ sở các thành phần của hệ thống frame được thiết lập gồm các luật là

Luật 1 : if (temperature < setting) and (funace state ≠ off) and (mode ≠ heat) and (roomoccupancy ≠ yes) then send message (funace state = on).

Luật 2 : if (temperature < setting - 5) and (funace state ≠ off) and (mode ≠ heat) and (roomoccupancy ≠ no) then send message (funace state = on).

Luật 3 : if (temperature >= setting) and (funace state ≠ on) and (mode ≠ heat) and (roomoccupancy ≠ yes) then send message (funace state = off).

Luật 4 : if (temperature >= setting - 5) and (funace state ≠ on) and (mode ≠ heat) and (roomoccupancy ≠ no) then send message (funace state = off).

Luật 5 : if (temperature < setting) and (air_conditioner state ≠ off) and (mode ≠ air) and (roomoccupancy ≠ yes) then send message (air_conditioner state = on).

Luật 6 : if (temperature < setting - 5) and (air_conditioner state ≠ off) and (mode ≠ air) and (roomoccupancy ≠ no) then send message (air_conditioner state = on).

Luật 7 : if (temperature >= setting) and (air_conditioner state ≠ on) and (mode ≠ air) and (roomoccupancy ≠ yes) then send message (air_conditioner state = off).

Luật 8 : if (temperature >= setting - 5) and (air_conditioner state ≠ on) and (mode ≠ air) and (roomoccupancy ≠ no) then send message (air_conditioner state = off).

hiện dưới dạng if condition_1 and condition_2, … and condition_N then conclusion với cú pháp tổng quát của Prolog là

conclusion : - condition_1, condition_2, …,condition_N.

+ goal : là vùng thực hiện các đích đề ra của bài tóan.

Ví dụ : Cho các sự kiện và luật suy diễn là

1) John likes wine.
2) Lance likes skiing.
3) Lance likes books.
4) Lance likes films.
5) If Z reads and Z is inquisitive then Z likes books.

Chương trình Prolog sau thể hiện các sự kiện, luật suy diễn để thỏa mãn hai thành phần đích con của John đó là John thích uống rượu, đọc sách và là người tìm tòi do đó john thích sách.

```
domains
        name, thing = symbol
predicates
        nondeterm likes(name, thing)
        reads(name)
        is_inquisitive(name)
clauses
        likes(john,wine).
        likes(lance,skiing).
        likes(lance,books).
        likes(lance,films).
        likes(Z,books) :-
        reads(Z),is_inquisitive(Z).
        reads(john).
        is_inquisitive(john).
goal
likes(X,wine),likes(X,books).
```

Cạy chương trình cho kết quả là

```
X = john.
1 solution.
```

```
append(List,List,List)
clauses
append([],List,List).
append([X|T],L,[X|NL]) :- append(T,L,NL).
goal
append([a,b,c],[d,e],Y).
```

Chạy chương trình này cho kết quả là Y = [“a”,”b”,”c”,”d”,”e”].

+ Hiển thị danh sách : Mệnh đề hiển thị hanh sách ra màn hình với clauses được thiết lập là

```
clauses
    writelist([]).
    writelist([H|T) :-  write(H),nl,writelist(T).
```

+ Đảo ngược thứ tự trong danh sách : mệnh đề đảo ngược thứ tự các phần tử trong danh sách với clauses được thiết lập là

```
clauses
    reverse_writelist([]).
    Reverse_writelist([H|T]) :- reverse_writelist(T),write(H),nl.
```

Ví dụ 1 : Cho bài tóan quan hệ gia đình với các sự kiện và các luật suy diễn là

1) John is son of Dan.
2) Mary is sister of Suzan.
3) Harold is brother of Larry.
4) John married Mary.
5) Larry married Sue.
6) If B is son of A then A is father of B.
7) If A is father of C and C is father of B then A is grandfather of B.
8) If A married C and C is sister of B then A is sister in law of B.
9) If A is brother of C and C married B then A is sister in law of B.

Chương trình Prolog là một ví dụ minh chứng giải quyết bài tóan mối quan hệ gia đình này.

```
database - tmp
  son(STRING,STRING)
  sister(STRING,STRING)
```

```
  brother(STRING,STRING)
  married(STRING,STRING)

clauses
  son("John","Dan").
  sister("Mary","Suzan").
  brother("Harold","Larry").
  married("John","Mary").
  married("Larry","Sue").

predicates
  father(STRING father,STRING child)
  grandfather(STRING grandfather,STRING grandchild)
 nondeterm sister_in_law(STRING,STRING)

clauses
  father(A,B):-
        son(B,A).

  grandfather(A,B):-
        father(A,C),
        father(C,B).

  sister_in_law(A,B):-
        married(A,C),
        sister(C,B).
  sister_in_law(A,B):-
        brother(A,C),
        married(C,B).

goal
  sister_in_law("John",Z),
  format(Msg,"sister_in_law(\"John\",%)",Z),
  write(Msg).
```

Ví dụ 2 : Chương trình Prolog sau là một ví dụ điển giải bài tóan người nông dân sắp xếp các chuyến thuyền qua lại sông.

```
domains
LOC   = east;
        west
 STATE = state(LOC farmer,LOC wolf,LOC goat,LOC cabbage)
 PATH  = STATE*
predicates
 go(STATE,STATE)              % Start of the algorithm
 path(STATE,STATE,PATH,PATH) % Finds a path from one state to another
 nondeterm move(STATE,STATE)  % Transfer a system from one side to
another
 opposite(LOC,LOC)            % Gives a location on the opposite side
 nondeterm unsafe(STATE)     % Gives the unsafe states
 nondeterm member(STATE,PATH) % Checks if the state is already visited
 write_path(PATH)
 write_move(STATE,STATE)

 clauses

 go(StartState,GoalState):-
        path(StartState,GoalState,[StartState],Path),
        write("A solution is:\n"),
        write_path(Path).

 path(StartState,GoalState,VisitedPath,Path):-
        move(StartState,NextState),                 % Find a move
        not(member(NextState,VisitedPath)),        % Check that we have not
had this situation before
        path(NextState,GoalState,[NextState|VisitedPath],Path),
        !.
 path(GoalState,GoalState,Path,Path).                % The final state is
reached
```

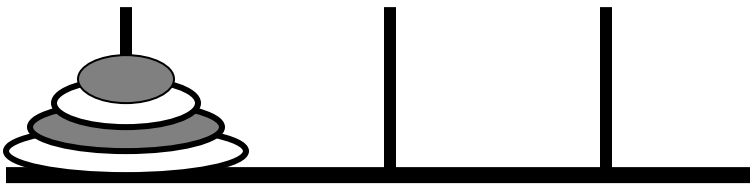
```
 move(state(X,X,G,C),state(Y,Y,G,C)):-
        opposite(X,Y),not(unsafe(state(Y,Y,G,C))).
move(state(X,W,X,C),state(Y,W,Y,C)):-
        opposite(X,Y),not(unsafe(state(Y,W,Y,C))).
move(state(X,W,G,X),state(Y,W,G,Y)):-
        opposite(X,Y),not(unsafe(state(Y,W,G,Y))).
move(state(X,W,G,C),state(Y,W,G,C)):-
        opposite(X,Y),not(unsafe(state(Y,W,G,C))).

opposite(east,west).
opposite(west,east).

unsafe(state(F,X,X,_)):-        % The wolf eats the goat
        opposite(F,X),
        !.
unsafe(state(F,_,X,X)):-        % The goat eats the cabbage
        opposite(F,X),
        !.

member(X,[X|_]):-
        !.
member(X,[_|L]):-
        member(X,L).

write_path([H1,H2|T]):-
        write_move(H1,H2),
        write_path([H2|T]).
write_path([]).

write_move(state(X,W,G,C),state(Y,W,G,C)):-
        !,
        write("The farmer crosses the river from ",X," to ",Y),
        nl.
write_move(state(X,X,G,C),state(Y,Y,G,C)):-
        !,
        write("The farmer takes the Wolf from ",X," of the river to ",Y),
```

```
        nl.
    write_move(state(X,W,X,C),state(Y,W,Y,C)):-
        !,
        write("The farmer takes the Goat from ",X," of the river to ",Y),
        nl.
    write_move(state(X,W,G,X),state(Y,W,G,Y)):-
        !,
        write("The farmer takes the cabbage from ",X," of the river to ",Y),
        nl.

goal
  go(state(east,east,east,east),state(west,west,west,west)),
  write("solved").
```

Chạy chương trình này cho kết quả là

```
A solution is:
The farmer takes the Goat from west of the river to east
The farmer crosses the river from east to west
The farmer takes the cabbage from west of the river to east
The farmer takes the Goat from east of the river to west
The farmer takes the Wolf from west of the river to east
The farmer crosses the river from east to west
The farmer takes the Goat from west of the river to east
no
```

Ví dụ 3 : Cho bài tóan tháp Hà Nội như hình vẽ



Mục tiêu của bài tóan là di chuyển tất cả các đĩa từ cột bên trái sang cột bên phải nhờ thông qua cột trung gian ở giữa mỗi lần di chuyển một đĩa không được phép đĩa lớn nằm trên đĩa nhỏ.

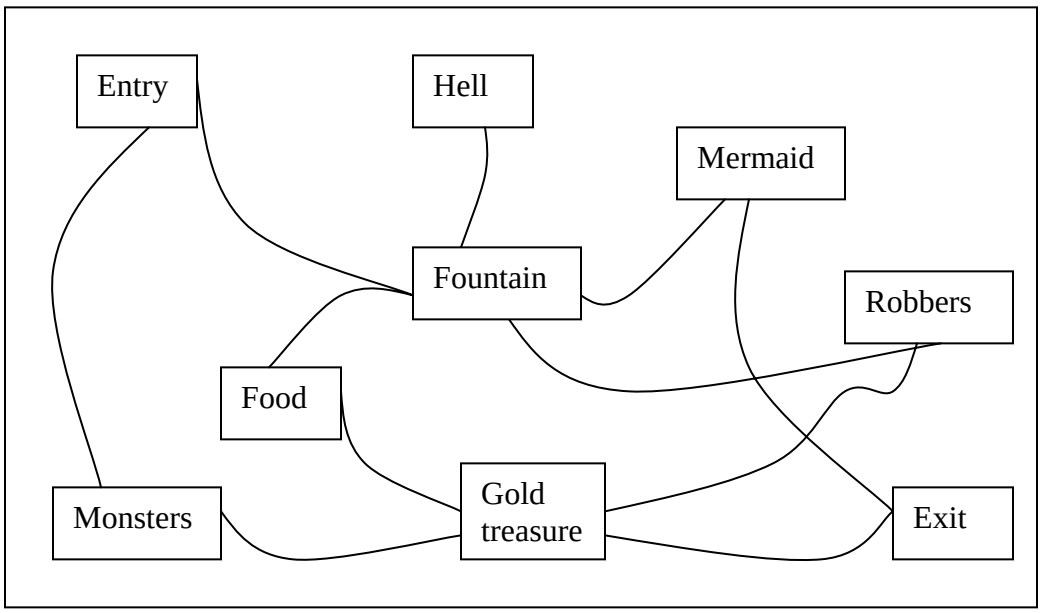
Chương trình Prolog sau là một ví dụ minh chứng giải bài tóan tháp Hà Nội này.

```
domains
  loc =right;middle;left

predicates
  hanoi(integer)
  move(integer,loc,loc,loc)
  inform(loc,loc)

clauses
  hanoi(N):-
        move(N,left,middle,right).
        move(1,A,_,C):-
        inform(A,C),
        !.

  move(N,A,B,C):-
        N1=N-1,
        move(N1,A,C,B),
        inform(A,C),
        move(N1,B,A,C).

  inform(Loc1, Loc2):-
        write("\nMove a disk from ", Loc1, " to ", Loc2).

GOAL
  hanoi(3).
```

Chạy chương trình này cho kết quả là

```
Move a disk from left to right
Move a disk from left to middle
Move a disk from right to middle
Move a disk from left to right
Move a disk from middle to left
Move a disk from middle to right
Move a disk from left to rightyes
```

Ví dụ 4 : Cho bản đồ chỉ đường đến kho báu chứa vàng như hình vẽ



Trên đường tìm đến kho báu chứa vàng phải qua các hang động nguy hiểm như monsters (quái vật) và robbers (những kẻ cướp).
Chương trình Prolog sau là một ví dụ minh chứng tìm đường an tòan đến hang động kho báu chứa vàng.

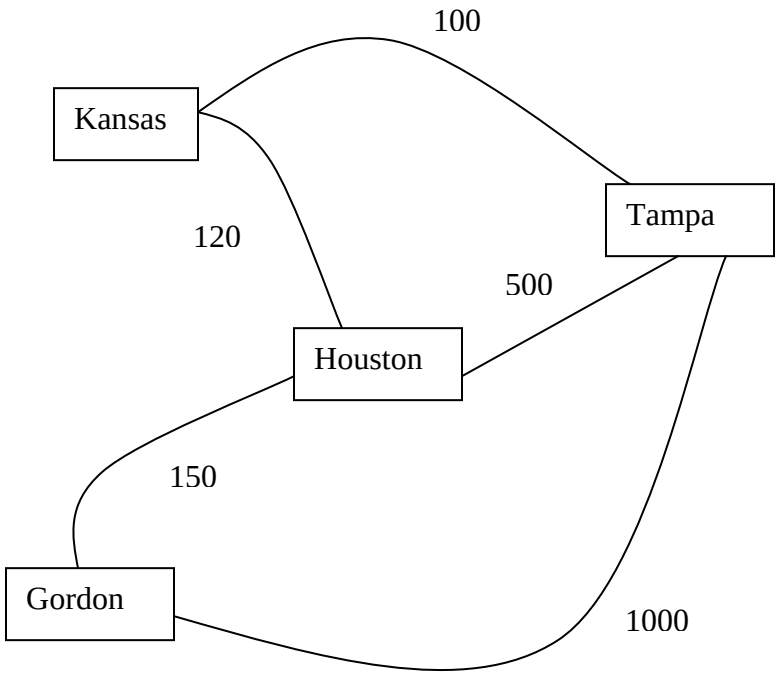
```
domains
        room = symbol
        roomlist = room*
predicates
        nondeterm gallery(room,room)
        nondeterm neighborroom(room,room)
        avoid(roomlist)
        nondeterm go(room,room)
        nondeterm route(room,room,roomlist)
        nondeterm member(room,roomlist)
clauses
        gallery(entry,monsters).
        gallery(entry,fountain).
        gallery(fountain,hell).
        gallery(fountain,food).
        gallery(exit,gold_treasure).
        gallery(fountain,mermaid).
```

```
        gallery(robbers,go_treasure).
        gallery(fountain,robbers).
        gallery(food,gold_treasure).
        gallery(mermaid,exit).
        gallery(monsters,gold_treasure).
        gallery(gold_treasure,exit).
        gallery(mermaid,gold_treasure).
        neighborroom(X,Y) :- gallery(X,Y).
        neighborroom(X,Y) :- gallery(Y,X).
        avoid([monsters,robbers]).
        go(Here,There) :- route(Here,There,[Here]).
        go(_,_).
route(Room,Room,VisitedRooms) :- member(gold_treasure,VisitedRooms),
                                 write(VisitedRooms),nl,fail.
route(Room,Way_out,VisitedRooms) :- neighborroom(Room,Nextroom),
                                 avoid(DangerousRooms),
                                 not(member(NextRoom,DangerousRooms)),
                                 not(member(NextRoom,VisitedRooms)),
route(NextRoom,Way_out,[NextRoom|VisitedRooms]).
member(X,[X|_]):-!.
  member(X,[_|L]):-
    member(X,L).
goal
go(entry,exit).
```

Chạy chương trình này cho kết quả là

```
["exit","gold_treasure","food","fountain","entry"]
["exit","gold_treasure","food","fountain","entry"]
yes
```

Ví dụ 5 : Cho bản đồ của các thành phố như hình vẽ



Chương trình Prolog sau là một ví dụ minh chứng giải bài tóan tìm đường đi ngắn nhất từ thành phố Gordon đến thành phố Tampa.

```
domains
  town     = symbol
  townlist = town*
  distance = integer

predicates
  nondeterm road(town,town,distance)
clauses
  road(tampa,houston,500).
  road(gordon,tampa,1000).
  road(houston,gordon,150).
  road(houston,kansas_city,120).
  road(tampa,kansas_city,100).

predicates
  nondeterm connected(town,town,distance)
clauses
  connected(X,Y,Dist):-
```

```
  road(X,Y,Dist).
 connected(X,Y,Dist):-
  road(Y,X,Dist).

predicates
 determ member(town,townlist)
clauses
 member(X,[X|_]):-!.
 member(X,[_|L]):-
  member(X,L).

predicates
 nondeterm route(town,town,townList,townList,distance)
clauses
 route(Town,Town,VisitedTowns, VisitedTowns, 0) :-
  !.
 route(Town1,Town2,VisitedTowns,ResultVisitedTowns,Distance):-
  connected(Town1,X,Dist1),
  not(member(X,VisitedTowns)),
  route(X,Town2,[X|VisitedTowns],ResultVisitedTowns,Dist2),
  Distance=Dist1+Dist2.

predicates
 showAllRoutes(town,town)
 write_rote(town FirstTown,townList,distance)
 reverse_list(townList InList, townList Tmp, townList Reversed)

clauses
 showAllRoutes(Town1,Town2):-
  write("All routes from ",Town1," to ",Town2," are:\n"),
  route(Town1,Town2, [Town1] ,VisitedTowns, Dist),
  write_rote(Town1,VisitedTowns,Dist),nl,
  fail.
 showAllRoutes(_,_).
 write_rote(Town1,[Town1|VisitedTowns],Dist):-
  !,
```

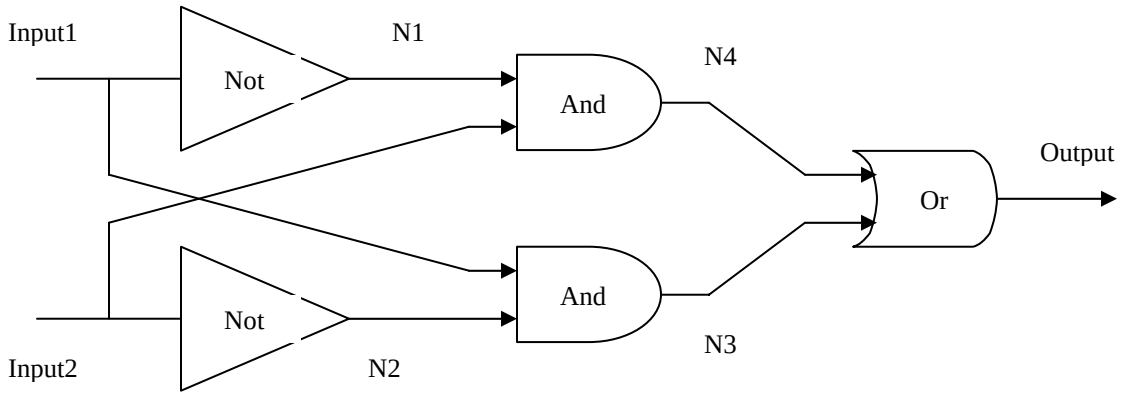
```
    Towns = [Town1|VisitedTowns],
    write("  ",Towns," --> ",Dist),nl.
  write_rote(_,VisitedTowns,Dist):-
    reverse_list(VisitedTowns, [], VisitedTowns_Reversed),
    write("  ",VisitedTowns_Reversed," --> ",Dist),nl.

  reverse_List([],LIST,LIST):-!.
  reverse_List([H|SeenListRest],Interm,SeenList):-
        reverse_List(SeenListRest,[H|Interm],SeenList).

predicates
  showShortestRoutes(town,town)
  determ shorterRouteExist(town,town,distance)

clauses
  showShortestRoutes(Town1,Town2):-
    write("Shortest routes between ",Town1," to ",Town2," is:\n"),
    route(Town1,Town2, [Town1] ,VisitedTowns, Dist),
    not(shorterRouteExist(Town1,Town2,Dist)),
    write_rote(Town1,VisitedTowns,Dist),nl,
    fail.
  showShortestRoutes(_,_).

  shorterRouteExist(Town1,Town2,Dist):-
    route(Town1,Town2, [Town1] ,_, Dist1),
    Dist1<Dist,!.

goal
  showAllRoutes("gordon", "tampa"),nl,
  showShortestRoutes("gordon", "tampa").
```

Chạy chương trình này cho kết quả là

```
All routes from gordon to tampa are:
  ["gordon","tampa"] --> 1000

  ["gordon","houston","kansas_city","tampa"] --> 370
```

```
  ["gordon","houston","tampa"] --> 650

Shortest routes between gordon to tampa is:
  ["gordon","houston","kansas_city","tampa"] --> 370
yes
```

Ví dụ 6 : Mô phỏng phần cứng. Cho mạch số Xor như hình vẽ



Chương trình Prolog sau là một ví dụ minh chứng kiểm tra cách vận hành của mạch.

```
domains
  d = integer

predicates
  not_(D,D)
  and_(D,D,D)
  or_(D,D,D)
  xor_(D,D,D)

clauses
  not_(1,0).
  not_(0,1).
  and_(0,0,0).
  and_(0,1,0).
  and_(1,0,0).
```

```
and_(1,1,1).
or_(0,0,0).
or_(0,1,1).
or_(1,0,1).
or_(1,1,1).

% See the documentarion for the XOR circuit
xor_(Input1,Input2,Output):-
        not_(Input1,N1),
        not_(Input2,N2),
        and_(Input1,N2,N3),
        and_(Input2,N1,N4),
        or_(N3,N4,Output).

goal
 xor_(Input1,Input2,Output),   % Use GOAL mode to see results !!!
 format(Msg," xor_(%,%,%)",Input1,Input2,Output),
 write(Msg).
```

Chạy chương trình này cho kết quả là

```
xor_(1,1,0)Input1=1, Input2=1, Output=0, Msg= xor_(1,1,0)
 xor_(1,0,1)Input1=1, Input2=0, Output=1, Msg= xor_(1,0,1)
 xor_(0,1,1)Input1=0, Input2=1, Output=1, Msg= xor_(0,1,1)
 xor_(0,0,0)Input1=0, Input2=0, Output=0, Msg= xor_(0,0,0)
4 Solutions
```

1) Xác định vị trí xảy ra sự cố trên hệ thống.
2) Xác định trạng thái vận hành và trạng thái không vận hành của các máy cắt trên hệ thống.
3) Chỉ rõ các thành phần của hệ thống.
4) Xác định hệ thống của các máy cắt dự phòng cho các máy cắt không vận hành trên hệ thống.

Cho hệ thống năng lượng gồm 8 máy cắt và 4 đường dẫn với các sự kiện hiện có trên hệ thống như đã được mô tả trên, công việc phân tích bảo vệ hệ thống này là giả sử rằng nếu có sự cố xảy ra trên mỗi đường dẫn LineX thì việc phân tích bảo vệ hệ thống phải xác định được vị trí đường dẫn LineX, các máy cắt bảo vệ đường dẫn LineX vận hành hoặc không vận hành và nếu máy cắt không vận hành thì phải có máy cắt khác dự phòng vận hành để bảo vệ hệ thống.

Kỹ thuật trí tuệ nhân tạo được ứng dụng để thiết kế hệ thống phân tích bảo vệ hệ thống năng lượng bao gồm các công việc như sau :

+ Công việc mô tả các sự kiện hiện có của hệ thống như nguồn cung cấp năng lượng, hai máy cắt bảo vệ đường một đường dẫn, máy cắt vận hành và máy cắt không vận hành, các máy cắt liên thông qua thanh góp.

+ Công việc thiết kế cơ sở luật suy diễn từ các sự kiện hiện có được mô tả trên hệ thống như luật suy diễn liên thông giữa hai máy cắt, luật suy diễn xác định máy cắt cùng bảo vệ đường dẫn, luật suy diễn máy cắt có nguồn, luật suy diễn máy cắt dự phòng cho một máy cắt khác, luật suy diễn máy cắt dự phòng cho một máy cắt khác không vận hành, luật suy diễn máy cắt mất nguồn và luật suy diễn xác định đường dẫn có sự cố.

Để mô tả các sự kiện hiện có trên hệ thống năng lượng điện, các vị từ tổng quát được thiết lập là

+ Vị từ generation(B) : mô tả máy cắt B nối trực tiếp với nguồn.

+ Vị từ protected_by(LineX, B1, B2) : Đường dẫn LineX được bảo vệ bởi hai máy cắt B1 và B2.

+ Vị từ connect(B1,B2) : mô tả máy cắt B1 là liên thông với máy cắt B2 qua thanh góp.

+ Vị từ operate(B) : mô tả máy cắt vận hành.

Để thiết kế hệ cơ sở luật suy diễn giải quyết bài tóan phân tích bảo vệ hệ thống năng lượng như được mô tả trên, các luật suy diễn được thiết lập là

+ Luật xử lý các máy cắt liên thông qua thanh góp.

If connect(B1, B2) then connection(B1, B2).

If connect(B2, B1) then connection(B1,B2).

+ Luật xử lý máy cắt cùng bảo vệ đường dẫn.

If protected_by(LineX, B1, B2) then other_breaker(B1, B2).

If protected_by(LineX, B2, B1) then other_breaker(B1, B2).

+ Luật xử lý máy cắt có nguồn.

If generation(B) then has_gen(B).

If connection(B, B1) and other_breaker(B1, B2) and has_gen(B2) then has_gen(B).

+ Luật xử lý máy cắt dự phòng cho một máy cắt khác.

If not(generation(B1)) and connection(B1, B3) and other_breaker(B3,B2) and has_gen(B2) then back_up(B1, B2).

+ Luật xử lý máy cắt dự phòng cho một máy cắt khác không vận hành.

If back_up(B1, B2) and not(operate(B2)) then backup_did_not_work(B1, B2).

+ Luật xử lý máy cắt mất nguồn.

If not(has_gen(B)) then no_source_coming(B).

If has_gen(B) and operate(B) then no_source_coming(B).

If back_up(B1, B2) and not(backup_did_not_work(B1, B2)) then No_source_coming(B1).

+ Luật xử lý xác định đường dẫn LineX có sự cố.

If no_source_coming(B1) and no_source_coming(B2) then fault(LineX, B1, B2).

Chương trình Prolog sau là một ví dụ minh chứng giải quyết bài tóan phân tích bảo vệ hệ thống nặng lượng với mô hình topo hình học đã cho trên.

```
database -tmp
protected_by(STRING,STRING,STRING)
connect(STRING,STRING)
operate(STRING)
generation(STRING)
predicates
```

```
nondeterm connection(STRING,STRING)
nondeterm other_breaker(STRING,STRING)
nondeterm has_gen(STRING)
nondeterm back_up(STRING,STRING)
nondeterm backup_did_not_work(STRING,STRING)
nondeterm no_source_coming(STRING)
        fault(STRING,STRING,STRING)
printbackup(STRING)
printout(STRING)
run
clauses
protected_by("line1","1","2").
protected_by("line2","3","4").
protected_by("line3","5","6").
protected_by("line4","7","8").
connect("2","3").
connect("2","6").
connect("2","7").
connect("3","6").
connect("3","7").
connect("6","7").
generation("1").
generation("4").
generation("5").
operate("1").
operate("4").
operate("5").
connection(B1,B2) :- connect(B1,B2).
connection(B1,B2) :- connect(B2,B1).
other_breaker(B1,B2) :- protected_by(_,B1,B2).
other_breaker(B1,B2) :- protected_by(_,B2,B1).
has_gen(B) :- generation(B),!.
has_gen(B) :- connection(B,B1),other_breaker(B1,B2),has_gen(B2),!.
back_up(B1,B2)                                                    :-
not(generation(B1)),connection(B1,B3),other_breaker(B3,B2),has_gen(B2).
backup_did_not_work(B1,B2) :- back_up(B1,B2),not(operate(B2)).
```

```
no_source_coming(B1) :- not(has_gen(B1)).
no_source_coming(B1) :- has_gen(B1),operate(B1).
no_source_coming(B1) :- back_up(B1,_),not(backup_did_not_work(B1,_)).
fault(_,B1,B2) :- no_source_coming(B1),no_source_coming(B2),!.
printbackup(B) :- back_up(B,B1),operate(B1),
                  write("Breaker"),
                  write(B1),
                  write(" Operated correctly as a backup breaker"),nl,fail.
printout(B) :- has_gen(B),operate(B),
                  write("Breaker"),write(B),
                  write(" operated correctly"),nl,!.
printout(B) :- has_gen(B),not(operate(B)),
                  write("Breaker"),
                  write(B),
                  write(" Malfunctioned"),nl,
                  not(printbackup(B)),!.
run :-
        protected_by(L,B1,B2),
        fault(L,B1,B2),
        write("Possible Fault Location is on "),
        write(L),nl,
        printout(B1),
        printout(B2),nl,nl,fail.
goal
  run.
```

Khi chạy chương trình này cho kết quả là

```
        Possible Fault Location is on line1
        Breaker1 operated correctly
        Breaker2 Malfunctioned
        Breaker4 Operated correctly as a backup breaker
        Breaker5 Operated correctly as a backup breaker

        Possible Fault Location is on line2
        Breaker3 Malfunctioned
        Breaker5 Operated correctly as a backup breaker
```

Bài tóan đặt ra là giả sử rằng robot đang ở tại vị trí ô có chỉ số (1,1) viếng thăm qua các ô khác để tìm ô chứa vàng, lấy vàng và mang vàng trở về lại nhà là ô (1,1). Quá trình thăm dò qua các ô, robot phải đối mặt với các ô chứa các chướng ngại vật như hầm bẩy và kẻ trông coi vàng. Robot sẽ bị nguy hiểm nếu nó đi vào các ô này. Trước khi robot đi vào các ô chứa các đối tượng này, nó có thể đánh mùi các đối tượng này ở các ô kề của chúng. Hãy xây dựng hệ cơ sở tri thức cho robot có thể thực hiện các thao tác thăm dò qua các ô biết suy nghĩ tránh được các ô chứa các chướng ngại vật và tìm đường an tòan đến ô chứa vàng, lấy vàng và mang vàng trở về lại nhà là ô (1, 1) ?

Các ký hiệu sử dụng với bài tóan này có nghĩa như sau :

+ Agent : robot.
+ Gold : vàng.
+ Wumous : kẻ trông coi vàng.
+ Pits : hầm bẩy.
+ Stench : mùi kẻ trông coi vàng.
+ Breeze : mùi hầm bẩy.
+ gliter : mùi có vàng.

Để xây dựng một hệ thống cơ sở tri thức cho robot có thể thực hiện được các yêu cầu đề ra như trên, các công việc sau cần phải được xem xét đó là

+ Mô tả các sự kiện về robot và các sự kiên liên quan với robot như thao tác di chuyển và thao tác lấy vàng của robot, vị trí và tình huống của robot, vị trí ô kề đối mặt với robot đến thăm dò hoặc không đến thăm dò, vị trí ô chứa chướng ngại vật và các ô kề chứa mùi chướng ngại vật.

+ Hệ thống cơ sở luật suy diễn cho robot biết suy nghĩ tính tóan để thực hiện các thao tác cần thiết của nó.

Để mô tả các sự kiện về robot và các sự kiện liên quan với robot, các vị từ và các hàm vị từ sau đây được thiết lập là

+ Thao tác di chuyển và thao tác lấy vàng của robot.

- turn(left) : lệnh quẹo trái.
- turn(right) : lệnh quẹo phải.
- forward : lệnh đi tới.
- grab : lệnh lấy vàng.

+ Vị trí, tình huống và định hướng nhìn của robot.
  - result(Action, S_i) = S_{i+1} : hàm trả về tình huống S_{i+1} khi thực hiện thao tác Action tại tình huống S_i .
  - at(Object, Location, Situation) : mô tả đối tượng tại vị Location với tình huống Situation.
  - orientation(Agent, Situation) = D : hàm trả về góc D định hướng nhìn của robot với tình huống situation. Theo qui ước, D quay tròn 360^0, D = 0, mặt của robot nhìn về hướng đông; D = 90, mặt của ro bot nhìn về hướng bắc; D = 180 , mặt của robot nhìn về hướng tây và D = 270, mặt của robot nhìn về hướng nam.
  - locationtoward([X, Y], D) = Location : hàm trả về vị trí chỉ số Location của ô kề đối mặt với ô (X, Y) được xác định bởi góc định hướng D.

Hệ thống cơ sở luật suy diễn cho robot có khả năng suy nghĩ tính tóan để thực hiện các thao tác cần thiết tránh chướng ngại vật và bám theo đường trên các ô an tòan tìm đến ô chứa vàng, lấy vàng và mang vàng về ô (1,1) được thiết lập gồm các luật suy diễn như sau :

+ Luật xử lý vị trí ô đối mặt với robot.

at(Agent, L, S)$\rightarrow$ locationAhead(A, S) =
locationtoward(L,orientation(Agent, S)).

+Luật xử lý vị trí các ô kề liên kết.

adjacent(L_1, L_2) $\leftrightarrow$ $\exists$D L_1 = locationtoward(L_2, D).

+ Luật xử lý xác định vị trí các ô chứa các đường biên.

wall(X, Y) $\leftrightarrow$ (X = 0 $\vee$ X = 5 $\vee$ Y = 0 $\vee$ Y = 5).

+ Luật xử lý thực hiện lệnh forward đi tới.

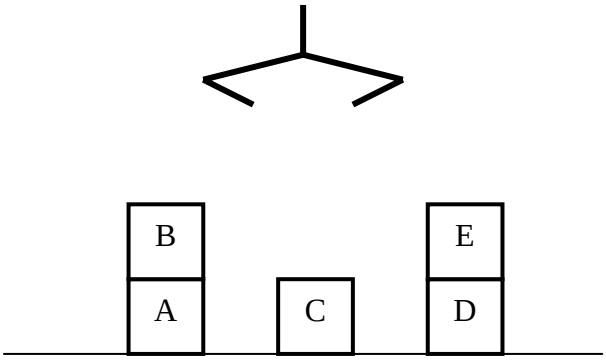
at(Agent, L, result(Action, S)) $\leftrightarrow$ Action = forward$\wedge$L =
locationAhead(Agent, S)$\wedge\neg$wall(L).

+ Luật xử lý thực hiện lệnh queo trái.

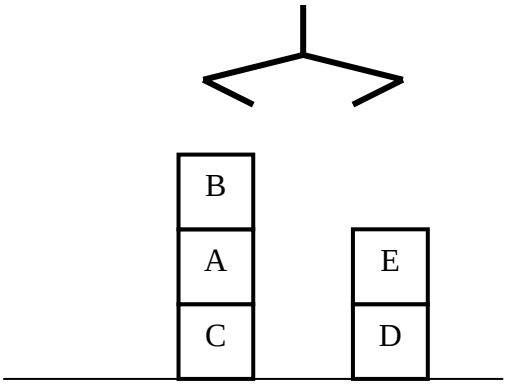
orientation(Agent, result(Action, S)) = D $\leftrightarrow$ Action = turn(left)$\wedge$
D = Mod(orientation(Agent, S) + 90, 360).

+ Luật xử lý thực hiện lệnh queo phải.

Cho bài tóan khối trên mặt bàn và cánh tay robot với trạng thái ban đầu và trạng thái đích như hình vẽ



Trạng thái ban đầu



Trạng thái đích

Bài tóan đặt ra là lập phương án cho cánh tay robot dời các khối từ trạng thái ban đầu của bài tóan sang trạng thái đích của bài tóan. Để làm được việc này, cánh tay robot phải thực hiện các thao tác là

+ goto(X, Y, Z) : di chuyển cánh tay robot đến vị trí có tọa độ X, Y, Z.
+ pickup(X) : thực hiện lệnh nhặt khối X lên từ mặt bàn.
+ putdown(X) : thực hiện lệnh đặt khối X xuống mặt bàn.
+ takeoff(X, Y) : thực hiện lệnh lấy khối X từ đỉnh của khối Y.
+ puton(X, Y) : thực hiện lệnh đặt khối X lên đỉnh của khối Y.

Để biểu diễn các trạng thái của bài tóan, các vị từ tổng quát được thiết lập là

+ location(W, X, Y, Z) : mô tả khối W định vị tại vị trí có tọa độ X, Y, Z.
+ on(X, Y) : mô tả khối X name trên khối Y.
+ clear(X) : mô tả làm sạch khối X (không có khối bất kỳ nằm trên khối X).
+ hold(X) : mô tả cánh tay robot cầm giữ khối X.
+ hold() : mô tả cánh tay robot rỗng.
+ ontable(X) : mô tả khối X name trên mặt bàn.

Hệ thống cơ sở luật suy diễn điều khiển cánh tay dời các khối từ trạng thái ban đầu đến trạng thái của bài tóan được thiết lập gồm các luật là

+ Luật xác định làm sạch khối.

$\forall X(clear(X) \leftarrow \neg\exists Y(on(Y, X)))$.

+ Luật xác định khối name trên mặt bàn.

$\forall X\ \forall Y\ (\neg on(Y, X) \leftrightarrow ontable(Y))$.

+ Luật xác định cánh tay robot rỗng.

$\forall Y$ (hold() $\leftrightarrow \neg$hold(Y)).

+ Luật thay đổi trạng thái khi thực hiện lệnh pickup.

$\forall X$ (pickup(X) $\rightarrow$ (hold(X) $\leftarrow$ (hold()$\wedge$ontable(X)$\wedge$clear(X)))).

+ Luật thay đổi trạng thái khi thực hiện lệnh putdown.

$\forall X$ (putdown(X) $\rightarrow$ (hold()$\wedge$ontable(X)$\wedge$clear(X)) $\leftarrow$ hold(X))).

+ Luật thay đổi trạng thái khi thực hiện lệnh puton.

$\forall X \forall Y$ (puton(X, Y) $\rightarrow$ ((hold()$\wedge$on(X, Y)$\wedge$clear(X))$\leftarrow$ (hold(X)$\wedge$clear(Y)))).

+ Luật thay đổi trạng thái khi thực hiện lệnh takeoff.

$\forall X \forall Y$ (takeoff(X, Y) $\rightarrow$ ((hold(X)$\wedge$clear(Y))$\leftarrow$ (hold()$\wedge$on(X,Y)$\wedge$clear(X)))).

+ Luật xử lý ràng buộc khi thực hiện lệnh takeoff.

$\forall X \forall Y \forall Z$ (takeoff(Y, Z) $\rightarrow$ (ontable(X)$\leftarrow$ontable(X))).

+ Luật xử lý ràng buộc khi thực hiện lệnh puton.

$\forall X \forall Y \forall Z$ (puton(Y, Z) $\rightarrow$ (ontable(X)$\leftarrow$ontable(X))).

Tương tự với hai luật ràng buộc trên, các luật ràng buộc khác cho quan hệ on và clear phải được thiết lập.

Do có nhiều luật ràng buộc khung kết hợp với các luật thay đổi trạng thái phát sinh ra một không gian trạng thái tìm kiếm của bài tóan là quá lớn và quá phức tạp tạo ra nhiều đường khác nhau dẫn về đích của bài tóan, trong đó mỗi đường là một phương án có thể điều khiển cánh tay robot dời các khối từ trạng thái ban đầu sang trạng thái đích.

Vì sử dụng quá nhiều luật ràng buộc khung với các lệnh thay đổi trạng thái của bài tóan tạo ra một không gian trạng thái tìm kiếm là quá phức tạp, điều này dẫn đến việc lập phương án cho cánh tay robot dời khối từ trạng thái ban đầu đạt đến trạng thái đích là rất khó khăn. Để khắc phục điều này, công việc loại bỏ việc sử dụng các luật ràng buộc khung, hệ thống luật thay đổi trạng thái của bài tóan có thể được cải tiến với luật ba thành phần là

$$pickup(X): \begin{cases} P: hold() \wedge ontable(X) \wedge clear(X) \\ A: hold(X) \\ D: hold() \wedge ontable(X) \wedge clear(X) \end{cases}$$

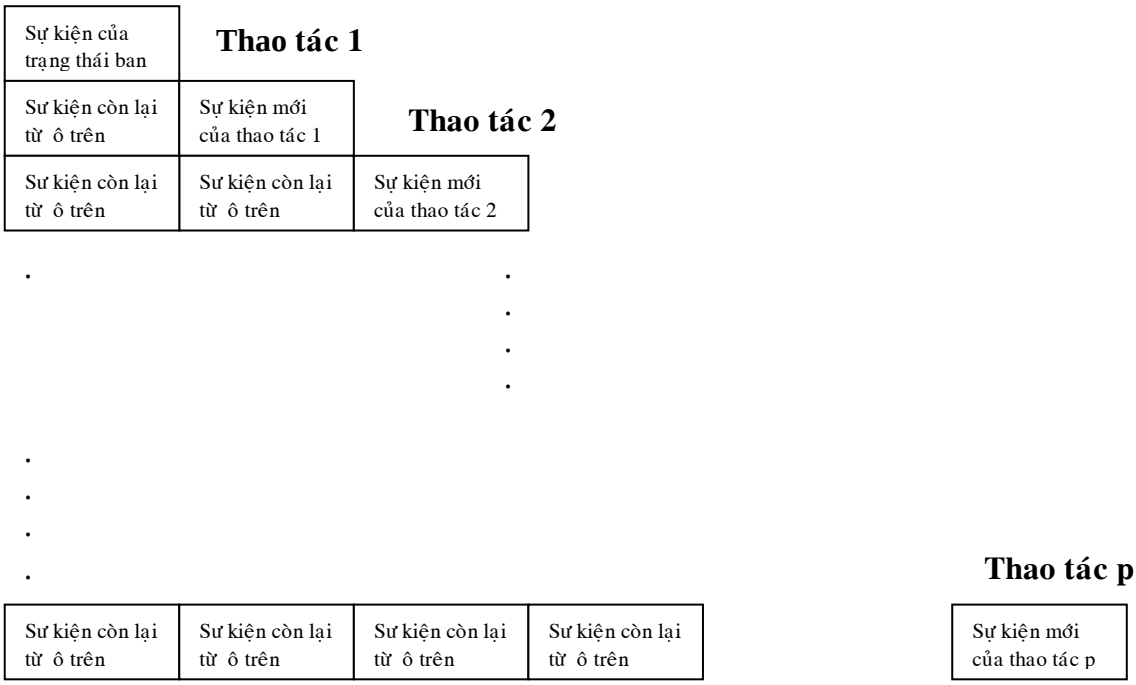
$$putdown(X):\begin{cases}P: hold(X)\\A: hold() \wedge ontable(X) \wedge clear(X)\\D: hold(X)\end{cases}$$

$$puton(X,Y):\begin{cases}P: hold(X) \wedge clear(Y)\\A: hold() \wedge on(X,Y) \wedge clear(X)\\D: hold(X) \wedge clear(Y)\end{cases}$$

$$takeoff(X,Y):\begin{cases}P: hold() \wedge on(X,Y) \wedge clear(X)\\A: hold(X) \wedge clear(Y)\\D: hold() \wedge on(X,Y) \wedge clear(X)\end{cases}$$

Trong đó, P là danh sách chứa các tiền điều kiện, A là danh sách chứa các sự kiện mới và D là danh sách chứa các tiền điều kiện đã sử dụng và được hủy bỏ.

<u>Bảng biểu tam giác</u> : Để nhớ lại các thao tác của một phương án, một cấu trúc dữ liệu mới được đề xuất đó là bảng biểu tam giác. Nếu phương án được thiết lập cho cánh tay robot dời khối từ trạng thái ban đầu có số p thao tác, thì bảng biểu tam giác được thiết lập với p + 1 hàng và p + 1 cột như bảng.



Cách thiết lập bảng biểu tam giác để nhớ lại các thao tác của phương án được thiết lập cho cánh robot dời các khối từ trạng thái ban đạt đến trạng thái đích của bài tóan khối được mô tả như sau :

+ Phương án có p thao tác, bảng biểu tam giác được thiết lập là p + 1 hàng và p + 1 cột.

+ Ô đầu tiên của bảng biểu với chỉ số (0, 0) chứa các sự kiện mô tả trạng thái ban đầu của bài tóan.

+ Ô có chỉ số (n, n) với n > 0 chứa các sự kiện mới của thao tác thứ n.

+ Ô có chỉ số (n, m) với m < n chứa các sự kiện còn lại từ ô (n-1, m) tức lã loại bỏ một số tiền điều kiện mà thao tác thứ n đã sử dụng ở ô(n-1, m) và số các sự kiện còn lại ở ô (n-1, m) được ghi xuống ô (n, m) với m < n.

Ví dụ : Cho luật suy diễn là

Nếu số người có bệnh tim thì trong số đó sẽ có một số người bị bệnh phổi.

Cho H là số người có bệnh tim và C là một số người trong số đó sẽ bị bệnh phổi, vậy thì luật suy diễn trên có thể được viết lại là

$$H \to C.$$

Qua thực nghiệm khảo sát cho thấy rằng :

+ Cứ 100 người, trong đó có 10 người bị bệnh tim. Vì thế xác suất của số người có bệnh tim là $P(H) = 0,1$.

+ Cứ 100 người, trong đó có 90 người không bị bệnh tim. Vì thế xác suất của những người không có bệnh tim là $P(\neg H) = 0,9$.

+ Cứ 100 người có bệnh tim thì trong số đó có 90 người bị bệnh phổi. Vì thế xác suất điều kiện số người bị bệnh phổi cho bởi số người có bệnh tim là $P(C\backslash H) = 0,9$.

+ Cứ 100 người không có bệnh tim thì trong số đó có 95 người không bị bệnh phổi. Do đó, xác suất điều kiện số người không bị bệnh phổi cho bởi số người không có bệnh tim là $P(\neg C\backslash\neg H) = 0,95$.

+ Cứ 100 người không có bệnh tim thì trong số có 5 người bị bệnh phổi. Do đó, xác suất điều kiện số người bị bệnh phổi cho bởi số người không có bệnh tim là $P(C\backslash\neg H) = 0,05$.

Ta có xác suất của luật suy diễn $H \to C$ đó chính là xác suất điều kiện C cho bởi bằng chứng H đó là $P(C\backslash H) = 0,9$.

Công thức tính xác suất của kết luận C với dạng luật suy diễn $H \to C$ là

$$P(C) = P(H)\times P(C\backslash H) + P(\neg H)\times P(C\backslash\neg H).$$

Vậy thì ta có xác suất của kết luận C là

$$P(C) = 0,1\times 0,9 + 0,9\times 0.05 = 0,135 \text{ hay } 13,5\%.$$

Với lý giải chính xác dưới điều kiện không chắc chắn dùng xác suất cho các luật suy diễn dạng phức tạp, công việc tính xác suất của vế kết luận sẽ xuất hiện nhiều ẩn số xác suất chưa biết trong công thức tính xác suất. Để khắc phục điều này, công

CF(b,a) = MB(b,a) – MD(b,a)

+ Nếu số đo chắc chắn của kết luận b với bằng chứng a là CF(b,a) = -1 thì kết luận rằng b là sai.

+ Nếu số đo chắc chắn của kết luận b với bằng chứng a là CF(b,a) = 0 thì kết luận rằng b là chưa biết.

+ Nếu số đo chắc chắn của kết luận b với bằng chứng a là CF(b,a) = 1 thì kết luận rằng b là đúng.

Khảo sát các phương trình trên với các trường hợp là

+ **Trường hợp 1** : Bằng chứng a dẫn đến kết luận b là đúng hay nói cách khác, xác suất điều kiện b cho bởi a là đúng.

Với trường hợp này, ta có P(b\a) = 1 và P(b) = 1; do đó ta có MB(b,a) = 1 và MD(b,a) = 0. Vậy thì CF(b,a) = 1; do đó ta kế luận rằng b là đúng.

+ **Trường hợp 2** : Bằng chứng a dẫn đến kết luận b là sai hay nói cách khác, xác suất điều kiện không có mặt b cho bởi a là đúng.

Với trường hợp này, ta có P(¬b\a) = 1 và P(b) = 0; do đó ta có MB(b,a) = 0 và MD(b,a) = 1. Vậy thì CF(b,a) = -1; do đó ta có thể kết luận rằng b là sai.

+ **Trường hợp 3** : Không có mặt bằng chứng a dẫn đến kết luận b.

Với trường hợp này, ta có P(b\a) = P(b); do đó MB(b,a) = 0 và MD(b,a) = 0.

Vậy thì CF(b,a) = 0 và do đó ta kết luận rằng b là chưa biết.

+ **Trường hợp 4** : Bằng chứng khả thi a dẫn đến kết luận b.

Với trường hợp này, ta tcó xác suất điều kiện b cho bởi a bị chặn bởi là

P(b) < P(b\a) < 1.

Vì thế MB và MD được xác định là

$$MB(b,a) = \frac{P(b \setminus a) - P(b)}{1 - P(b)}$$

và MD(b,a) = o.

Do đó, CF(b,a) = MB(b,a) là một số dương. Điều này chứng tỏ rằng kết luận b là khả thi.

+ **Trường hợp 5** : Bằng chứng không khả thi dẫn đến kết luận b.

Với trường hợp này, xác suất điều kiện b cho bởi a bị chặn bởi là

$$A = \sum_{i=1}^{n} \mu_A(x_i) / x_i$$

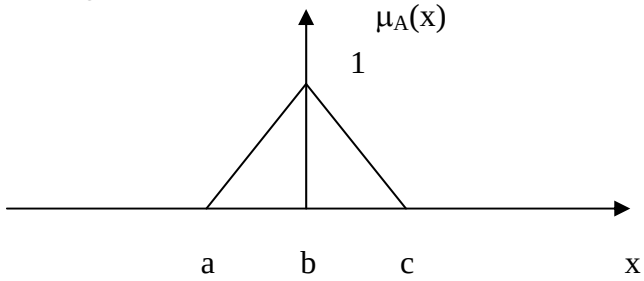
Trong đó, ký hiệu $\sum$ là tóan tử hợp và ký hiệu / là tóan tử kết hợp giữa giá trị rõ và giá trị mờ tương ứng.

+ **Hàm liên thuộc** : Có hai cách xây dựng hàm liên thuộc cho tập mờ A đó là xây dựng hàm liên thuộc dưới dạng bảng và xây dựng hàm liên thuộc dưới dạng hàm.

- Hàm liên thuộc dưới dạng bảng gồm hai cột và nhiều hàng, cột thứ nhất chứa giá trị rõ và cột chứa các giá trị mờ tương ứng được mô tả tổng quát như bảng

Đại lượng rõ x_i	Đại lượng mờ $\mu_A(x_i)$
x_1	$\mu_A(x_1)$
x_2	$\mu_A(x_2)$
x_n	$\mu_A(x_n)$

- Hàm liên thuộc dưới dạng hàm có nhiều hàm khác nhau nhưng hàm liên thuộc dạng tam giác là được sử dụng phổ biến nhất. Cho đồ thị biểu diễn tập mờ A dạng tam giác như hình



Hàm liên thuộc dạng tam giác được thiết lập là

$$\mu_A(x) = \begin{cases} \dfrac{x-a}{b-a} & if\ a \le x \le b. \\ \dfrac{c-x}{c-b} & if\ b \le x \le c. \end{cases}$$

trong đó, a là cận trái, b là tâm và c là cận phải của tam giác trên trục hòanh x.

$$\mu_R(x,y) = \begin{cases} 1 & if\ (x,y) \in R \\ 0 & if\ (x,y) \notin R \end{cases}$$

+ **Quan hệ mờ** : Cho R là tập con của tập tích X×Y, R được gọi là quan hệ mờ trong X×Y, nếu R được định nghĩa bằng hàm liên thuộc của nó sao cho bị chặn giữa 0 và 1 đó là

$$0 \le \mu_R(x,y) \le 1.$$

+ **Biểu diễn quan hệ mờ** : Quan hệ mờ có thể được biểu diễn dưới dạng ma trận là

$$R(x,y) = \begin{bmatrix} \mu_R(x_1,y_1) & \mu_R(x_1,y_2). & . & . & \mu_R(x_1,y_n) \\ & & & & \\ & & & & \\ \mu_R(x_m,y_1) & \mu_R(x_m,y_2). & . & . & \mu_R(x_m,y_n) \end{bmatrix}$$

+ **Các phép tóan trên các quan hệ mờ** : Cho P là quan hệ mờ trong tập tích X×Y và Q là quan hệ mờ trong tập tích Y×Z. Quan hệ mờ trong tập tích X×Z được xác định bằng phương trình là

$$R = PoQ$$

Trong đó ký hiệu o là tóan tử hợp thành mờ.

Có nhiều loại tóan tử hợp thành mờ, tuy nhiên hai loại tóan tử hợp thành mờ thông dụng nhất đó là tóan tử max-min và tóan tử max-product.

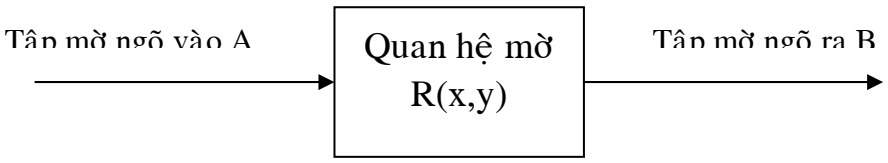
- Tóan tử max-min được thiết lập là

$$\mu_R(x,z) = \mu_{P\circ Q}(x,z) = \max\min\left\{\mu_P(x,y), \mu_Q(y,z)\right\}$$

- Tóan tử max-product được thiết lập là

$$\mu_R(x,z) = \mu_{P\circ Q}(x,z) = \max\left\{\mu_P(x,y) \times \mu_Q(y,z)\right\}$$

+ Phương trình quan hệ mờ : Cho A là tập mờ ngõ vào trên biến ngôn ngữ vào X, R là quan hệ mờ trong tập tích X×Y và B là tập mờ ngõ ra trên biến ngôn ngữ ngõ ra Y. Quan hệ vào ra của hệ thống mờ này được mô tả bằng lưu đồ khối như hình



Do đó, giá trị chân lý của phép tóan giao P và Q được thiết lập là

$$T(P \wedge Q) = \min\{T(P), T(Q)\}.$$

- **Phép tóan logic kéo theo** :

Cho đề xuất P với $x \in A$ và đề xuất Q với $x \in B$, trong đó A và B là hai tập mờ trong tập cơ sở X với các hàm liên thuộc là $\mu_A(x)$ và $\mu_B(x)$. Khi đó phép tóan logic kéo theo P cho Q là

$$P \rightarrow Q : \quad x \in A \rightarrow x \in B.$$

Do đó, giá trị chân lý của phép tóan kéo theo P cho Q được thiết lập là

$$T(P \rightarrow Q) = T(\neg P \vee Q) = \max\{T(\neg P), T(Q)\}.$$

Xét luật suy diễn mờ với dạng là

$$P \rightarrow Q \text{ if x is A then y is B,}$$

trong đó, A là tập mờ ngõ vào trong tập cơ sở ngõ vào X với hàm liên thuộc là $\mu_A(x)$ và B là tập mờ ngõ ra trong tập cơ sở ngõ ra Y với hàm liên thuộc là $\mu_A(x)$.

Mô hình luật suy diễn mờ này là tương đương với quan hệ mờ là

$$R = (A \times B) \vee (\neg A \times Y).$$

Do đó hàm liên thuộc của nó được thiết lập là

$$\mu_R(x,y) = \max[\mu_A(x) \wedge \mu_B(y), (1 - \mu_A)].$$

<u>Ví dụ</u> : Cho X là tập cơ sở ngõ vào biểu diễn tốc độ động cơ và A là tập mờ ngõ vào biểu diễn tốc độ động cơ an tòan trong X được thu thập từ thực nghiệm là

$$A = \{0.3/20 + 0.6/30 + 0.8/40 + 1/50 + 0.7/60 + 0.4/70\}.$$

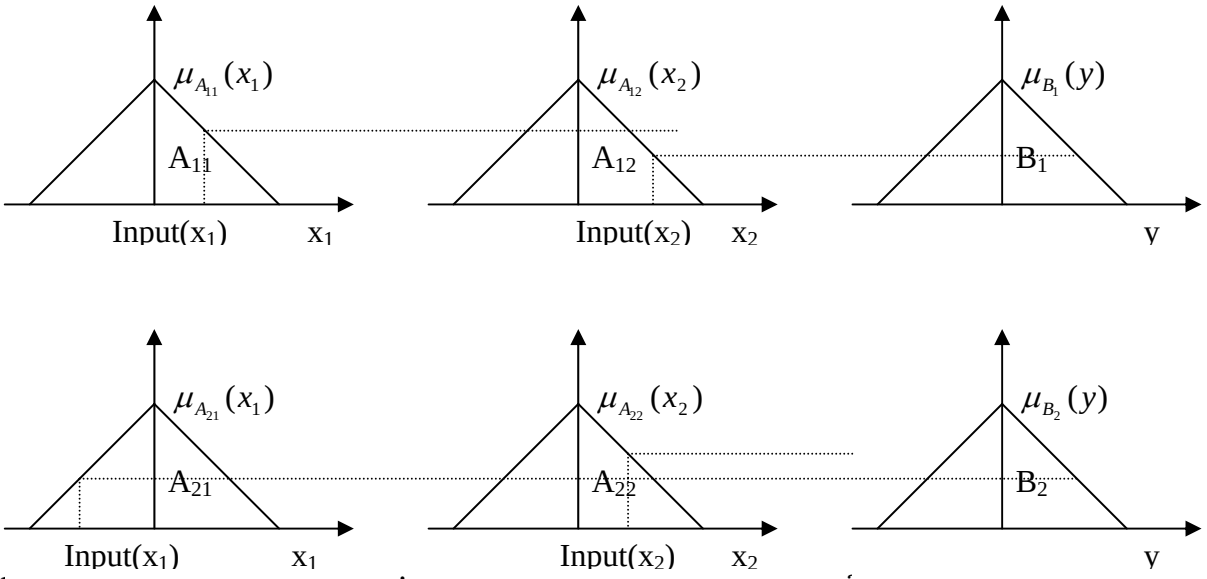
Cho Y là tập cơ sở ngõ ra biểu diễn điện áp động cơ và B là tập mơ ngõ rà biểu diễn điện áp động cơ bình thường được thu thập từ thực nghiệm là

$$B = \{0.1/1 + 0.3/2 + 0.8/3 + 1/4 + 0.7/5 + 0.4/6 + 0.2/7\}.$$

Quan hệ mờ giữa tốc độ động cơ an tòan và điện áp động cơ bình thường được thiết lập là

$$R = x \in A \rightarrow y \in B = (A \times B) \vee (\neg A \times Y).$$

Từ đây, ta có quan hệ mờ R là



Kỹ thuật suy diên mờ max-min xác định tập mờ ngõ ra của hệ thông được mô tả như sau :

- Tập mờ ngõ ra B_1' của luật thứ nhất được xác định với hàm liên thuộc của nó là

$$\mu_{B_1'}(y) = \min\{\alpha_1, \mu_{B_1}(y)\}$$

trong đó, α_1 là số đo mờ ở vế điều kiện của luật 1 được xác định là

$$\alpha_1 = \min\{\mu_{A_{11}}(x_1), \mu_{A_{12}}(x_2)\}$$

- Tập mờ ngõ ra B_2' của luật thứ 2 được xác định với hàm liên thuộc của nó là

$$\mu_{B_2'}(y) = \min\{\alpha_2, \mu_{B_2}(y)\}$$

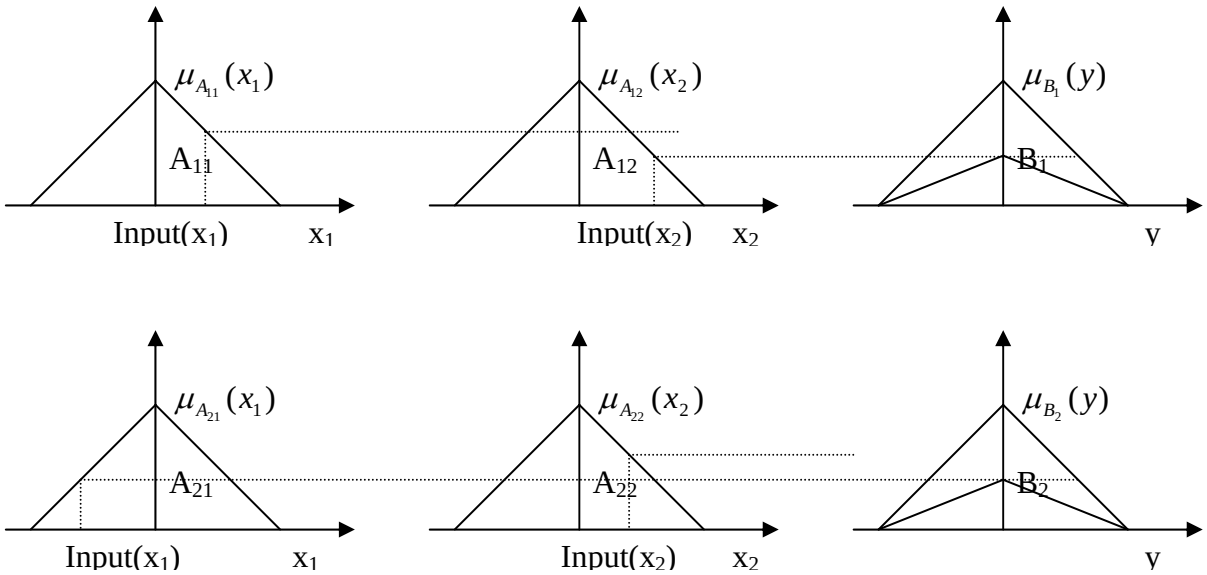
trong đó, α_2 là số đo mờ ở vế điều kiện của luật 2 được xác định là

$$\alpha_2 = \min\{\mu_{A_{21}}(x_1), \mu_{A_{22}}(x_2)\}$$

- Tập mờ ngõ ra B' của hệ thống đó chính là tập hợp B' của hai tập mờ ngõ ra B_1' và B_2' trước đó của hai luật là B' = $B_1' \vee B_2'$ và nó được định bằng hàm liên thuộc của nó là

$$\mu_{B'}(y) = \max\{\mu_{B_1'}(y), \mu_{B_2'}(y)\}$$

+ **<u>Kỹ thuật suy diễn mờ max-product</u>** : Cũng giống như kỹ thuật suy diễn mờmax-min, kỹ thuật suy diễn mờ max-product được mô tả bằng đồ thị như hình



Kỹ thuật suy diễn mờ max-product xác định tập mờ ngõ ra của hệ thống được mô tả như sau :

- Tập mờ ngõ ra B_1' của luật thứ nhất được xác định với hàm liên thuộc của nó là

$$\mu_{B_1'}(y) = \alpha_1 \times \mu_{B_1}(y)$$

trong đó, α_1 là số đo mờ ở vế điều kiện của luật 1 được xác định là

$$\alpha_1 = \min\left\{\mu_{A_{11}}(x_1), \mu_{A_{12}}(x_2)\right\}$$

- Tập mờ ngõ ra B_2' của luật thứ 2 được xác định với hàm liên thuộc của nó là

$$\mu_{B_2'}(y) = \alpha_2 \times \mu_{B_2}(y)$$

trong đó, α_2 là số đo mờ ở vế điều kiện của luật 2 được xác định là

$$\alpha_2 = \min\left\{\mu_{A_{21}}(x_1), \mu_{A_{22}}(x_2)\right\}$$

- Tập mờ ngõ ra B' của hệ thống đó chính là tập hợp B' của hai tập mờ ngõ ra B_1' và B_2' trước đó của hai luật là B' = $B_1' \vee B_2'$ và nó được định bằng hàm liên thuộc của nó là

$$\mu_{B'}(y) = \max\left\{\mu_{B_1'}(y), \mu_{B_2'}(y)\right\}$$

Giải thuật học hướng tổng quát hóa đến đặc trưng hóa được mô tả là

Begin

- Cho danh sách G chứa mẫu với các thành phần tổng quát nhất đó là các biến số mô tả các thành phần của đối tượng.
- Cho P là danh sách chứa các mẫu huấn luyện dương.
- Cho mỗi mẫu huấn luyện âm n

  Begin

  - Cho mỗi mẫu g∈G hợp với n thì thay thế các thành phần tổng quát của g với các thành phần đặc trưng sao cho không hợp với n.
  - Loại bỏ tất cả các mẫu đặc trưng hơn một vài mẫu khác trong G.
  - Loại bỏ tất cả các mẫu không hợp với vài mẫu dương p trong P.

  End;
- Cho mỗi mẫu dương p

  Begin

  - Loại bỏ tất cả các mẫu không hợp với p trong G.
  - Cộng p vào tập P để giám sát các mẫu quá đặc trưng trong quá trình học.

  End;

End.

Ví dụ : Học nhận dạng các đối tượng của lớp quả bóng sử dụng giải thuật học hướng đặc trưng và hướng tổng quát.

Cho miền của các đối tượng với các giá trị là

Kích_thước = {lớn, nhỏ}.

Màu = {đỏ, trắng, xanh}.

Hình = {quả_bóng, viên_gạch, hộp_phấn}.

Dữ liệu học cho các đối tượng này được thiết lập là

+ Tập các mẫu dữ liệu huấn luyện dương P gồm các mẫu là

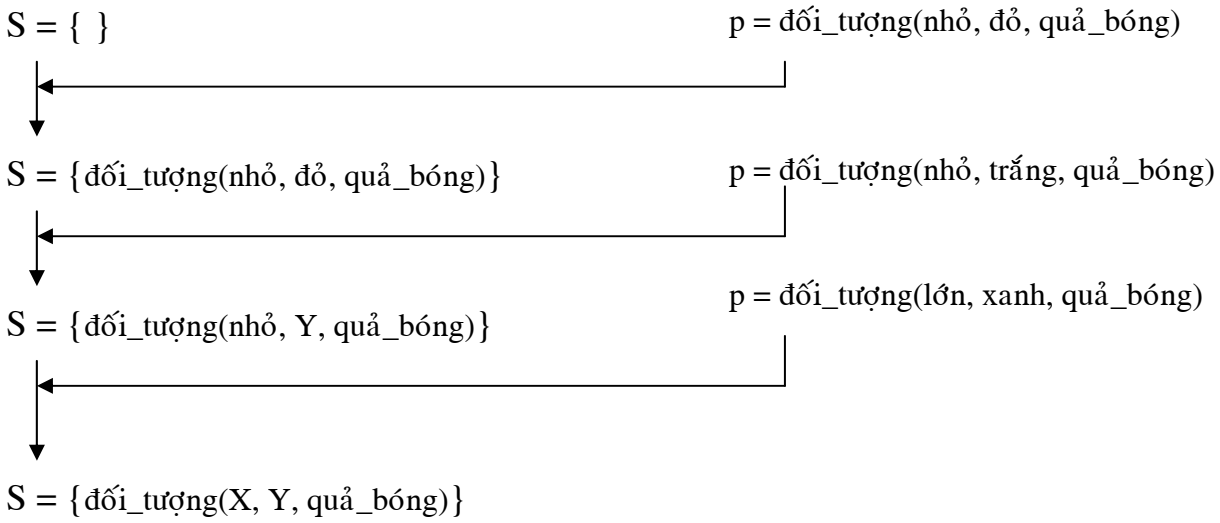
P = {đối_tượng(nhỏ, đỏ, quả_bóng), đối_tượng(lớn, đỏ, quả_bóng), đối_tượng(nhỏ, trắng, quả_bóng), đối_tượng(lớn, trắng, quả_bóng), đối_tượng(nhỏ, xanh, quả_bóng), đối_tượng(lớn, xanh, quả_bóng)}.

+ Tập các mẫu dữ liệu huấn luyện âm N gồm các mẫu là

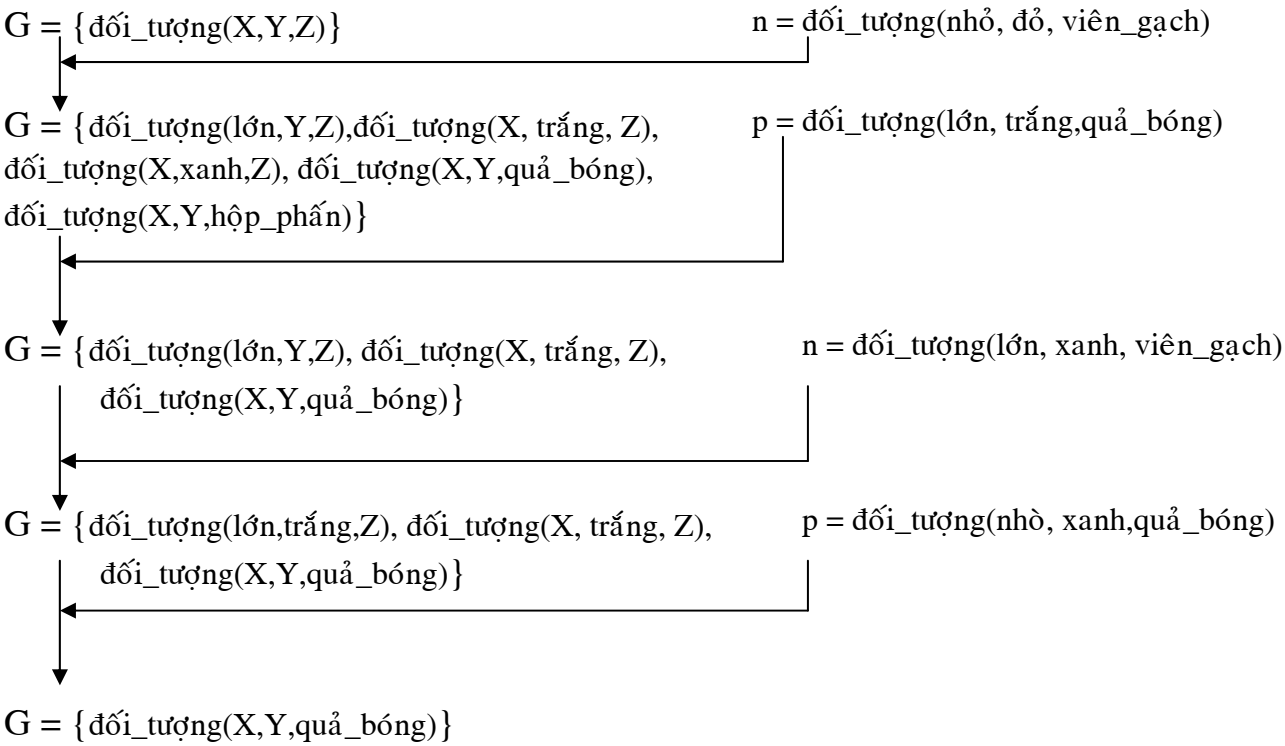
N = { đối_tượng(nhỏ, đỏ, viên_gạch), đối_tượng(lớn, đỏ, viên_gạch), đối_tượng(nhỏ, trắng, viên_gạch), đối_tượng(lớn, trắng, viên_gạch), đối_tượng(nhỏ, xanh, viên_gạch), đối_tượng(lớn, xanh, viên_gạch), đối_tượng(nhỏ, đỏ, hộp_phấn), đối_tượng(lớn, đỏ, hộp_phấn),

đối_tượng(nhỏ, trắng, hộp_phấn), đối_tượng(lớn, trắng, hộp_phấn),
đối_tượng(nhỏ, xanh, hộp_phấn), đối_tượng(lớn, xanh, hộp_phấn)}.

+ Quá trình học để nhận dạng các đối tượng của lớp quả bóng dùng giải thuật học hướng đặc trưng được mô tả như hình



Quá trình học để nhận dạng các đối tượng của lớp quả bóng dùng giải thuật học hướng tổng quát được mô tả như hình



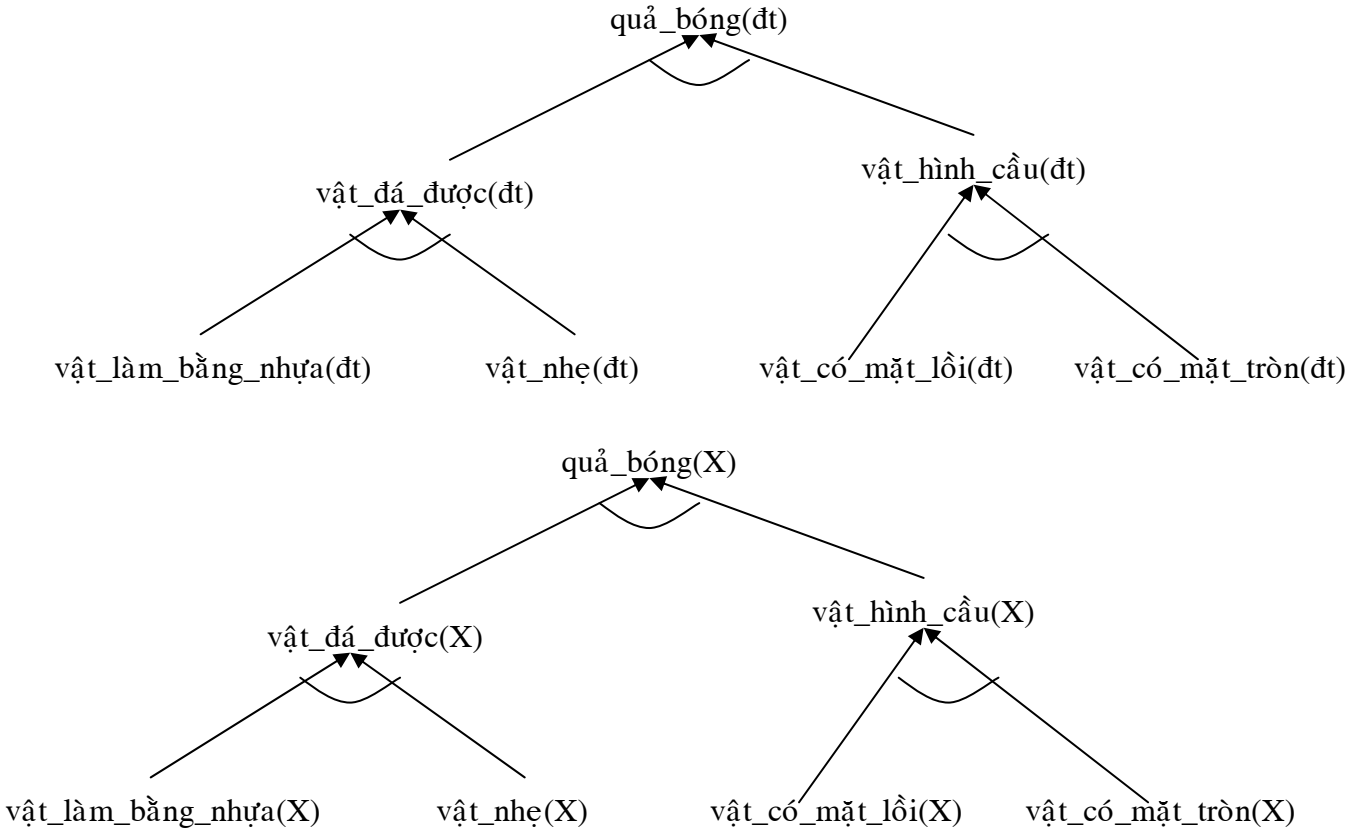
```
Function    induce_tree(Example_set, Properties)
    Begin
        If ( Tất cả các thành viên trong Example_set là cùng lớp )
            Then ( tạo ra nút lá đánh nhãn với lớp đó)
        Elseif ( Properties là danh sách rỗng) Then ( Trả về nút lá có
        đánh nhãn giới từ hoặc của tất cả các lớp trong Example_set )
        Else begin
        -Chọn một thuộc tính P bất kỳ trong danh sách Properties làm gốc
         của cây và loại bỏ thuộc tính này khỏi danh sách.
        - Cho mỗi giá trị V của thuộc tính P
         Begin
           -  Tạo ra một nhánh của cây có đánh nhãn V.
           -  Đặt Partition chứa tất cả các mẫu có giá trị V.
           -  Thủ tục đệ quy cho mỗi cây con bằng cách gọi hàm
           induce_tree(Partition, Properties), nối kết quả vào nhánh V.
           End;
        End;
    End.
```

Ví dụ : Cho dữ liệu thu thập được về việc cho con nợ vay vốn như bảng

stt	Rủi Ro	Uy Tín	Khỏan Nợ	Thế Chấp	Thu Nhập
1	cao	xấu	nhiều	không	thấp
2	cao	chưa biết	nhiều	không	trung bình
3	vừa	chưa biết	ít	không	trung bình
4	cao	chưa biết	ít	không	thấp
5	thấp	chưa biết	ít	không	cao
6	thấp	chưa biết	ít	có	cao
7	cao	xấu	ít	không	thấp
8	vừa	xấu	ít	có	cao
9	thấp	tốt	ít	không	cao
10	thấp	tốt	nhiều	có	cao
11	cao	tốt	nhiều	không	thấp
12	vừa	tốt	nhiều	không	trung bình
13	thấp	tốt	nhiều	không	cao
14	cao	xấu	nhiều	không	trung bình

các thuộc tính đặc trưng của đối tượng. Giai đọan tổng quát hóa đó là tổng quát hóa các thuộc tính đặc trưng của đối tượng đã được giải thích với biến số X để tìm ra các nhân tố quyết định tổng quát nhất cho tiên_đề(X) dẫn đến kết luận rằng X là lớp của các đối tượng quả bóng.

Quá trình học với thể loại này được mô tả bằng cây như hình

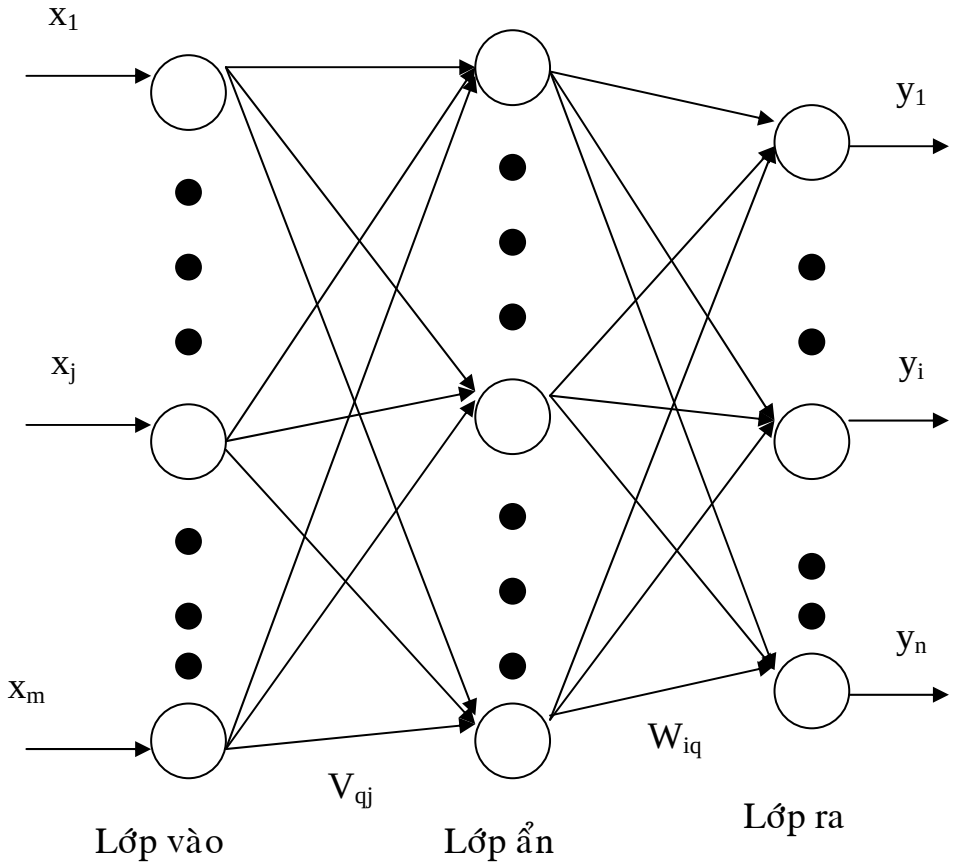


Luật suy diễn tổng quát cho tiên_đề(X) là một biểu thức của các phép tóan giao liên từ và với các thành phần của nó là các nút lá của cây tổng quát. Luật được thiết lập là

$$\text{vật\_làm\_bằng\_nhựa}(X) \wedge \text{vật\_nhẹ}(X) \wedge \text{vật\_có\_mặt\_lồi}(X) \wedge \text{vật\_có\_mặt\_tròn}(X) \rightarrow \text{quả\_bóng}(X).$$

+ **<u>Học không giám sát</u>** : Học không giám sát còn được gọi là thể loại tự học. Hệ thống không được cung cấp bất kỳ một thông tin tín hiệu học nào của thầy giáo. Hệ

hai neuron được kết nối dưới dạng ức chế. Nếu tín hiệu truyền giữa hai neuron là điện áp zero thì hai neuron là không có sự kết nối. Lượng điện áp truyền giữa các neuron được gọi là cường độ kết nối. Trên cơ sở của hệ neuron sinh học con người như được mô tả, một mạng neuron nhân tạo nhiều lớp được thiết lập như hình



Có ba thành phần cơ bản của các mạng neuron nhân tạo đó là mô hình kết nối, đơn vị xử lý và luật học.

+ **<u>Mô hình kết nối</u>** : Có hai mô hình kết nối đó là kết nối truyền thẳng và kết nối hồi quy. Mô hình kết nối truyền thẳng được gọi là mạng truyền thẳng đó là cấu trúc mạng được kết nối chuyển tiếp tín hiệu từ lớp vào thông qua lớp ẩn và đến lớp ra. Mô hình kết nối hồi quy được gọi là mạng hồi quy đó là cấu trúc mạng được kết nối chuyển tiếp tín hiệu từ lớp vào thông qua lớp ẩn đến lớp ra và đồng thời hồi tiếp tín hiệu về đơn vị xử lý chính nó hoặc các đơn vị xử khác trong lớp hoặc ở lớp khác.

+ **Đơn vị xử lý** : Một mạng neuron nhân tạo có nhiều lớp đó là lớp vào, các lớp ẩn và lớp ra. Lớp vào chứa các neuron được xem như nơi chứa các tín hiệu vào. Các lớp ẩn chứa các neuron được xem như các đơn vị xử lý. Lớp ra chứa các neuron được xem như các đơn vị xử lý ra quyết định. Kết hợp với mỗi đơn vị xử lý có hàm tổng hợp và hàm kích hoạt. Hàm tổng hợp có chức năng tổng hợp tất cả các thông tin từ các ngõ vào của đơn vị và hàm kích hoạt có chức năng tạo tín hiệu ra của đơn vị khi nhận được tín hiệu vào từ hàm tổng hợp.

- Hàm tổng hợp dạng tuyến tính của mỗi đơn vị xử lý thứ i được thiết lập là

$$f_i = \sum_{j=1}^{m} W_{ij} x_j - \theta_i$$

trong đó, W_{ij} là trọng số kết nối giữa đơn vị j và đơn vị i, x_j là ngõ ra của đơn vị j đó chính là ngõ vào của đơn vị i và θ_i là đơn vị ngưỡng của đơn vị xử lý i.

- Hàm kích hoạt tạo tín hiệu ra của đơn vị xử lý thứ i được thiết lập một trong các dạng là

* Hàm bậc thang đơn vị : $$a(f_i) = \begin{cases} 1 & if \quad f_i \geq 0 \\ 0 & if \quad f_i \leq 0 \end{cases}$$

* Hàm unipolar sigmoid : $$a(f_i) = \frac{1}{1 + e^{-\lambda f_i}}$$

* Hàm hyperbolic tangent : $$a(f_i) = \frac{e^{f_i} - e^{-f_i}}{e^{f_i} + e^{-f_i}}$$

+ **Luật học** : Có hai cách học trong các mạng neuron nhân tạo đó là học cấu trúc và học thông số. Học cấu trúc là quá trình học thay đổi cấu trúc bên trong của mạng. Học thông số là quá trình học cập nhật các trọng số kết nối giữa các đơn vị xử lý trong mạng sao cho xấp xỉ với bộ trọng số mong muốn để có được ánh xạ vào ra như mong muốn.

Cho W_{ij} là trọng số kết nối giữa đơn vị thứ j và đơn vị thứ i, luật học cập nhật trọng trọng số tại thời điểm t+1 được thiết lập là

$$W_{ij}(t+1) = W_{ij}(t) + \eta \Delta W_{ij}(t)$$

Trong đó, η là hằng số dương đó được gọi là tốc độ học và $\Delta W_{ij}(t)$ là lượng gia tăng trọng số tại thời điểm t.

Có ba thể loại học trong các mạng neuron nhân tạo đó là học giám sát, học củng cố và học không giám sát.

$$y_i(k) = a(net_i(k)).$$

Bước 2 : Tính sai số chuan 2 giữa ngõ ra mong muốn và ngõ ra thực sự của mạng.

$$E(k) = \frac{1}{2}\sum_{i=1}^{n}(d_i(k) - y_i(k))$$

Bước 3 : Lan truyền ngược cập nhật trọng số kết nối giữa các lớp.

Cho i = 1 đến n

$$\delta_i(k) = (d_i(k) - y_i(k)) \times a'(net_i(k))$$

Cho i = 1 đến n

Cho q = 1 đến l

$$W_{iq}(k+1) = W_{iq}(k) + \eta \times \delta_i(k) \times z_q(k).$$

Cho q = 1 đến l

$$\delta_q(k) = a'(net_q(k)) \times \sum_{i=1}^{n} W_{qi}(k)\delta_i(k).$$

Cho q = 1 đến l

Cho j = 1 đến m

$$V_{qj}(k+1) = V_{qj}(k) + \eta \times \delta_q(k) \times x_j(k).$$

Bước 4 : Kiểm tra nếu k < p thì tăng k = k + 1 và quay về bước 1; mặt khác đến bước 5.

Bước 5 : Kiểm tra nếu E < E_{max} thì dừng thủ tục huấn luyện; mặt khác thiết lập E = 0 và k = 1 và quay về bước 1 thực hiện một thế hệ huấn luyện khác.

……..o0o……..

GIÁO TRÌNH TRÍ TUỆ NHÂN TẠO

NÓI ĐẦU

Trí tuệ nhân tạo là môn cơ sở chuyên ngành trong chương trình đào tạo kỹ sư và cử nhân Tin học. Mục đích của môn học này là trang bị cho sinh viên những kiến thức cơ bản nhất về các phương pháp giải quyết vấn đề và các kỹ thuật xử lý tri thức. Trên cơ sở các kiến thức tiếp thu được học sinh có thể đi sâu tìm hiểu về các chuyên đề khác như hệ chuyên gia, hệ hỗ trợ ra quyết định, phần mềm dạy học thông minh...

Nội dung giáo trình được trình bày thành ba chương:

Chương 1: Nội dung chương này trình bày các khái niệm cơ bản về khoa học Trí tuệ nhân tạo.

Chương 2: Đây là chương trọng tâm của giáo trình. Ở đây chúng tôi giới thiệu các phương pháp biểu diễn vấn đề, các kỹ thuật tìm kiếm lời giải.

Chương 3. Trình bày các kỹ thuật biểu diễn tri thức và xử lý tri thức. Vì thời lượng của môn học quá ít nên trong chương này chúng tôi chỉ trình bày những nét hết sức cơ bản và không đi sâu vào các vấn đề cụ thể.

Giáo trình này được biên soạn để làm tài liệu giảng dạy cho giáo viên và học tập cho sinh viên đồng thời là tài liệu tham khảo cho những ai quan tâm đến Trí tuệ nhân tạo.

Trong quá trình biên soạn tác giả đã nhận được ý kiến đóng góp và động viên của bạn bè, đồng nghiệp và đặc biệt là các cán bộ có kinh nghiệm trong khoa CNTT Đại học Vinh và ĐHBK Hà nội về lĩnh vực này. Nhân đây tác giả muốn bày tỏ lòng biết ơn tới TS Nguyễn Thanh Thuỷ (ĐHBK Hà nội) người đã giúp đỡ tác giả trong suốt quá trình biên soạn giáo trình này.

Để hoàn thành tập giáo trình này, mặc dù đã cố gắng rất nhiều nhưng tác giả tin rằng vẫn không tránh khỏi các sai sót và hạn chế. Hy vọng rằng tác giả sẽ tiếp tục nhận được các ý kiến góp ý quý báu của bạn đọc và đồng nghiệp.

Vinh, tháng 7 năm 2000

Tác giả

Chương I
TỔNG QUAN VỀ TÌNH HÌNH NGHIÊN CỨU VÀ ỨNG DỤNG CỦA TRÍ TUỆ NHÂN TẠO

1.1. LỊCH SỬ PHÁT TRIỂN CỦA KHOA HỌC TTNT

Những năm gần đây, ta thường gặp các thuật ngữ: Máy tính thông minh, máy tính thế hệ thứ 5, trí tuệ nhân tạo, hệ chuyên gia... Sự xuất hiện của các ngôn ngữ lập trình: LISP (List Processing), PROLOG đã mở đường cho việc xây dựng và áp dụng vào thực tế hàng loạt các chương trình có khả năng xử lý "thông minh".

Trước đây, thuật ngữ trí tuệ nhân tạo (TTNT) thường được hiểu như là các máy có khả năng "suy nghĩ", có khả năng cạnh tranh với bộ óc con người.

Ý tưởng về mô hình các máy thông minh đã được đưa ra hàng trăm năm trước đây, song mãi đến năm 1930, vấn đề này mới được nghiên cứu một cách nghiêm túc, sau khi Allen Turing công bố các kết quả đầu tiên.

Phát hiện của Turing:

- Xây dựng máy tính dựa trên các phép toán cơ sở của logic như AND, OR, NOT.

- Máy tính được điều khiển bởi các chương trình lưu ở bộ nhớ trong.

Năm 1956: Chương trình dẫn xuất kết luận trong hệ hình thức được công bố.

Năm 1959: Chương trình chứng minh các định lý hình học phẳng và chương trình giải quyết bài toán tổng quát GPS (General Problem Solving) được đưa ra.

Năm 1960: Mcathy đưa ra ngôn ngữ lập trình LISP.

Thuật ngữ TTNT do Marvin Minsky (của MIT- Massachussets Institute of Technology) đưa ra năm 1961. Đến đây các nghiên cứu về trí tuệ nhân tạo bắt

đầu phát triển mạnh. Trong thời gian này, các chương trình tính tích phân, chơi cờ, chứng minh định lý toán học cũng được công bố.

Năm 1961: Chương trình tính tích phân bất định.

Năm 1963: Các chương trình Heuristics (suy nghiệm, phát hiện), chương trình chứng minh định lý hình học không gian có tên "Tương tự", chương trình chơi cờ của Samuel.

Năm 1964: Chương trình giải phương trình đại số sơ cấp, chương trình trợ giúp ELIZA (phân tích tâm lý).

Năm 1966: Chương trình phân tích và tổng hợp tiếng nói.

Năm 1968: Chương trình nhận dạng hình ảnh, Robot chế tạo dựa theo đồ án mắt- tay, chương trình học nói.

Hạn chế của giai đoạn này là khả năng của máy tính bị hạn chế về cả bộ nhớ và thời gian thực hiện. Sang những năm 70, máy tính phát triển đáng kể về chất (bộ nhớ, tốc độ), song TTNT chưa có thành công đáng kể vì trong quá trình tìm kiếm lời giải bài toán thường gặp phải sự bùng nổ tổ hợp.

Cuối những năm 70 các nhà nghiên cứu đã đạt được những thành công đáng kể trong các lĩnh vực:

- Xử lý ngôn ngữ tự nhiên
- Biểu diễn tri thức
- Lý thuyết giải quyết vấn đề
- Hệ chuyên gia.

Năm 1972, ngôn ngữ Prolog ra đời là một sự kiện quan trọng trong sự phát triển của khoa học TTNT.

Năm 1981, ngôn ngữ Prolog được chọn làm ngôn ngữ cơ sở trong dự án xây dựng máy tính thế hệ thứ 5 của Nhật. Điều này đã làm thay đổi khá nhiều tình hình phát triển TTNT ở Mỹ và Châu Âu.

Cuối những năm 80, đầu những năm 90 có khá nhiều sản phẩm dân dụng trình độ cao sử dụng TTNT như: máy giặt, máy ảnh.

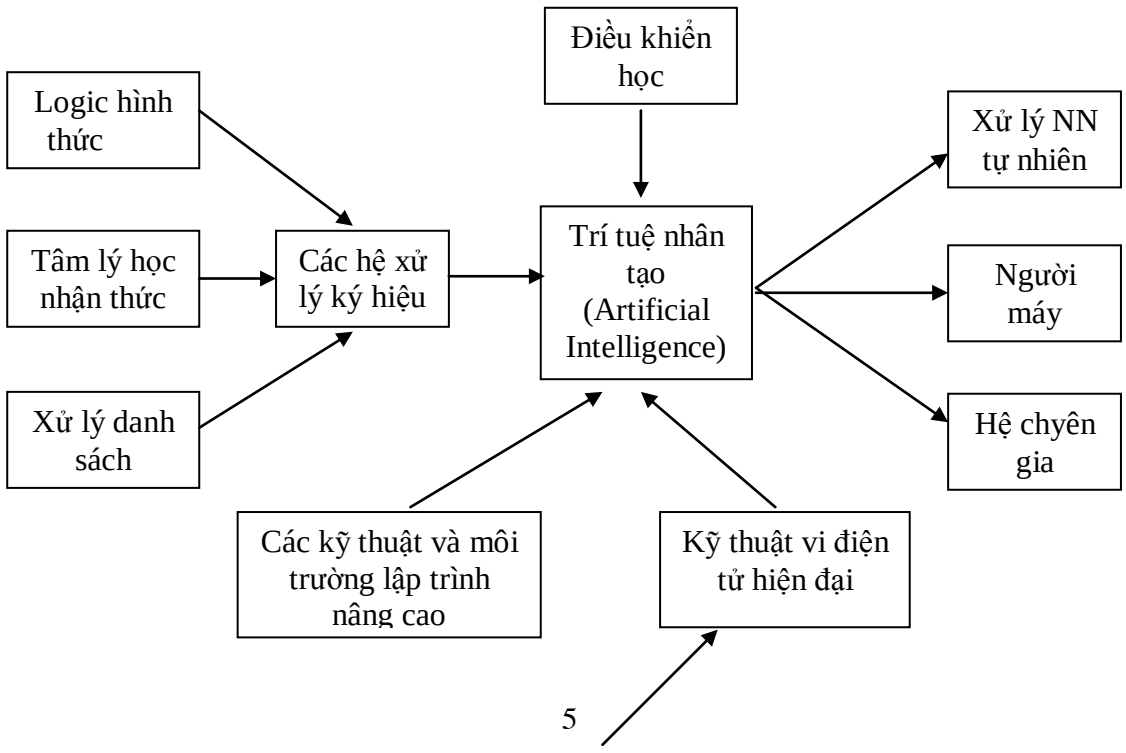
Những năm 90, một số lĩnh vực trí tuệ nhân tạo được phát triển mạnh như:

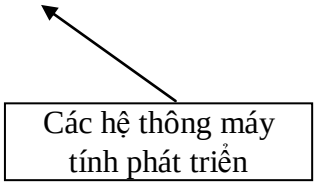
+ Các hệ thống nhận dạng và xử lý hình ảnh
+ Mạng Neuron
+ Kỹ thuật xử lý tri thức, dữ liệu hỗn hợp
+ Hệ hỗ trợ ra quyết định dựa trên tri thức
+ Biểu diễn tri thức và dữ liệu hướng đối tượng
+ Kỹ thuật Multimedia (đa phương tiện), Hypertext (siêu văn bản).

1.2. NHỮNG TIỀN ĐỀ CƠ BẢN CỦA TTNT

Những tiền đề ban đầu cho sự ra đời của TTNT là những nghiên cứu lý thuyết sâu sắc của các chuyên gia về: Logic hình thức, tâm lý học nhận thức và điều khiển học. Tuy vậy những tiến bộ trong kỹ thuật vi điện tử cũng là một yếu tố tạo tiền đề cho sự thành công của các chương trình TTNT.

Những tiền đề hình thành và các hướng nghiên cứu, ứng dụng cơ bản và các hướng phát triển chính của trí tuệ nhân tạo được thể hiện bởi sơ đồ sau:





1.3. CÁC KHÁI NIỆM CƠ BẢN CỦA KHOA HỌC TTNT

1.3.1 Trí tuệ của con người (Human Intelligence)

Hiện nay chưa có định nghĩa thống nhất về trí tuệ con người. Sau đây là một số định nghĩa được chú ý nhiều nhất:

A.Turing:

"Trí tuệ là những gì có thể đánh giá được thông qua các phép kiểm tra (test) thông minh".

Từ điển bách khoa toàn thư Webster:

"Trí tuệ là khả năng:

1. Phản ánh một cách thích hợp lại những tình huống mới thông qua hiệu chỉnh hành vi một cách thích đáng.

2. Hiểu rõ những mối liên hệ qua lại giữa các sự kiện của thế giới bên ngoài nhằm đưa ra những hành động phù hợp để đạt tới mục đích nào đó".

Các chuyên gia tâm lý học nhận thức chỉ ra rằng hoạt động trí tuệ của con người bao gồm 4 thao tác:

1. Xác định tập các đích (goals) cần đạt tới.

2. Thu tập sự kiện (facts) và các luật suy diễn (interence) để đạt tới các đích đặt ra.

3. Thu gọn quá trình suy luận nhằm xác định nhanh chóng tập những luật suy diễn có thể sử dụng được để đạt tới một đích trung gian nào đó.

4. Áp dụng các cơ chế suy diễn cụ thể, dựa trên các thao tác thu gọn quá trình suy luận và các sự kiện trung gian mới được tạo ra để dẫn dắt từ những sự kiện ban đầu đi tới những đích đặt ra.

Ví dụ 1.1.

* Thao tác1. Cần xác định cách đi tốt nhất từ nhà tới cơ quan.

* Thao tác2.

+ Các sự kiện: Thời tiết, sức khoẻ, ngày nghỉ hay làm việc, loại phương tiện, khoảng cách, giá cả từng loại phương tiện, thời gian, tình hình kinh tế, giá cả nhiên liệu,...

+ Các luật suy diễn: Giả sử ta thu nạp được các luật suy diễn sau:

Luật 1: Nếu hôm đó là ngày nghỉ Thì không phải đi làm

Luật 2: Nếu thời tiết xấu Thì công viên không có người

Luật 3: Nếu thời tiết xấu Thì nên đi đường nhựa

Luật 4: Nếu thời tiết đẹp Thì đi đường tắt

Luật 5: Nếu tình hình chính trị xấu Thì tình hình kinh tế xấu

Luật 6: Nếu giá cả tăng Thì tình hình kinh tế xấu

Luật 7: Nếu thời tiết xấu và đi bằng xe Thì đi bằng xe buýt

Luật 8: Nếu thời tiết đẹp và đi bằng xe Thì đi xe đạp

Luật 9: Nếu đường xa Thì đi bằng xe

Luật 10: Nếu đường gần Thì đi bộ

Luật 11: Nếu đi đường nhựa Thì đi bằng xe.

* Thao tác 3: Giả sử hôm đó thời tiết xấu, khi đó cơ chế thu gọn quá trình suy luận cho phép chỉ xét các luật liên quan tới thời tiết xấu (luật 2, 3, 7) hoặc chỉ các luật có liên quan tới đích đặt ra (luật 1, 3, 4, 7, 8, 9, 10, 11).

* Thao tác 4: Chú ý tới các luật thoả mãn điều kiện "thời tiết xấu", chỉ có luật 2, 3. Tuy nhiên chỉ có luật 3 được lưu lại và ta có sự kiện khẳng định mới là "đi đường nhựa". Sự kiện này được nạp thêm vào tập các sự kiện đã biết. Tiếp tục

quá trình, ta đi đến kết luận để đi từ nhà tới cơ quan, nếu thời tiết xấu, nên đi bằng xe buýt (xem sơ đồ trên).

1.3.2. Trí tuệ máy (Machine Intellgence)

Trí tuệ máy chưa có định nghĩa, song có một số dấu hiệu quan trọng để đánh giá trí tuệ máy là:

1. Khả năng học.
2. Khả năng mô phỏng các hành vi sáng tạo của con người, nghĩa là có thể giải quyết một bài toán sáng tạo nào đó giống như một chuyên gia (chương trình chơi cờ).
3. Khả năng trừu tượng hoá, khái quát hoá và suy diễn.
4. Khả năng tự giải thích hành vi.
5. Khả năng thích nghi với tình huống mới trong đó gồm có khả năng thu nạp tri thức và dữ liệu.
6. Khả năng xử lý các biểu diễn hình thức như các kí hiệu tượng trưng, danh sách.
7. Khả năng sử dụng các tri thức, Heuristics.
8. Khả năng xử lý các thông tin không đầy đủ, không chính xác.

1.3.3. Vai trò của TTNT trong CNTT

Khoa học TTNT có nhiệm vụ:

- Nghiên cứu các kỹ thuật làm cho máy tính có thể "suy nghĩ một cách thông minh" .
- Mô phỏng quá trình suy nghĩ của con người khi đưa ra những quyết định, lời giải nhờ tìm kiếm trong không gian bài toán hay phân rã bài toán con, do vậy nó chia nhỏ quá trình suy nghĩ này thành những bước cơ bản.
- Tạo nên cách tiếp cận đơn giản và có cấu trúc để xây dựng các chương trình ra quyết định phức hợp đòi hỏi phải dựa trên những tri thức nhất định.

- TTNT làm cho máy tính có khả năng suy nghĩ. Có thể mô phỏng quá trình học của con người, trên cơ sở đó thu nạp được những thông tin mới phục vụ cho quá trình suy diễn sau đó.

Sự ra đời và phát triển của trí tuệ nhân tạo tạo nên những bước nhảy vọt về chất trong kỹ thuật và kỹ nghệ xử lý thông tin. Nó là cơ sở của công nghệ xử lý thông tin mới, độc lập với công nghệ xử lý thông tin truyền thống dựa trên văn bản giấy tờ.

Những đặc điểm của công nghệ xử lý thông tin mới dựa trên nền tảng TTNT bao gồm:

1. Nhờ những công cụ hình thức hoá, các tri thức thủ tục và tri thức mô tả có thể biểu diễn được trong máy tính, quá trình giải quyết bài toán được tiến hành hữu hiệu hơn.

2. Các hệ thống TTNT có tính thích nghi cao, mềm dẻo đối với các lớp bài toán thuộc nhiều lĩnh vực khác nhau.

3. Các hệ thống trí tuệ nhân tạo hoạt động dựa trên tri thức và các Heuristics, trong khi đó các hệ thống xử lý thông tin truyền thống dựa trên các thuật giải.

4. Việc bảo trì các hệ thống TTNT do các kỹ sư xử lý tri thức đảm nhận, trong khi đó việc bảo trì các chương trình do các lập trình viên chịu trách nhiệm.

Ta có thể so sánh kỹ thuật lập trình truyền thống và kỹ thuật xử lý ký hiệu trong TTNT như sau:

Lập trình truyền thống	***Xử lý ký hiệu***
- Định hướng xử lý các dữ liệu (số, văn bản).	- Định hướng xử lý các ký hiệu tượng trưng.
- CSDL được đánh địa chỉ số.	- Cơ sở tri thức được cấu trúc theo theo các ký hiệu.
- Xử lý theo thuật toán.	- Xử lý theo các thuật giải Heuristic, cơ chế lập luận.
- Xử lý tuần tự hay theo mẻ.	- Xử lý theo chế độ tương tác cao (hội thoại ngôn ngữ tự nhiên).

- Không giải thích trong quá trình thực hiện.	- Có thể giải thích hành vi hệ thống trong quá trình thực hiện.

1.3.4. Các kỹ thuật TTNT

Các kỹ thuật TTNT thường thiên về xử lý các ký hiệu tượng trưng, vì vậy nó thường khá phức tạp khi cài đặt cụ thể. Hơn nữa nó đòi hỏi phải sử dụng những tri thức chuyên môn thuộc vào các lĩnh vực khác nhau. Những tri thức này có các đặc điểm sau:

- Khối lượng tri thức cần để giải quyết bài toán có thể rất lớn.
- Khó có thể đặc trưng hoá và biểu diễn tri thức chuyên gia một cách chính xác.
- Tri thức đối với bài toán thường xuyên bị thay đổi.

Do vậy các đặc tính quan trọng của kỹ thuật TTNT:

- Đạt được mức tổng quát.
- Dễ hiểu đối với các chuyên gia con người cung cấp tri thức cho hệ thống.
- Dễ sửa đổi, dễ hiệu chỉnh lỗi và dễ đưa vào những thay đổi cần thiết.
- Dễ sử dụng trong nhiều tình huống khác nhau, kể cả khi các tri thức không chính xác và đầy đủ.
- Dễ thu hẹp khả năng cần xét để đi tới lời giải cuối cùng.

Với các yêu cầu trên ta có các phương pháp và kỹ thuật trí tuệ nhân tạo cơ bản sau:

- Các phương pháp biểu diễn tri thức và kỹ nghệ xử lý tri thức
- Các phương pháp giải quyết vấn đề
- Các phương pháp Heuristics
- Các phương pháp học
- Các ngôn ngữ TTNT.

Ngoài ra, các kỹ thuật của tin học truyền thống như: Xử lý danh sách, đệ quy, quay lui, sử dụng cú pháp hình thức cũng có liên quan trực tiếp đến TTNT.

1.3.5. Các thành phần trong hệ thống TTNT

Mỗi hệ thống TTNH gồm hai thành phần cơ bản sau:

+ Các phương pháp biểu diễn vấn đề, các phương pháp biểu diễn tri thức

+ Các phương pháp tìm kiếm trong không gian bài toán, các chiến lược suy diễn.

Hai thành phần này có quan hệ chặt chẽ với nhau. Thể hiện ở chỗ việc lựa chọn phương pháp biểu diễn tri thức sẽ quyết định phương pháp giải quyết vấn đề tương ứng có thể áp dụng. Ngược lại, phương pháp tìm kiếm sẽ có hiệu quả nếu phương pháp biểu diễn tri thức thích hợp.

Dựa trên các thành phần của các hệ thống TTNT, có thể chia các hệ thống TTNT thành các loại sau:

1. Các hệ thống tìm kiếm thông tin, các hệ thống hỏi đáp thông minh cho phép hội thoại giữa những người sử dụng đầu cuối không chuyên tin, với cơ sở tri thức và cơ sỡ dữ liệu thông qua ngôn ngữ chuyên ngành gần với ngôn ngữ tự nhiên.

2. Các hệ thống suy diễn- tính toán cho phép giải quyết những bài toán phức tạp dựa trên các mô hình toán học và tri thức chuyên gia.

3. Các hệ chuyên gia cho phép sử dụng các tri thức chuyên gia trong các lĩnh vực tri thức tản mạn.

1.3.6. Các cách tiếp cận trong TTNT

Từ đầu, các công trình nghiên cứu về TTNT được chia thành 2 hướng cơ bản:

+ *Hướng phỏng sinh học:* Mô phỏng hoàn toàn các hoạt động của bộ não, các tính chất tâm sinh lý của nó, trên cơ sở đó tái tạo trong máy tính hoặc các thiết bị kỹ thuật những khía cạnh khác nhau của TTNT.

+ *Hướng phỏng vật lý:* Định hướng vào các khía cạnh thực hành trong đó máy tính là công cụ thử nghiệm các chương trình cho phép đạt tới kết quả giống như hoạt động sáng tạo của con người.

Tuy nhiên, trong những năm gần đây hai hướng nghiên cứu này đã xích lại gần nhau.

Bốn cách tiếp cận khác nhau trong TTNT:

- Tạo lập các mạng thông minh
- Tái tạo quá trình tiến hoá nhân tạo
- Lập trình Heuristics
- Biểu diễn và xử lý tri thức.

Ở đây ta hiểu Mạng thông minh nhân tạo bao gồm một số lượng lớn các phần tử đơn giản (máy tính hoặc thiết bị vật lý khác) và mối liên hệ giữa chúng. Chẳng hạn mạng neuron nhân tạo nhằm tái tạo những phương thức tổ chức và cơ chế hoạt động của các neuron trong bộ não người. Ưu điểm của mạng neuron là khả năng thích nghi và khả năng học. Khó khăn của nó là khó tạo ra được 10^{10} neuron như bộ não người.

Tái tạo quá trình tiến hoá nhân tạo là cách tiếp cận hướng tới xây dựng các hệ thống chương trình có khả năng mô phỏng quá trình tiến hoá tự nhiên thông qua hai hoạt động cơ bản là đột biến và lựa chọn.

Các nghiên cứu về lập trình Heuristics có những mục đích:

- Giải thích bản chất của trí tuệ tự nhiên
- Sử dụng "trí tuệ máy" để giải quyết các bài toán sáng tạo khó.

Biểu diễn và xử lý tri thức là cách tiếp cận dựa trên những đặc tả chính của bài toán đặt ra. Cách tiếp cận này thường sử dụng trong các hệ chuyên gia.

1.4. CÁC LĨNH VỰC NGHIÊN CỨU VÀ ỨNG DỤNG CƠ BẢN CỦA TTNT

Có thể phân chia các lĩnh vực nghiên cứu và ứng dụng quan trọng trong TTNT theo bốn hướng cơ bản sau:

1. Mô hình hoá trên máy tính các chức năng khác nhau trong quá trình sáng tạo: Các trò chơi, chứng minh định lý tự động, tổng hợp các chương trình, phân tích và tổng hợp các tác phẩm nghệ thuật,...

2. Nâng cao khả năng trí tuệ "bên ngoài" của máy tính, bao gồm các nghiên cứu cơ bản và ứng dụng, gắn liền với các giao tiếp, hội thoại bằng cách sử dụng các kỹ thuật tìm kiếm và suy diễn.

3. Nâng cao trí tuệ "bên trong" của máy tính trên cơ sở chế tạo các máy tính thế hệ mới với các kiến trúc vật lý mới dựa trên các nguyên lý của TTNT.

4. Chế tạo người máy thông minh, có khả năng thực hiện các thao tác phức tạp, biết" suy nghĩ" và "hành động" để đạt được đích đặt ra.

1.4.1. Các lĩnh vực nghiên cứu cơ bản của TTNT

1. Lý thuyết giải quyết vấn đề và các kỹ thuật suy diễn thông minh

Các phương pháp giải quyết vấn đề bao gồm:

- Phương pháp biểu diễn bài toán trong không gian trạng thái
- Phương pháp quy bài toán về các bài toán con
- Phương pháp GPS
- Phương pháp hình thức sử dụng cách tiếp cận logic.

2. Lý thuyết tìm kiếm Heuristics

Bao gồm các phương pháp và kỹ thuật tìm kiếm, sử dụng các tri thức nảy sinh từ bản thân bài toán để rút ngắn quá trình giải, nhanh chóng thu được kết quả.

3. Lý thuyết biểu diễn tri thức và kỹ thuật xử lý tri thức

Bốn phương pháp biểu diễn tri thức:

- Dùng logic mệnh đề, logic vị từ
- Dùng các hệ sản xuất
- Dùng frame
- Dùng mạng ngữ nghĩa.

4. Lý thuyết nhận dạng

Phát triển theo các hướng tiếp cận:

- Lý thuyết thống kê về nhận dạng

- Lý thuyết cấu trúc về nhận dạng
- Lý thuyết đại số về nhận dạng
- Lý thuyết Heuristics về nhận dạng.

* Hiện nay, vấn đề nhận dạng thông tin hình ảnh và tiếng nói đang là một thách thức đối với các chuyên gia TTNT.

* Cách tiếp cận sử dụng mạng Neuron nhân tạo đang có nhiều hứa hẹn.

5. Các ngôn ngữ trí tuệ nhân tạo

- Ngôn ngữ LISP (List Processing) - 1960
- PROLOG (Programinig for Logics) - 1972
- CLIPS - 1980.

1.4.2. Các ứng dụng cơ bản của TTNT

Các ứng dựng cơ bản của TTNT bao gồm các lĩnh vực:

1. Kỹ thuật người máy
2. Các chương trình trò chơi
3. Xử lý ngôn ngữ tự nhiên
4. Các hệ thống xử lý tri thức và dữ liệu tích hợp.

1.4.3. Những nghiên cứu hiện đại xung quanh TTNT

Những năm gần đây các nghiên cứu TTNT tập trung vào các lĩnh vực sau:

- Lập trình logic, xử lý phép phủ định trong lập luận
- Các phương pháp học, thu nạp tri thức
- Giải thuật di truyền
- Mạng neuron
- Các hệ hỗ trợ quyết định dựa trên tri thức.

1.5. NHỮNG VẤN ĐỀ CHƯA ĐƯỢC GIẢI QUYẾT TRONG TTNT

Hai vấn đề thách thức còn mở đối với các nhà nghiên cứu TTNT trong giai đoạn hiện nay là:

1. Trí tuệ nhân tạo có mô phỏng được phần lớn hoạt động của bộ não con người?

2. Liệu có tồn tại những khía cạnh trí tuệ con người về nguyên tắc là không mô phỏng được trên máy tính?

Với các thành tựu đạt được cho thấy rằng các dự án xây dựng máy tính có khả năng suy nghĩ là có tính thực tiễn, song trong nhiều lĩnh vực máy tính còn thua xa so với hoạt động của hệ thần kinh con người.

1.5.1. Sự khác nhau cơ bản trong hoạt động giữa máy tính và bộ não người

Sự khác nhau cơ bản trong hoạt động giữa máy tính và bộ não người được thể hiện rõ nét nhất ở các điểm sau:

- *Khả năng xử lý song song:* Khả năng xử lý song song của bộ não rất lớn, máy tính chưa thể đạt đến được.

Khả năng diễn giải: Con người có khả năng xem xét cùng một vấn đề theo nhiều phương pháp khác nhau, để từ đó diễn giải theo cách dễ hiểu nhất. Các hệ thống TTNT tạo không thể mô phỏng được điều này.

- *Xử lý các đại lượng rời rạc và liên tục:* Bộ não xử lý thông tin liên tục một cách dễ dàng như là một hoạt động bản năng. Nó dễ dàng kết hợp việc xử lý thông tin liên tục với thao tác xử lý thông tin rời rạc. Đối với Máy tính, xử lý thông tin liên tục còn là một thách đố ghê gớm.

- *Khả năng học:* Đối với con người, khả năng học như là bản năng của bộ não. Con người có thể tiếp thu các tri thức từ thế giới bên ngoài, có khả năng học khi thông tin về thế giới xung quanh không đầy đủ, không chính xác, phụ thuộc vào tình huống. Máy tính chưa có những khả năng này.

- *Khả năng tự tổ chức:* Khả năng tự tổ chức, điều chỉnh hành vi là điểm nổi bật trong hoạt động của bộ não con người. Nó cùng với khả năng học tạo nên khả năng thích nghi làm cho con người có thể hoà nhập với nhiều tình huống

khác nhau. Với máy tính, để thực hiện được khả năng này đòi hỏi phải có những đột biến trong chế tạo phần cứng. Đến nay chưa đáp ứng được.

1.5.2. Những vấn đề đặt ra trong tương lai

1. Nghiên cứu và thử nghiệm mạng neuron, các hệ thống TTNT mô phỏng chức năng hoạt động của bộ não với các khả năng học, tự tổ chức, tự thích nghi, tổng quát hoá, xử lý song song, có khả năng diễn giải, xử lý thông tin liên tục và rời rạc.

2. Nghiên cứu và tạo lập các hệ thống có giao tiếp thân thiện giữa người với máy trên cơ sở nghiên cứu nhận thức máy, thu thập và xử lý tri thức, thu thập và xử lý thông tin hình ảnh và tiếng nói.

3. Nghiên cứu các phương pháp biểu diễn tri thức và các phương pháp suy diễn thông minh, các phương pháp giải quyết vấn đề đối với các bài toán phát biểu không đầy đủ, không chính xác, các bài toán phụ thuộc không gian, thời gian đòi hỏi xử lý tích hợp các tri thức và dữ liệu

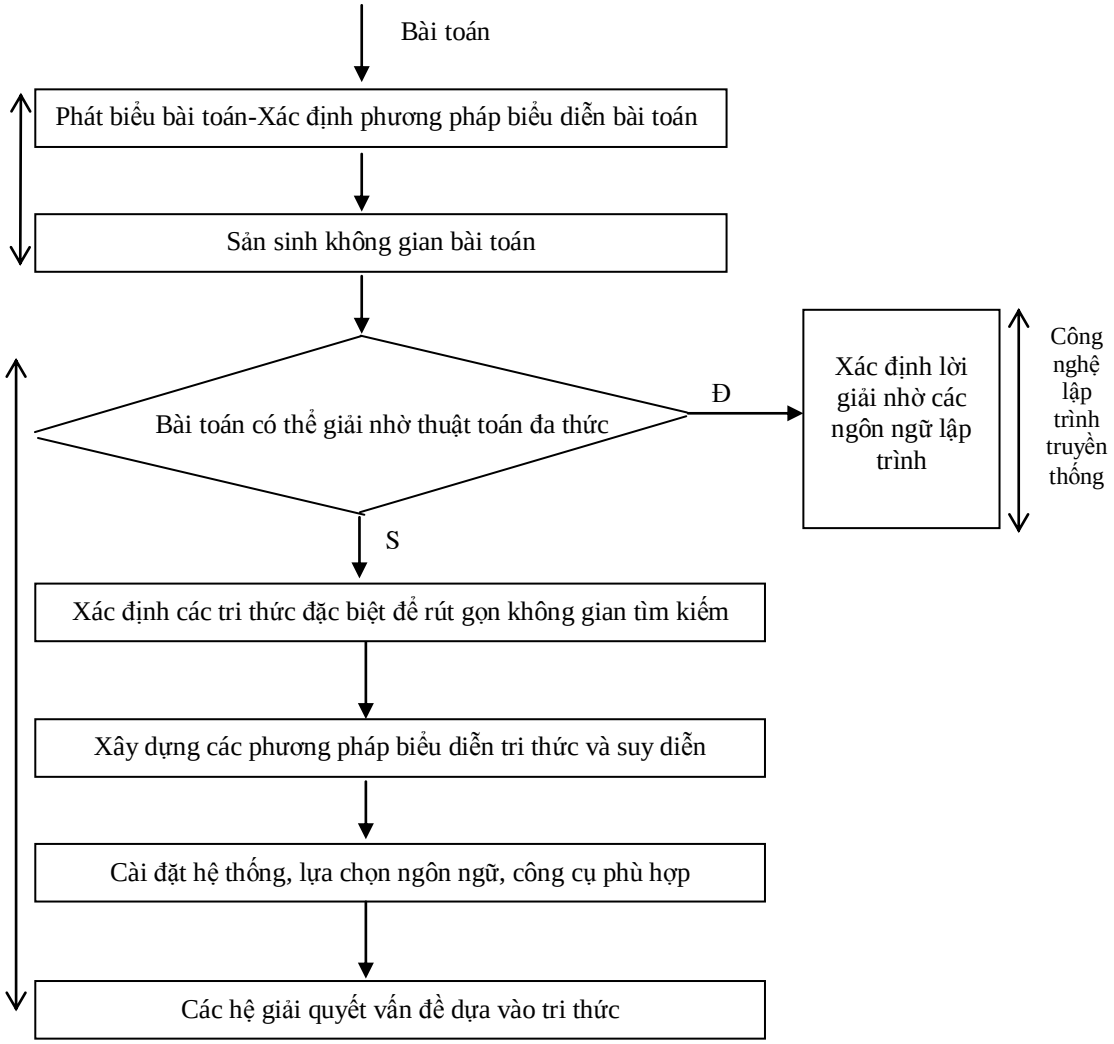
Chương II

CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

2.1. GIẢI QUYẾT VẤN ĐỀ VÀ KHOA HỌC TTNT

Từ đây cho hết giáo trình này các thuật ngữ "vấn đề" và "bài toán" được dùng tương đương.

Có thể xem lịch sử phát triển của TTNT gắn liền với sự bùng nổ các kỹ thuật giải quyết vấn đề, từ các kỹ thuật Heuristics đến các kỹ thuật xử lý tri thức (xem hình vẽ 2.1). Nhiều hoạt động trong thực tiễn của con người (tính toán, quản lý, trò chơi, giải đáp câu đố,...) đều đòi hỏi sự tham gia của trí tuệ.



Hình 2.1. Những khía cạnh khác nhau của trí tuệ nhân tạo

Máy tính trở thành công cụ đắc lực giúp con người trong công việc xử lý thông tin và giải quyết các nhiệm vụ ở mức độ trí tuệ ngày càng cao. Theo một nghĩa nào đó các máy tính hiện nay đã được trang bị trí tuệ nhân tạo. Trong một số lĩnh vực, máy tính vượt quá khả năng của con người. Tuy nhiên, có những bài toán con người giải quyết đơn giản thì máy tính chưa làm được.

Cốt lõi của mọi hoạt động trí tuệ diễn ra trong máy tính cũng như bộ não con người là sự vận dụng linh hoạt các kỹ thuật giải quyết vấn đề. Trong chương này, chúng ta quan tâm đến các phương pháp giải quyết bài toán dựa trên một cách biểu diễn bằng mô hình hoá bài toán nào đó.

2.2. GIẢI QUYẾT VẤN ĐỀ CỦA CON NGƯỜI

Cách giải quyết vấn đề của con người là mô hình thực tiễn quan trọng để các chuyên gia TTNT tìm cách mô phỏng lại trên máy tính quá trình giải quyết bài toán. Có hai bộ phận nghiên cứu quá trình giải quyết vấn đề của con người:

+ *Khoa học về nhận thức:* Nghiên cứu quá trình tổ chức, lưu trữ, truy nhập, xử lý và thu nạp tri thức trong bộ não người.

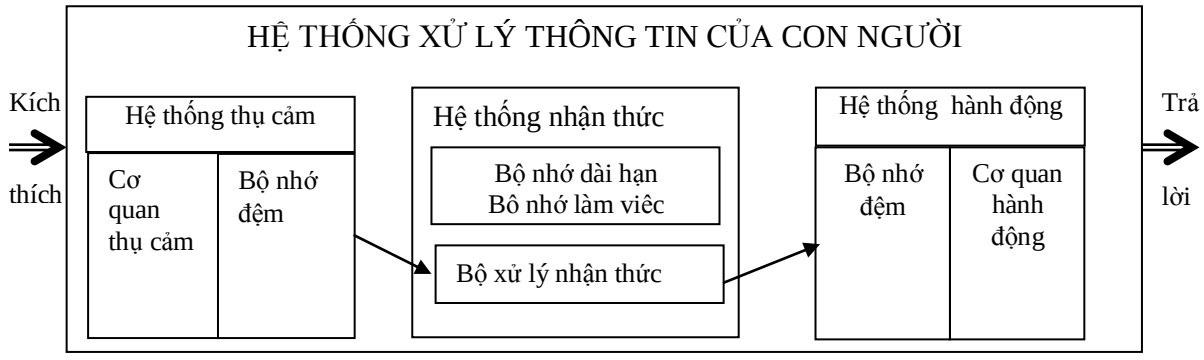
+ *Tâm lý học nhận thức và khoa học điều khiển:* Tạo ra các mô hình tổ chức bộ não.

2.2.1. Quá trình xử lý thông tin của con người

Mô hình xử lý thông tin trong bộ não gồm 3 hệ thống con sau:

- Hệ thống cảm nhận (perception subsystem)
- Hệ thống nhận thức (cognitive subsystem)
- Hệ thống hoạt động(action subsystem).

Ta có thể mô tả quá trình xử lý thông tin của con người như sau:



Hình 2.2. Tổng quan về hệ thống xử lý thông tin của con người

2.2.2. Giải quyết vấn đề của con người

Giải quyết vấn đề của con người là một trường hợp riêng của quá trình xử lý thông tin trong bộ não. Thực chất, đó là việc tìm cách đi từ tình huống ban đầu nào đó đến đích. Giải quyết vấn đề là một hoạt động đặc biệt của hệ thần kinh cần tới quá trình suy nghĩ, tìm kiếm trong không gian lời giải bộ phận để đi đến lời giải cuối cùng.

Tuy nhiên, cần lưu ý rằng không phải mọi hoạt động xử lý thông tin đều là giải quyết vấn đề.

Các chiến lược giải quyết vấn đề của con người:

+ Ước lượng mức độ phức tạp của vấn đề đặt ra:

- Nếu đơn giản, giải quyết ngay bằng các thao tác cơ sở.

- Nếu phức tạp, các cơ quan não tìm cách hiểu chi tiết nội dung của vấn đề để mã hoá, tìm phương pháp phù hợp.

+ Nới lỏng một vài ràng buộc của bài toán.

+ Phương pháp chia thành các bài toán con: Từ bài toán phức tạp, con người chia thành các bài toán nhỏ, ít phức tạp cho đến khi gặp các bài toán sơ cấp, giải quyết được ngay.

+ Tổng quát hoá bài toán : Chuyển các thông tin bên ngoài thành các kí hiệu làm cho bài toán dễ giải hơn. Quá trình này tạo ra một mô hình trí tuệ của bài toán, mô hình này thường được gọi không gian bài toán.

** Không gian bài toán bao gồm:*

- Các dạng mẫu kí hiệu, mỗi dạng biểu diễn một trạng thái hay một tình huống bài toán.

- Các mối liên kết giữa các dạng mẫu ký hiệu, mỗi mối liên kết tương ứng với các phép biến đổi từ dạng này sang dạng khác.

2.3. PHÂN LOẠI VẤN ĐỀ, CÁC ĐẶC TRƯNG CƠ BẢN CỦA VẤN ĐỀ

Trước khi xem xét việc phân loại vấn đề, ta xét các bài toán sau:

Bài toán 2.1. Bài toán trò chơi n^2-1 số (n[redacted], n>2).

Trong bảng ô vuông n hàng, n cột, mỗi ô chứa 1 số nằm trong phạm vi từ 1 đến n^2-1 sao cho không có 2 ô có cùng giá trị. Có một ô trong bảng bị bỏ trống. Xuất phát từ một cách sắp xếp nào đó các số trong bảng hãy dịch chuyển ô trống sang phải, sang trái, lên trên, xuống dưới (nếu có thể) để đưa về dạng sau:

1	2	3
4	5	6
7	8	

a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

b)

Hình 2.3. Hình trạng đích trong trò chơi n^2-1 số

a) n=3 b) n= 4

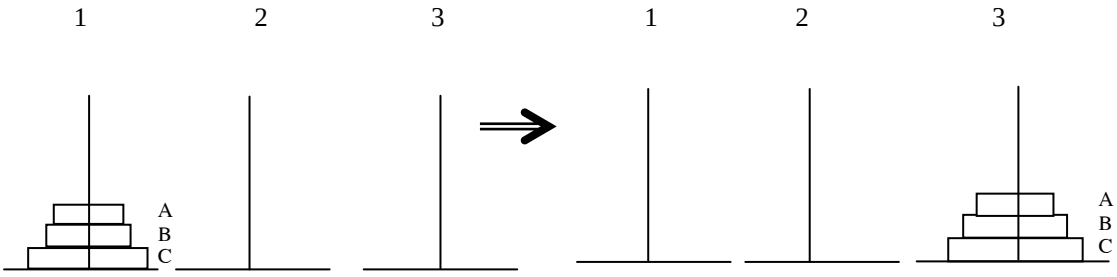
Bài toán 2.2. Bài toán tháp Hà Nội.

Cho 3 cọc 1, 2, 3. Ở cọc 1 ban đầu có n đĩa sắp theo thứ tự nhỏ dần từ dưới lên trên. Hãy tìm cách chuyển n đĩa đó sang cọc 3 với điều kiện:

- Mỗi lần chỉ chuyển 1 đĩa

- Chỉ sử dụng cọc trung gian 2

- Trong mỗi cọc, ở mỗi bước không cho phép đĩa to nằm trên đĩa nhỏ.



Hình 2.4. Bài toán tháp Hà Nội với n=3

Bài toán 2.3. Đố chữ

Hãy thay các chữ cái bằng các chữ số từ 0 đến 9 sao cho không có 2 chữ cái được thay bởi cùng 1 số và thoả mãn:

S END
\+ MORE
MONEY

2.3.1. Phân loại vấn đề

Các vấn đề được phân ra làm hai loại:

+ *Vấn đề (bài toán) phát biểu chỉnh (well-formed problems):* Đó là các bài toán có thể biết được hình trạng đầu, hình trạng đích và có thể quyết định khi nào vấn đề được coi là giải quyết xong. Các bài toán 2.1-2.3 trên đây là những vấn đề được phát biểu chỉnh.

+ *Vấn đề (bài toán) phát biểu không chỉnh (ill-formed problems):* Đó là các vấn đề được phát biểu chưa đầy đủ, thiếu dữ kiện. Các bài toán chẩn đoán và điều trị bệnh, bài toán xác định chất lượng sản phẩm là các bài toán phát biểu không chỉnh.

2.3.2. Các đặc trưng cơ bản của vấn đề

Các vấn đề có các đặc trưng cơ bản sau:

- Bài toán có thể phân tích thành các bài toán dễ giải hơn không?
- Các bước giải của bài toán có thể bỏ qua hay huỷ bỏ hay không?
- Không gian bài toán có thể đoán trước hay không?
- Có tiêu chuẩn để xác định lời giải nào đó là tốt đối với bài toán không?
- Có cần tri thức để giải quyết bài toán hay điều khiển quá trình tìm kiếm không?
- Cơ sở tri thức để giải quyết bài toán có nhất quán về nội dung không?
- Có cần tương tác người máy trong quá trình giải quyết không?

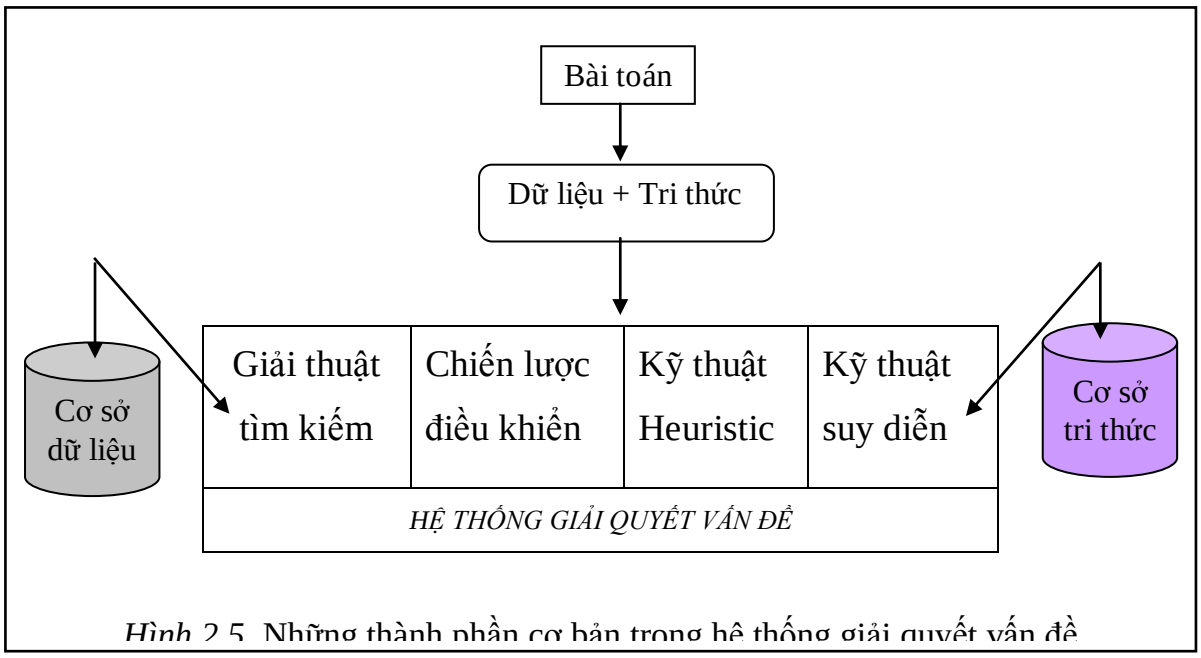
Để chọn lựa phương pháp phù hợp nhất đối với một bài toán cụ thể, cần phân tích bài toán theo các đặc trưng nói trên.

2.4. CÁC THÀNH PHẦN CƠ BẢN TRONG MỘT HỆ GIẢI QUYẾT VẤN ĐỀ

Một hệ giải quyết vấn đề bao gồm các thành phần cơ bản sau:

- Các phương pháp biểu diễn vấn đề

- Các giải thuật tìm kiếm
- Các chiến lược điều khiển
- Các kỹ thuật Heuristics
- Các kỹ thuật suy diễn.



Hình 2.5. Những thành phần cơ bản trong hệ thống giải quyết vấn đề

2.5. CÁC PHƯƠNG PHÁP BIỂU DIỄN VẤN ĐỀ

Các phương pháp biểu diễn vấn đề bao gồm:

- Phương pháp biểu diễn nhờ không gian trạng thái
- Phương pháp quy bài toán về các bài toán con
- Phương pháp sử dụng logic hình thức
- Phương pháp lựa chọn phương pháp biểu diễn thích hợp.

2.5.1. Phương pháp biểu diễn nhờ không gian trạng thái

Phương pháp biểu diễn vấn đề nhờ không gian trạng thái là phương pháp biểu diễn dựa trên hai khái niệm chính là trạng thái (state) và toán tử (operator). Trong đó mỗi trạng thái là một hình trạng của bài toán. Các toán tử là các phép biến đổi từ trạng thái này sang trạng thái khác. Mỗi phép biến đổi cho phép chuyển từ hình trạng này sang hình trạng khác ứng với các toán tử. Các hình trạng đầu, hình trạng cuối của bài toán được gọi là trạng thái đầu, trạng thái

cuối. Tập tất cả các trạng thái được sinh ra do xuất phát từ các trạng thái đầu và nhờ áp dụng các toán tử được gọi là không gian trạng thái (state space).

Một cách biểu diễn trực quan đối với không gian trạng thái và các toán tử là sử dụng đồ thị trong đó các đỉnh của đồ thị tương ứng với các trạng thái, còn các cung tương ứng với các toán tử.

2.5.2. Phương pháp quy bài toán về các bài toán con

Phương pháp giải quyết vấn đề quy bài toán về các bài toán con là một trong các phương pháp tiếp cận khá tinh tế, dựa vào khái niệm bài toán con. Theo cách này, ta có thể phân chia bài toán thành các bài toán con, các bài toán con lại được phân rã tiếp cho đến khi gặp được các bài toán sơ cấp cho phép xác định lời giải của bài toán ban đầu trên cơ sở lời giải của các bài toán con.

Ví dụ: Phương pháp tinh dần từng bước trong công nghệ lập trình.

2.5.3. Phương pháp sử dụng logic hình thức

Một trong các công cụ biểu diễn vấn đề hay được sử dụng là logic hình thức. Sở dĩ như vậy là vì để giải quyết vấn đề người ta cần tiến hành phân tích logic để thu gọn quá trình tìm kiếm và đôi khi nhờ phân tích logic có thể chứng tỏ được rằng một bài toán nào đó không thể giải được. Các dạng logic hình thức được sử dụng để biểu diễn vấn đề gồm:

- Logic mệnh đề
- Logic vị từ cấp 1.

Phương pháp biểu diễn vấn đề nhờ logic hình thức cho phép:

- Kiểm tra điều kiện kết thúc trong tìm kiếm đối với không gian trạng thái
- Kiểm tra tính áp dụng được của các toán tử
- Chứng minh không tồn tại lời giải.

Mục đích giải quyết vấn đề dựa trên logic hình thức là chứng minh một phát biểu nào đó trên cơ sở những tiền đề và luật suy diễn đã có.

2.5.4. Lựa chọn phương pháp biểu diễn thích hợp

Trong nhiều trường hợp, việc giải quyết bài toán dựa trên các thuật ngữ đã được dùng để phát biểu nó rất khó khăn. Trong trường hợp đó, thường người ta lựa chọn một dạng biểu diễn phù hợp nào đó đối với các dữ liệu của bài toán, làm cho bài toán trở nên dễ giải hơn. Việc lựa chọn phương pháp biểu diễn thích hợp nhằm:

- Tránh giải trực tiếp bài toán đặt ra ban đầu do những khó khăn liên quan tới kích cỡ, trọng số, tầm quan trọng và chi phí xử lý các dữ liệu của bài toán
- Bỏ bớt những thông tin thừa hoặc không quan trọng trong bài toán
- Tận dụng những phương pháp giải đã có đối với bài toán nhận được sau khi phát biểu lại.
- Cách phát biểu mới có thể cho phép thể hiện một vài tương quan nào đó giữa các yếu tố của bài toán nhằm thu gọn quá trình giải.

Sau khi đã giải quyết xong bài toán theo cách biểu diễn mới, ta diễn giải lời giải nhận được cho sát với bài toán ban đầu và chứng tỏ rằng cách diễn giải đó thực sự giải quyết được bài toán đặt ra.

2.5.5. Biểu diễn vấn đề trong máy tính

Để có thể giải quyết vấn đề trên máy tính, trước hết ta phải tìm cách biểu diễn lại vấn đề sao cho máy tính có thể hiểu được. Điều này có nghĩa là ta phải đưa các dữ liệu của bài toán về dạng có thể xử lý được trên máy tính. Sau đây là các cách biểu diễn vấn đề trong máy tính:

a. Cách biểu diễn dùng bảng (Array)

Sử dụng bảng để biểu diễn các hình trạng của bài toán. Chẳng hạn, đối với trò chơi 15 số, hình trạng đích (hình 2.3 b) tương ứng ma trận:

$$A = (a_{ij}) = \begin{cases} 4(i-1) + j & \text{với } (i, j) \neq (4, 4) \\ 0 & \text{với } (i, j) = (4, 4). \end{cases}$$

hoặc véc tơ A=(a_i), phần tử thứ 4(i -1) + j của nó nhận giá trị 4(i-1)+j, phần tử thứ 16 nhận giá trị 0.

b. Cách biểu diễn xâu ký hiệu

Sử dụng các xâu kí hiệu để biểu diễn các hình trạng của bài toán.

Ví dụ 2.1: Với bài toán tích phân bất định, hàm [redacted]được cho dưới dạng xâu ký tự [redacted]x)=x**2*cos(x+a) .

Ví dụ 2.2. Với hình trạng của một bàn cờ vua sau:

	TgT						
							ToĐ
				HĐ			
			MĐ				
		TgT					HT

Hình 2.6. Bàn cờ vua

Ta có thể biểu diễn hình trạng bàn cờ trong hình trên bởi các xâu ký tự:

"T, X, X , X, X, X, X, X, X,

X, TgT, X, X, X, X, X, X,

X, X, X, X, X, X, X, ToĐ

X, X, X, X, HĐ,X, X, X,

X, X, X,MĐ, X, X, X, X,

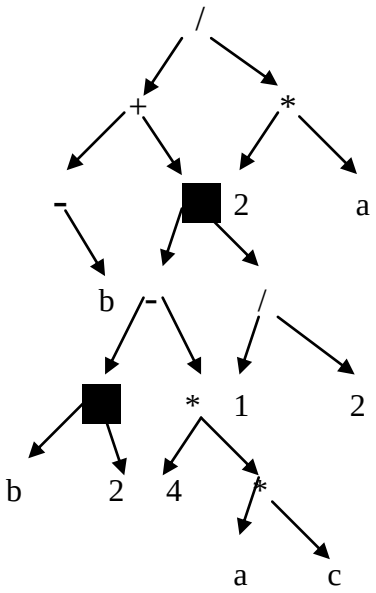
X, X, X, X, X, X, X, X,

X, X, TgT,X, X, X, X,HT “.

Ở đây X là ô trống, TgT là tượng trắng, ToĐ là tốt đen, HT, HĐ là hậu trắng, hậu đen, T có nghĩa là quân trắng đến lượt đi.

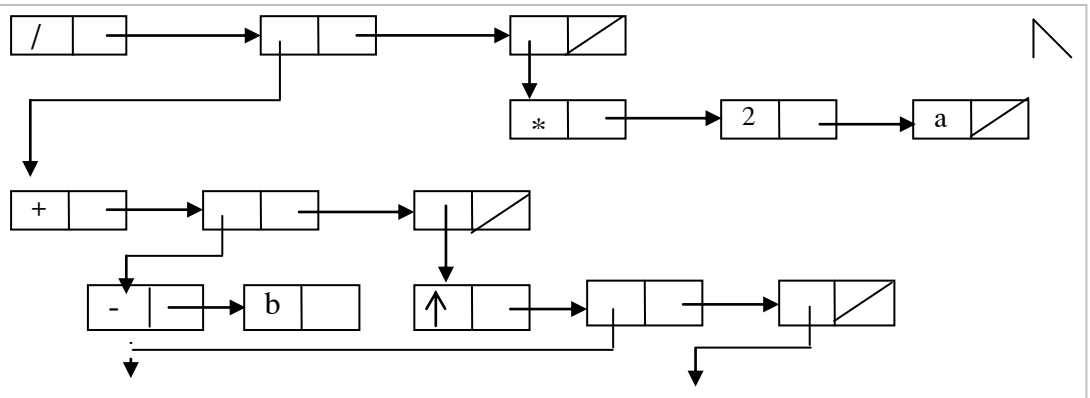
c. Cách biểu diễn dùng cấu trúc danh sách

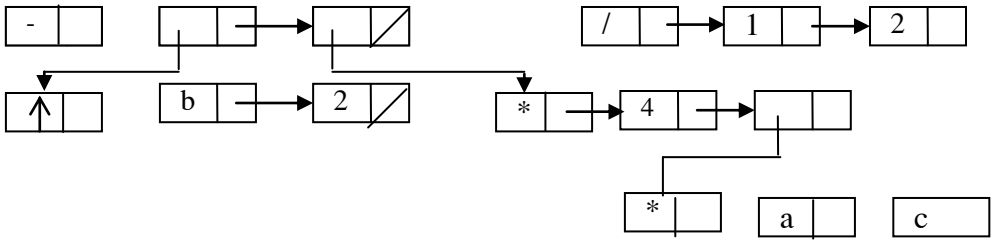
Một cách khác để biểu diễn vấn đề là sử dụng cấu trúc danh sách hoặc cấu trúc cây. Ví dụ biểu thức $\frac{(-b + (b^2 - 4ac)^{1/2})}{2a}$ có thể biểu diễn dạng cây như sau:



Hình 2.7. Cây biểu diễn biểu thức số học

Hoặc biểu diễn bằng cấu trúc danh sách như sau:





Hình 2.8. Cấu trúc danh sách biểu diễn biểu thức số học

2.5.6. Biểu diễn tri thức và giải quyết vấn đề

Có hai cách tiếp cận trong giải quyết vấn đề:

- Tổng quát hoá để đưa ra mô hình bài toán (xem mục 2.4.1, 2.4.2, 2.4.3).
- Cụ thể hoá trên cơ sở sử dụng các tri thức đặc tả (Chương 3).

Trên thực tế, có những bài toán không thể giải được nhờ sử dụng mô hình hoặc lời giải nhận được quá xa với lời giải thực tế. Trong trường hợp đó người ta sử dụng cách tiếp cận sử dụng tri thức đặc tả. Các phương pháp biểu diễn tri thức bao gồm: Frame, logic hình thức, mạng ngữ nghĩa và các hệ sản xuất.

2.6. CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ CƠ BẢN

2.6.1. Biểu diễn bài toán trong không gian trạng thái. Các chiến lược tìm kiếm trên đồ thị

a. Các mô tả trạng thái và toán tử

Phát biểu bài toán P_1: *Cho trạng thái đầu s_0 và tập trạng thái đích ĐICH.*

Hãy tìm dãy trạng thái $s_0, s_1,.., s_n$, sao cho s_n [redacted] ICH và với mỗi i=1,... n-1, từ trạng thái s_i có thể áp dụng toán tử biến đổi để nhận được s_{i+1}.

Có hai cách viết phép biến đổi trạng thái:

* Cách viết dùng các sản xuất

Ví dụ 2.3: Với bài toán phân tích cú pháp xem xâu [redacted] ó thuộc ngôn ngữ L(G) không, ở đây G = (N, T, P, S). Ta có:

- Mỗi trạng thái là một xâu nào đó trên tập N [redacted]
- Các toán tử là các sản xuất p: [redacted], chẳng hạn:

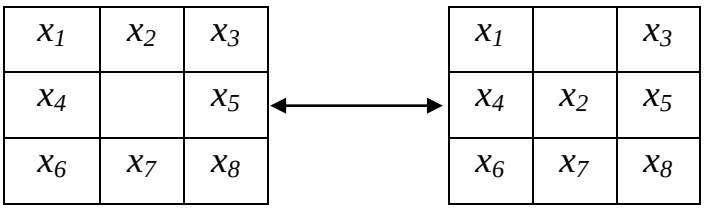
S[redacted]B

A[redacted]|a

B[redacted]|b

Ví dụ 2.4: Với bài toán 8 số ta có:

- Mỗi cách viết các số trong bảng ứng với một trạng thái
- Các sản xuất có dạng:



Ví dụ 2.5: Với bài toán tháp Hà Nội (n=3) ta có

- Mỗi trạng thái là một bộ ba (i j k), có nghĩa là đĩa C ở cọc i, đĩa B ở cọc j, đĩa A ở cọc k.

- Các toán tử dịch chuyển: (i j k)[redacted] j j) hoặc (i j k)[redacted] i)

* Cách viết dùng ký hiệu hàm:

Ví dụ 2.6: Với bài toán 8 số, ta có 4 loại toán tử ký hiệu là:

- dl: Dịch ô trống lên trên.
- dx: Dịch ô trống xuống dưới.
- df: Dịch ô trống sang phải.
- dt: Dịch ô trống sang trái.

Giả sử $A=(a_{ij})$, $a_{i0j0}= 0$ (ô trống ở vị trí (i_0,j_0). Khi đó ứng với thao tác dịch ô tróng lên trên ta có thể viết: $B=dl(A)=(b_{ij})$,

ở đây:

$$b_{ij}=\begin{cases} a_{ij} & \text{nếu } (i, j) \text{ [redacted] } (i_0, j_0) \text{ và } (i, j) \text{ [redacted] } (i_0-1, j_0), \\ 0 & \text{nếu } (i, j)=(i_0-1, j_0) \text{ và } i_0>1 \\ a_{i0-1\ j0} & \text{nếu } (i, j)=(i_0,j_0) \text{ và } i_0>1 \end{cases}$$

Phát biểu lại bài toán P_1 (Bài toán P_2).

1. Tìm dãy trạng thái $s_1, s_2,...,s_n$ sao cho s_n[redacted]ĐÍCH và mỗi i thì s_i[redacted]$_1$ (hay là [redacted]*oán tử o: $o(s_i)=s_{i+1}$.*

2. Tìm dãy toán tử $o_1, o_2,..., o_{n-1}, o_n$ sao cho :

$$o_n(o_{n-1}(...o_1(s_0)...))=s_n \in ĐICH$$

hay tìm dãy sản xuất $p_1, p_2,..., p_n$ sao cho:

$$S_0 \xrightarrow{p_1} S_1 \xrightarrow{p_2} ... \xrightarrow{p_n} S_n \in ĐICH.$$

b. Biểu diễn vấn đề nhờ đồ thị

Đồ thị G là cặp G=(N,A), ở đây N là tập các nút (node), A là tập các cung. Với mỗi $n \in N$, ký hiệu:

$B(n) = \{m \in N \mid (n,m) \in A\}$.

$B^1(n) = B(n)$

$B^k(n) = B(B^{k-1}(n)) = \bigcup_{m \in B^{k-1}(n)} B(m)$

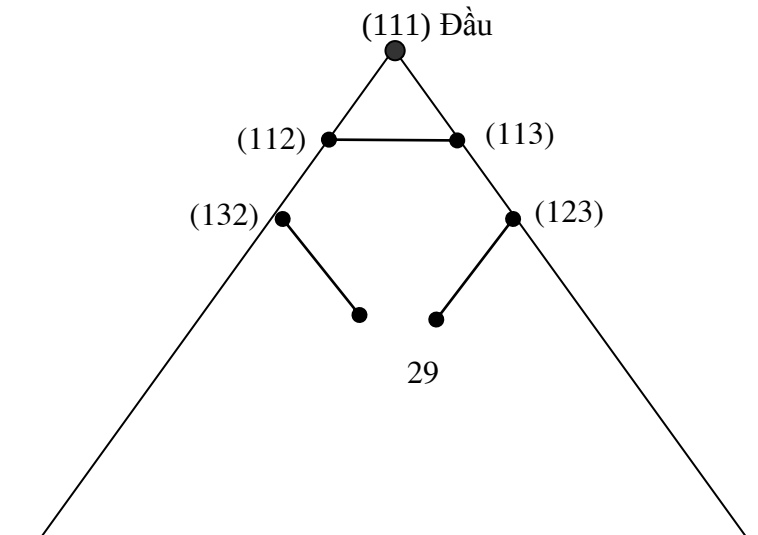
Phát biểu lại bài toán P_1 và P_2 (Bài toán P_3).

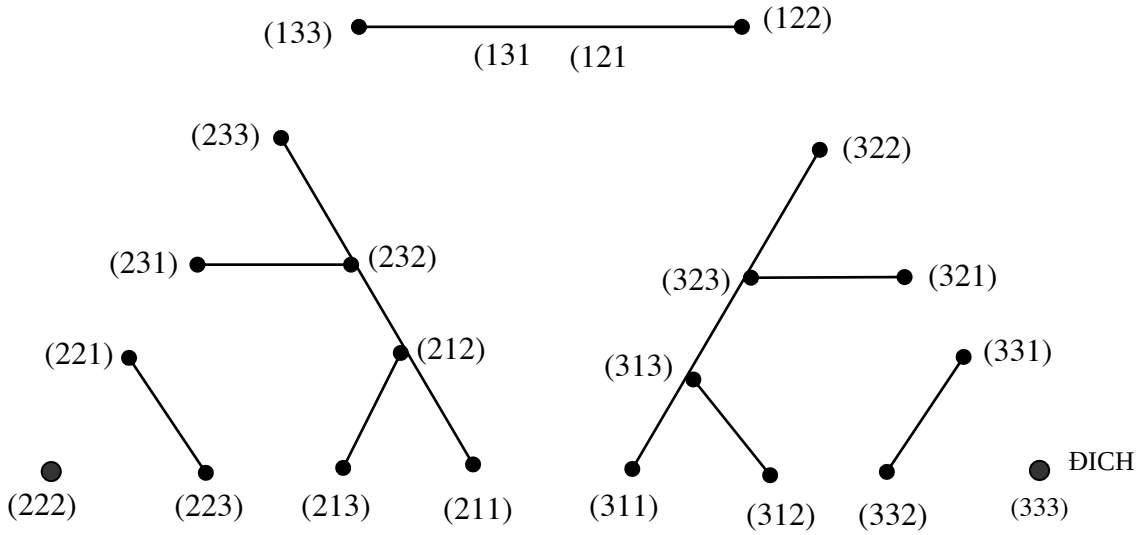
Cho đỉnh đầu s và tập đỉnh đích ĐICH. Hãy tìm một đường đi p (tối ưu) nào đó từ s tới một đỉnh $s_* \in ĐICH$.

Cách biểu diễn

Không gian trạng thái	***Đồ thị***
Trạng thái (đầu, đích) Toán tử (sản xuất) Dãy trạng thái liên tiếp Dãy toán tử Bài toán P_1, P_2	Nút (đầu, đích) Cung Đường đi Tìm đường đi trên đồ thị từ đỉnh đầu n_0 (tương ứng với s_0) tới đỉnh thuộc ĐICH.

Ví dụ 2.7. Không gian trạng thái của bài toán tháp Hà Nội với n=3 được thể hiện bằng hình vẽ sau:





Hình 2.9. Không gian trạng thái của bài toán Tháp Hà Nội

c. Các phương pháp tìm kiếm trong không gian trạng thái

Thực chất của việc giải quyết vấn đề trong không gian trạng thái là quá trình tìm kiếm đường đi từ trạng thái đầu tới các trạng thái đích sao cho thoả mãn một số điều kiện nào đó. Các phương pháp tìm kiếm trong không gian trạng thái điển hình mà ta sẽ xem xét gồm:

- Tìm kiếm theo chiều rộng (breath-first search)
- Tìm kiếm theo chiều sâu (depth-first search)
- Tìm kiếm sâu dần (depthwise search)
- Tìm kiếm cực tiểu hóa giá thành (cost minimization search)
- Tìm kiếm với tri thức bổ sung (heuristics search).

<1> Thủ tục tìm kiếm theo chiều rộng

Thủ tục TKR

Vào: Cây G=(N, A) với đỉnh gốc n_0.

Tập các đỉnh đích ĐICH [redacted]N

Ra: Một đường đi p: n_0[redacted]* [redacted]ICH.

Phương pháp: /* *Sử dụng hai danh sách kiểu FIFO là MO, ĐONG*/

{MO[redacted]$_0$}; /* *Cho n_0 vào **cuối** MO* */

While MO [redacted]o

{ n [redacted]et(MO) ; /* *lấy đỉnh n ở **đầu** danh sách MO* */

```
ĐONG [redacted]ONG [redacted]n};
if  B(n) [redacted]hen
   { MO [redacted]O [redacted](n) ; /* cho tập B(n) vào cuối danh sách MO*/
   If  B(n) [redacted]ICH [redacted]hen
      {  exit ("thành công");
         Xây dựng đường p;}}  }
Write ("không thành công");
}
```

Kết quả 2.1. Nếu trong cây G tồn tại ít nhất một đường đi từ n_0 tới tập ĐICH thì thủ tục TKR dừng và cho ra đường đi p có độ dài ngắn nhất. Nếu không tồn tại đường đi như vậy thì thuật toán dừng nếu và chỉ nếu đồ thị hữu hạn.

Ví dụ 2.8. Hãy chuyển trạng thái đầu (hình 2.10, a) về trạng thái đích (hình 2.10, b). Hình 2.11 là phần đồ thị đã duyệt cho đến khi tìm thấy trạng thái đích. Thứ tự các nút đã duyệt được chỉ ra bởi các số nằm bên cạnh các nút, còn đường đi tìm được tô nét kép.

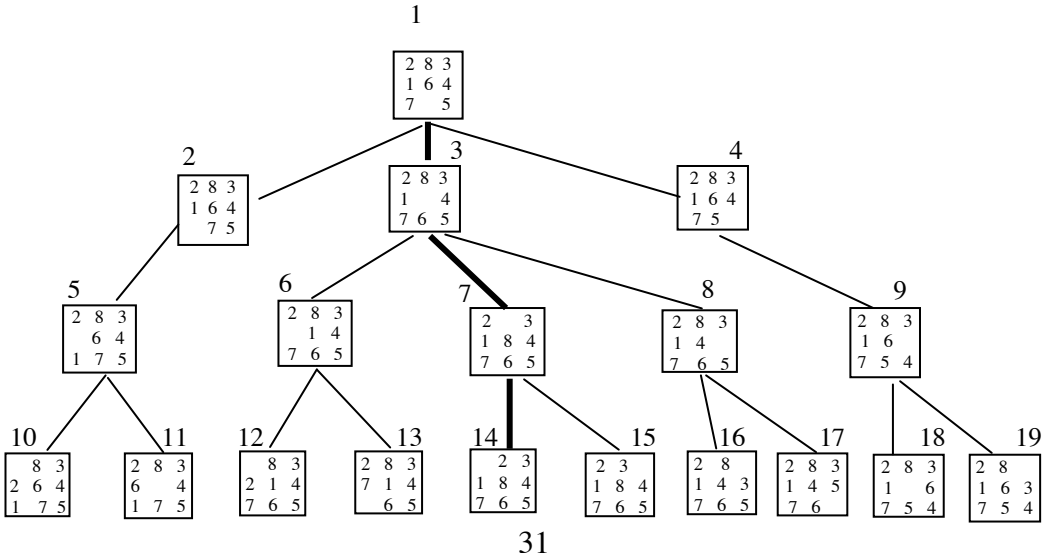
2	8	3
1	6	4
7		5

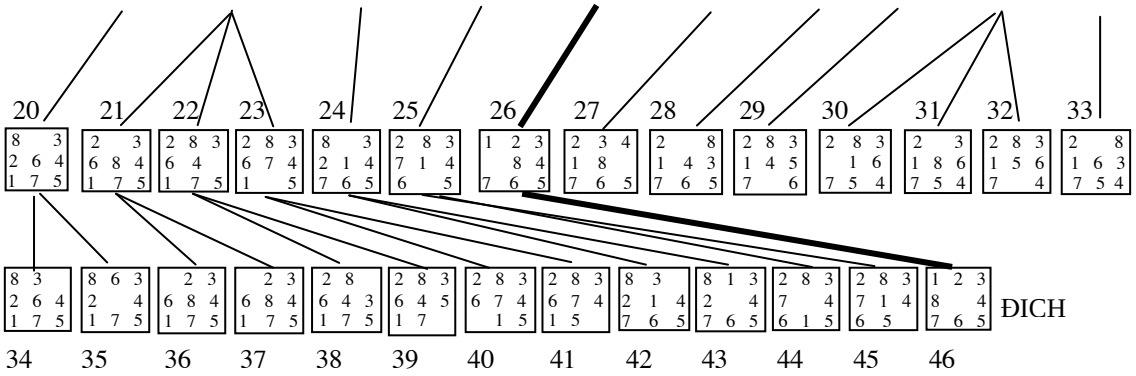
a)

1	2	3
8		4
7	6	5

b)

Hình 2.10. Trạng thái đầu và trạng thái đích của bài toán 8 số





Hình 2.11. Phần đồ thị trong trò chơi 8 số

<2> Thủ tục tìm kiếm theo chiều sâu

Thủ tục TKS

Vào: Cây G=(N,A) với đỉnh gốc n_0.

Tập các đỉnh ĐICH.

Ra: Một đường đi p : n_0 [redacted] ICH

Phương pháp: /* *Sử dụng dánh sách kiểu FIFO tên là ĐONG và* */

/* *danh sách kiểu LIFO tên là MO*/*

{MO [redacted] $_0$}; /* *Cho n_0 vào **cuối** MO* */

While MO [redacted] o

{n [redacted] et(MO) ; /* *lấy đỉnh n ở **đầu** danh sách MO* */

ĐONG [redacted] ONG [redacted] n};

if B(n) [redacted] hen

{ MO [redacted] O [redacted] (n) ; /* *cho tập B(n) vào **đầu** danh sách MO*/*

If B(n) [redacted] ICH [redacted] hen

{ exit (“thành công”);

Xây dựng đường p;

}

}

}

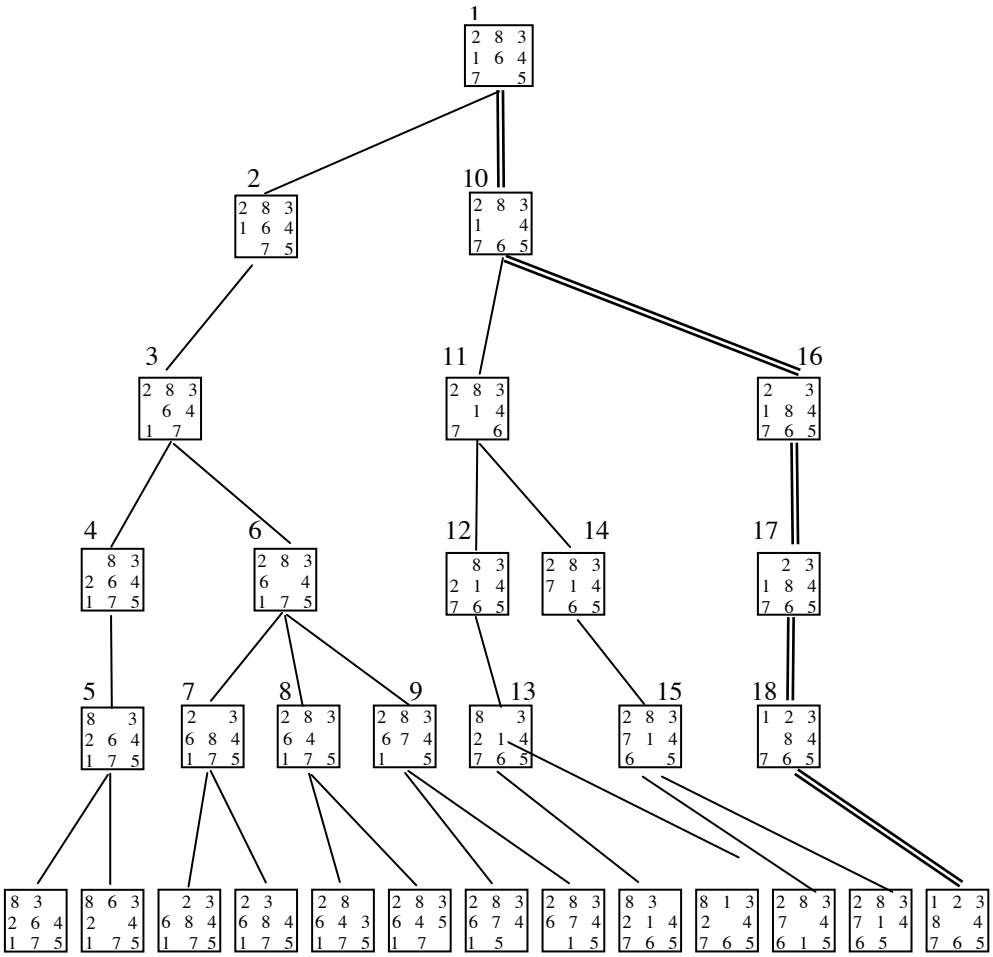
Write (“không thành công”);

}

Kết quả 2.2. Nếu cây G là hữu hạn và có đường đi từ n_0 tới tập ĐICH thì thuật toán TKS sẽ dừng và sẽ cho một đường đi từ nút đầu tới tập ĐICH.

Nhận xét: Đường đi nhận được theo thủ tục TKS có thể không ngắn nhất. Hơn nữa, nếu đồ thi vô hạn, thủ tục TKS có thể lặp vô hạn mặc dù trong đồ thị tồn tại đường đi từ n_0 tới tập ĐICH. Để khắc phục tình trạng này, ta xem xét thủ tục tìm kiếm sâu dần.

Ví dụ 2.9. Thủ tục duyệt cây theo chiều sâu trong đồ thị của bài toán ở ví dụ 2.8 được chỉ ra trên hình vẽ 2.12 sau:



Hình 2.12. Đồ thị biểu diễn trò chơi 8 số

<3> Thủ tục tìm kiếm sâu dần

Gọi d(n) là độ sâu của đỉnh n được xác định bởi.

$$d(n) = \begin{cases} \text{[illegible]} & \text{nếu } n = n_0 \\ \text{[illegible]}(m) & \text{nếu } n \text{ [illegible]}(m) \end{cases}$$

Ta đưa vào giá trị k gọi là mức sâu cho mỗi lần duyệt.

Thủ tục TKSD

Vào: Cây G=(N,A) với đỉnh đầu n_0, tập các đỉnh ĐICH.

Mức sâu k.

Ra: Một đường đi p : n_0 [illegible] CH

Phương pháp: /**Sử dụng danh sách kiểu FIFO: ĐONG và */*

/ và danh sách lai kiểu FIFO & LIFO MO*/*

{ MO [illegible] n_0 }; ĐS = k; /* *ĐS là giới hạn sâu hiện tại* */

While MO [illegible]o

{ n [illegible]et(MO); /* *Lấy đỉnh n ở đầu danh sách MO* */

ĐONG [illegible] ONG [illegible]n};

if B(n) [illegible]hen

{ if B(n) [illegible]ĐICH [illegible]en exit (“Thành công “);

Case d(n) do

[0 . . ĐS-1]: Đặt B(n) vào ***đầu*** danh sách MO;

ĐS : Đặt B(n) vào ***cuối*** danh sách MO;

ĐS+1 : { ĐS = ĐS + k;

if k=1 then đặt B(n) vào ***cuối*** danh sách MO

else đặt B(n) vào ***đầu*** danh sách MO;

} } }

Write(“không thành công”)

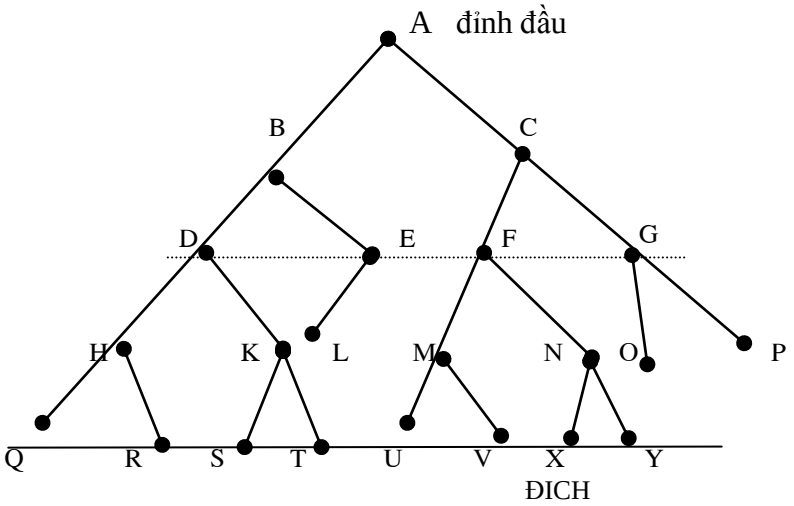
}

Nhận xét:

* Khi k =1 thủ tục TKSD trở thành TKR.

* Khi k [redacted] thủ tục TKSD tìm kiếm theo chiều sâu đối với các đỉnh có độ sâu nằm trong khoảng tk+1 và (t+1)k, t là số tự nhiên nào đó.

Kết quả 2.3. Nếu trong cây tồn tại ít nhất một đường đi từ n_0 đến ĐICH thì thủ tục TKSD sẽ dừng và cho đường đi có độ dài khác đường đi ngắn nhất không quá k-1. Nếu không tồn tại đường đi như vậy thì thủ tục TKSD dừng khi và chỉ khi đồ thị G hữu hạn.



Hình 2.13. Ví dụ đồ thị duyệt theo thứ tự sâu

Ví dụ 2.10. Xét thủ tục TKSD với đồ thị trên hình vẽ 2.17 khi k=2, thứ tự duyệt các đỉnh là : ABDECFGHQRKSTLMUVN.

<4> Thủ tục tìm kiếm đường đi với giá thành nhỏ nhất

Giả sử c: A[redacted] là hàm giá (cost) c:a[redacted]a)[redacted]$^+$. Một đường đi p trong G là dãy các đỉnh n_1, ..., n_k có giá $c(P)$ [redacted]$(n_i, n_i$[redacted]$)$.

Kí hiệu g(n) là giá cực tiểu của đường đi từ n_0 đến n [redacted]ĐICH. Khi đó, bài toán trên được phát biểu thành: *Tìm đường đi p_0 từ đỉnh gốc n_0 đến đỉnh n_k [redacted] ĐICH sao cho $g(n_k)=min\{g(n)|$ n [redacted]ĐICH}*.

Để tìm ra đường đi p_0 có giá $g(n_*)$ [redacted]in, ta sử dụng kí hiệu $g^0(n)$, xem như một ước lượng của g(n). Ta có thủ tục sau:

Thủ tục TKCT

Vào: Cây G=(N,A) với đỉnh gốc n_0.

Tập đỉnh đích ĐICH.

Hàm c: A[redacted]

Ra: Đường đi p: n_0 [redacted]ĐICH sao cho c(p)= $g(n_*)$ [redacted]n

Phương pháp: /* sử dụng hai danh sách MO, ĐONG*/

{MO[redacted]$_0$};

$g^0(n_0) = 0$;

While MO [redacted]o

{ n[redacted]et(MO); /* *Lấy n[redacted]O sao cho $g^0(n)$[redacted]n* */;

ĐONG [redacted]ONG [redacted]n};

if n [redacted]ĐICH then exit("thành công");

if B(n) [redacted]hen { MO[redacted](n) [redacted]MO ;

for each m[redacted]B(n) do $g^0(m)=g^0(n)+ c(n,m)$

}}

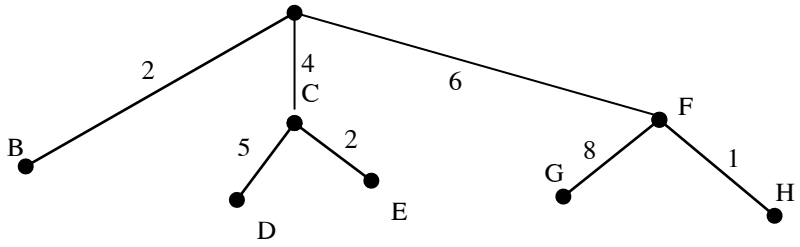
Write("không thành công") }

Nhận xét: Thủ tục TKR là trường hợp riêng của thủ tục TKCT khi c=1.

Kết quả 2.4: Nếu trong cây tồn tại đường đi p: n_0 [redacted]ĐICH thì thủ tục TKCT sẽ dừng và cho đường đi p^* sao cho $c(p^*) =g(n_k)=min\{g(n)|$ n [redacted]ĐICH}, n_k[redacted]ĐICH. Hơn nữa, thủ tục tối ưu theo nghĩa số đỉnh cho vào danh sách ĐONG là nhỏ nhất trong số các thủ tục tìm kiếm chỉ dựa trên giá các cung.

Ví dụ 2.11. Xét cây với giá các cung được cho như trên hình vẽ sau:

A



Với tập các đỉnh đích là {D, H}.

Khi đó thuật toán TKCT cho ra đường đi AFH.

Các Heuristics áp dụng cho thủ tục TKCT :

- Chỉ xét các đỉnh trong B(n) có triển vọng đạt tới tập ĐICH.
- Sắp các đỉnh trong MO trước mỗi lần xử lý nhờ các hàm đánh giá.

<5> Thuật toán tìm đường đi có giá nhỏ nhất với tri thức bổ sung

Đây là phương pháp tìm kiếm đường đi tối ưu dựa trên các thông tin đặc tả về bài toán. Các thông tin này dược gọi là các heuristic. Các kỹ thuật sử dụng heuristic được gọi là các mẹo giải. Các mẹo giải có thể là:

- Chọn toán tử xây dựng cung B sao cho có thể loại bớt những đỉnh không liên quan đến bài toán hoặc ít có triển vọng nằm trên đường đi tối ưu.
- Sử dụng thông tin bổ sung để xác định một hàm đánh giá cho phép lựa chọn đỉnh cần lấy từ tập MO.

Các phương pháp xây dựng hàm đánh giá:

+ Xác suất một đỉnh nào đó nằm trên đường đi tối ưu

+ Khoảng cách hay sự khác biệt giữa một đỉnh nào đó với tập các đỉnh đích.

Thủ tục TKCT*

Vào: Đồ thị G=(N,A) tuỳ ý với đỉnh gốc n_0.

Tập đỉnh đích ĐICH.

Hàm f^0: N[redacted].

Ra: Đường đi từ n_0 tới ĐICH.

Phương pháp:

{ MO[redacted]$_0$}; Tính $f^0(n_0)$;

While MO [redacted]o

{n [redacted]et(MO); /* *Lấy n [redacted]MO sao cho f^0 (n) [redacted]in* */

ĐONG [redacted]ONG [redacted]n};

if n[redacted]ĐICH then exit(" thành công");

if B(n) [redacted]hen

for each m [redacted]B(n) do

if m[redacted]ĐONG [redacted]MO then

{ MO[redacted]O [redacted]m};

Tính $f^0(m)$}

else if $f^0_{cũ}(m) > f^0_{mới}(m)$ then MO[redacted]O [redacted]m}};

Write(" không thành công") }

Trong thủ tục trên f^0 là hàm ước lượng heuristics nào đó, có thể dùng để sắp xếp các đỉnh trước khi xử lý. Các đỉnh trong danh sách MO được sắp theo thứ tự tăng dần của giá trị hàm f^0.

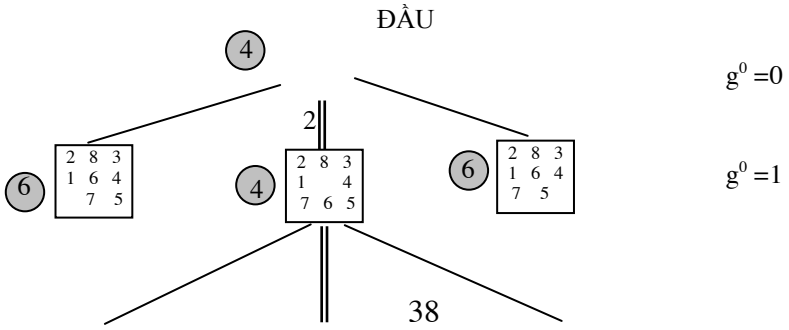
Nhận xét: Thuật giải TKCT* là tổng quát hơn TKCT khi xem f^0 [redacted]^0.

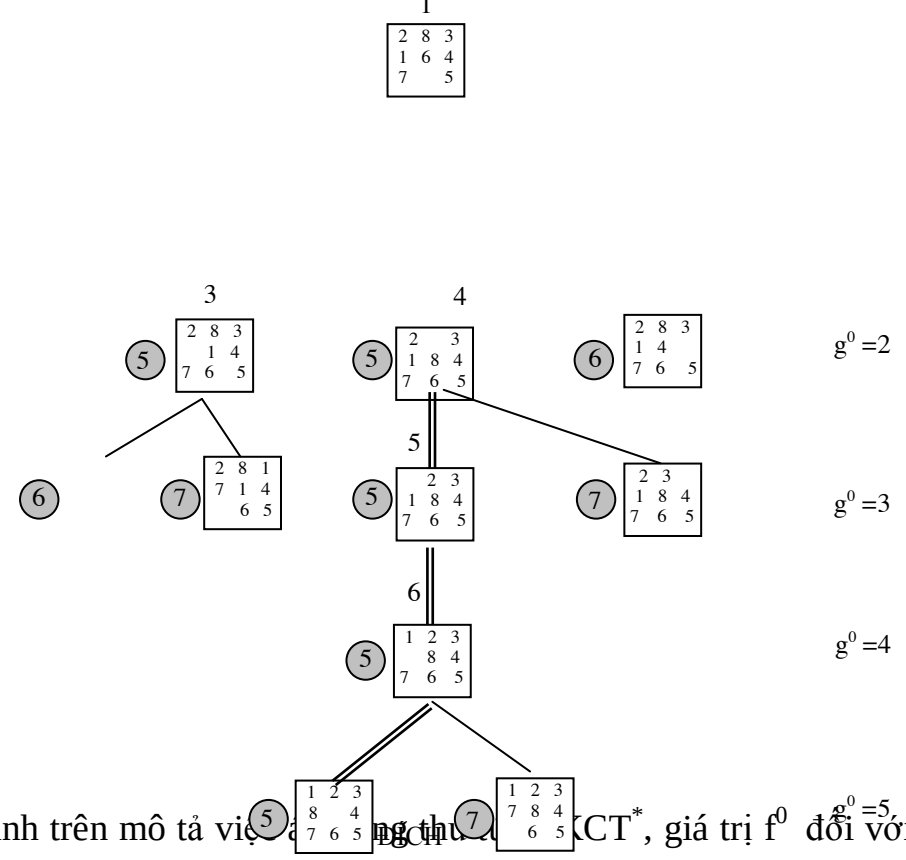
Ví dụ 2.12. Trở lại bài toán trò chơi 8 số (Bài toán 2.1).

Ta lấy hàm $f^0(n) = g^0(n)+w(n)$, ở đây $g^0(n)$ là độ dài đường đi hiện tại từ n_0 đến n, còn w(n) là số lượng các con số không nằm đúng vị trí của chúng ở hình trạng đích. Chẳng hạn, đối với đỉnh đầu n_0.

2	8	3
1	6	4
7		5

Ta có $g^0(n_0)=0$; $w(n_0)=4$. Do vậy $f^0(n_0) = 4$;



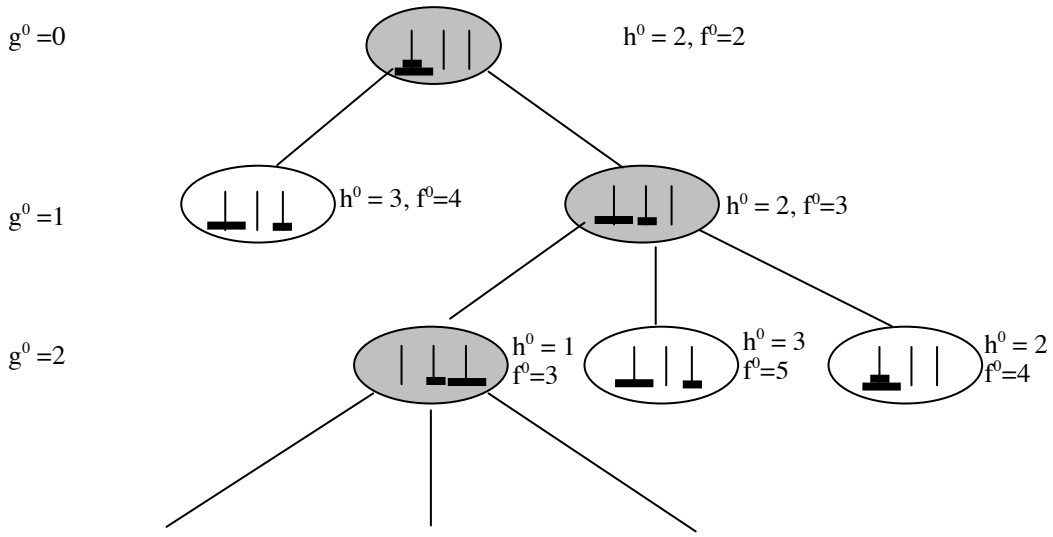


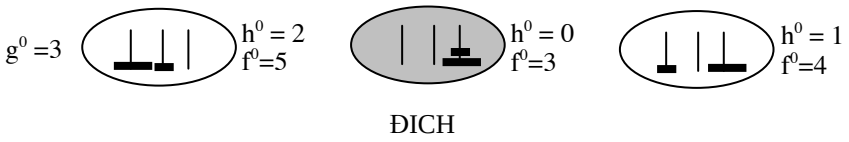
Hình trên mô tả việc áp dụng thử nghiệm KCT*, giá trị f^0 đối với mỗi nút chỉ ra trong hình tròn. Các số bên trên mỗi nút là thứ tự xét chúng trong quá trình tìm kiếm.

<u>Ví dụ 2.13:</u> Xét bài toán tháp Hà Nội với n=2, lấy hàm $f^0=g^0+h^0$, trong đó $h^0(n)$ là thông tin nói thêm về mối liên hệ giữa n và trạng thái đích. Chẳng hạn:

- $h^0=2$ nếu ở cọc C chưa có đĩa nào,
- $h^0=1$ nếu ở cọc C có đĩa to,
- $h^0=3$ nếu ở cọc C có đĩa nhỏ,
- $h^0=0$ nếu ở cọc C đã có hai đĩa.

Hình vẽ sau chỉ ra kết quả tìm kiếm.





Hình 2.16. Bài toán tháp Hà Nội với n=2.

Một cách tổng quát, có thể dùng thuật toán TKTC* để tìm ra đường đi p:n_0 CH sao cho c(p)=g(n_*), n_* ICH. Thật vậy, nếu xem f(n) là giá đường đi tối ưu từ n_0 CH và đi qua n thì có thể viết: $f^*(n)=g(n)+h(n)$, ở đây:

- g(n) : giá đường đi tối ưu từ n_0
- h(n): giá đường đi tối ưu từ n CH.

Đối với thủ tục TKCT* ta sử dụng ước lượng $f^0=g^0+h^0$, trong đó g^0 được tính như trong thủ tục TKCT, h^0 là một ước lượng của h.

Kết quả 2.5. (Tính đúng đắn và tính tối ưu của thuật toán).

Nếu đối với mỗi đỉnh n ta có 0 $h^0(n)$ h(n) thì thủ tục TKCT* dừng và cho đường đi p: n_0 ICH tối ưu khi đồ thị là cây. Trong trường hợp đồ thị tuỳ ý phải thêm điều kiện: Tồn tại 0 sao cho c(a)

Kết quả 2.6. Giả sử thủ tục TKCT* sử dụng hàm đánh giá $f^0_i(n)=g^0(n)+h^0_i$, i=1,2 và giả sử h_2 thoả mãn điều kiện $h^0_2(m) - h^0_2(n)$ h(m, n), (h(m,n) là độ dài đường đi ngắn nhất từ m đến n) và $h^0_1(n)$ $h^0_2(n)$ h(n) thì số nút đưa vào tập DONG của thuật toán $TCTK_2$* bao giờ cũng nhỏ hơn số nút đối với thuật toán $TCTK_1$*.

2.6.2 Quy bài toán về các bài toán con và các chiến lược tìm kiếm trên đồ thị VÀ/HOẶC

a. Quy bài toán về các bài toán con

Quy bài toán về các bài toán con là việc tách bài toán (vấn đề) ban đầu thành các bài toán (vấn đề) sơ cấp.

Ví dụ 2.14. Với bài toán 2.1 (trò chơi n^2 -1 số), các bài toán sơ cấp tương ứng với trường hợp chỉ cần một bước chuyển từ trạng thái đầu sang trạng thái đích.

Ví dụ 2.15. Xét bài toán tìm cách đi từ Nhà hát lớn đến ga Hà Nội.

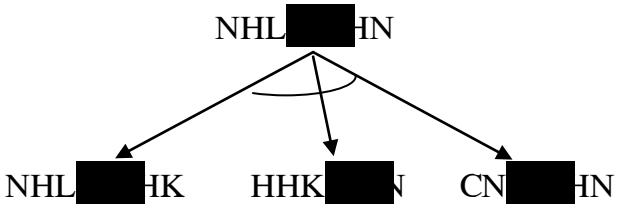
Bài toán này có thể chia nhỏ thành ba bài toán con:

Bài toán 1: Đi từ Nhà hát lớn tới Hồ Hoàn Kiếm.

Bài toán 2: Đi từ Hồ Hoàn Kiếm đến Cửa nam.

Bài toán 3: Đi từ Cửa nam tới ga Hà Nội.

Lời giải bài toán ban đầu nhận được trên cơ sở lời giải của cả ba bài toán con này. Ta có hình vẽ



Ví dụ 2.16. (Bài toán tháp Hà Nội) Xét trường hợp n=3, ta có kết quả như trên hình 2.19.

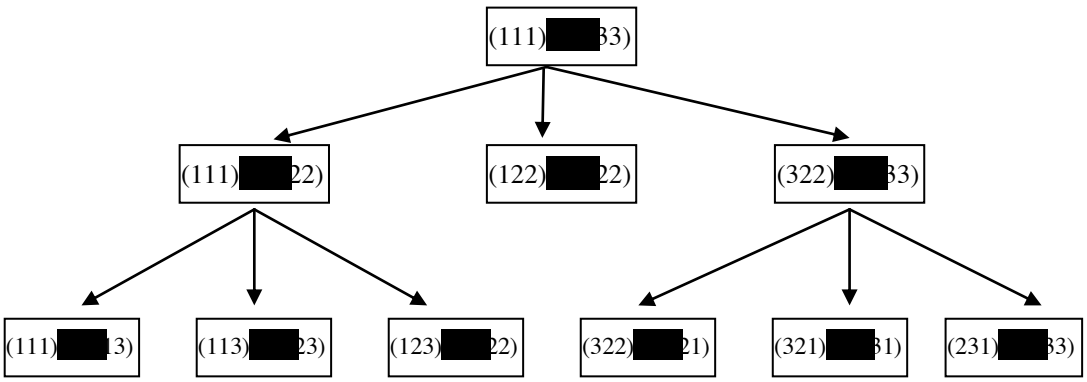
Bài toán ban đầu (111) 33) được quy về 3 bài toán con:

Bài toán 1. (111) 22) chuyển 2 đĩa A, B từ cọc 1 sang cọc 2;

Bài toán 2. (122) 22) chuyển đĩa C từ cọc 1 sang cọc 3;

Bài toán 3. (322) 33) chuyển 2 đĩa A,B từ cọc 2 sang cọc 3.

Bài toán 2 là bài toán sơ cấp (có thể giải được ngay!), còn các bài toán lại được phân rã tiếp tục



Hình vẽ 2.17. Phân rã bài toán tháp Hà Nội n=3

Ta nhận được lời giải của bài toán ban đầu như sau:

(111)[redacted]13)[redacted]23)[redacted]22)[redacted]22)[redacted]21)[redacted]11)[redacted]33)

b. Đồ thị VÀ/HOẶC

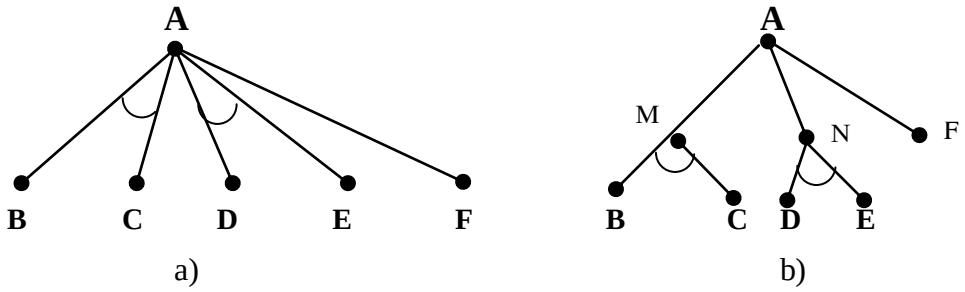
Đồ thị (có hướng) VÀ/HOẶC là cặp G=(N, A), sao cho với mọi n [redacted]N tất cả các đỉnh m[redacted](n) cùng thuộc vào một trong hai kiểu là đỉnh VÀ hay đỉnh HOẶC. Nếu các đỉnh con m của n là đỉnh VÀ thì các cung (n, m) m [redacted]3(n) được nối với nhau bởi ngoặc lớn.

Đỉnh con m[redacted]3(n) là đỉnh VÀ nếu bài toán ứng với đỉnh n không thể giải thông qua một mình bài toán ứng với đỉnh m. Đỉnh con m[redacted]3(n) là đỉnh HOẶC nếu bài toán ứng với đỉnh n có thể giải thông qua một mình bài toán ứng với đỉnh m

Ví dụ 2.17: Giả sử bài toán A có thể giải quyết theo ba cách sau:

- Hoặc thông qua giải quyết đồng thời hai bài toán B và C.
- Hoặc giải quyết các bài toán D và E.
- Hoặc giải bài toán F (phát biểu tại A).

Ta có thể biểu thị mối liên hệ đó nhờ đồ thị (hình 2.20, a):



Hình 2.18. Đồ thị VÀ/ HOẶC

Tuy nhiên đồ thị trên hình 2.15, a) chưa phải đồ thị VÀ/HOẶC, vì các đỉnh con của A có cả đỉnh VÀ và đỉnh HOẶC, do vậy ta chuyển về đồ thị VÀ/HOẶC, được biểu diễn bởi hình 2.15, b). Khi đó

- Các đỉnh B, C, D, E là đỉnh VÀ.

- Các đỉnh N, M, F là đỉnh HOẶC.

Trong đồ thị VÀ/HOẶC các đỉnh tương ứng với các bài toán sơ cấp gọi là đỉnh kết thúc.

Sau đây ta sẽ xem xét hai khái niệm liên qua tới đồ thị VÀ/HOẶC là: Đỉnh giải được và đỉnh không giải được.

* Định nghĩa đỉnh giải đượ:

- Các đỉnh kết thúc là đỉnh giải được,

- Nếu đỉnh n có các đỉnh con là đỉnh HOẶC thì nó là đỉnh giải được [redacted]n tại một đỉnh con của nó giải được,

- Nếu một đỉnh có các đỉnh con là đỉnh VÀ thì nó là đỉnh giải được [redacted] cả các đỉnh con của nó là giải được.

* Đồ thị lời giải là đồ thị con của đồ thị VÀ/ HOẶC chỉ bao gồm các đỉnh giải được và chứa đỉnh đầu.

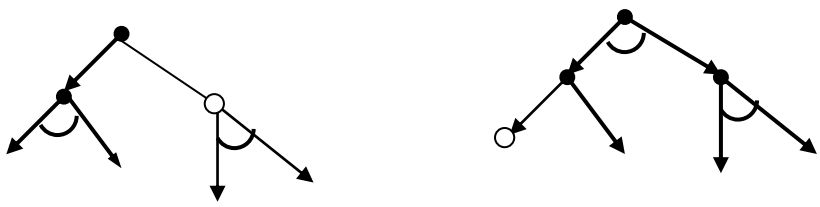
* Định nghĩa đỉnh không giải được:

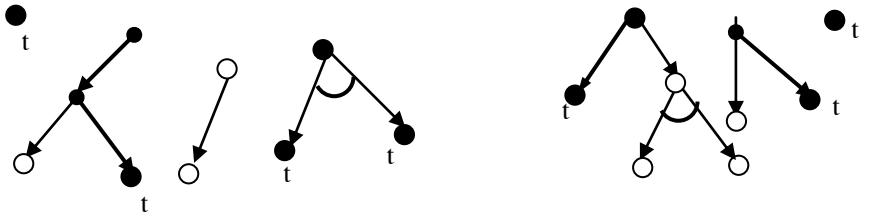
- Các đỉnh không là kết thúc và không có đỉnh con là đỉnh không giải được,

- Nếu đỉnh không là đỉnh kết thúc và có các đỉnh con là đỉnh VÀ thì nó là không giải được [redacted]ồn tại một đỉnh con của nó không giải được,

- Nếu đỉnh không là đỉnh kết thúc và có các đỉnh con là đỉnh HOẶC thì nó là không giải được [redacted]ọi đỉnh con của nó là không giải được.

Ví dụ 2.18: Hình vẽ dưới đây là đồ thị VÀ/HOẶC trong đó các đỉnh kết thúc được kí hiệu bởi chữ t, các đỉnh giải được tương ứng với các hình tròn tô đậm, các cung trong đồ thị lời giải cũng được tô đậm.



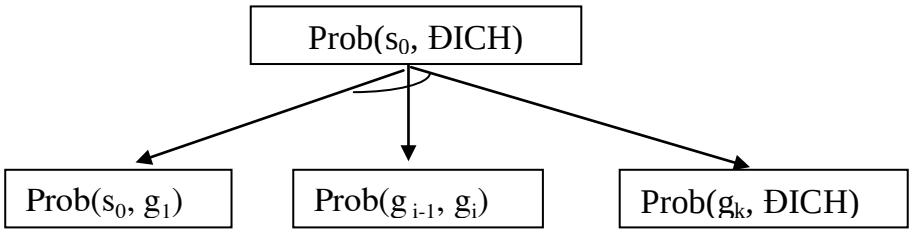


Sự tương ứng giữa quá trình quy bài toán về các bài toán con với đồ thị VÀ/HOẶC được thể hiện trong bảng sau:

Quy bài toán về các bài toán con	***Đồ thị VÀ/HOẶC***
Bài toán	Đỉnh
Toán tử quy bài toán về bài toán con	Cung
Bài toán ban đầu	Đỉnh đầu (đỉnh gốc)
Các bài toán sơ cấp	Đỉnh cuối, đỉnh kết thúc
Các bài toán con phụ thuộc	Đỉnh VÀ
Các bài toán con độc lập	Đỉnh HOẶC
Giải bài toán ban đầu.	Tìm đồ thị con lời giải.

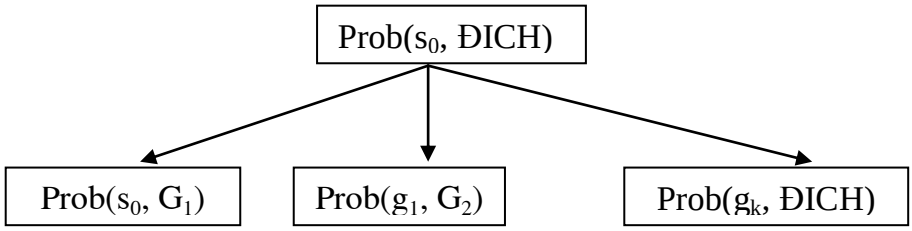
c. Giải bài toán biểu diễn trong không gian trạng thái nhờ phương pháp phân rã

Bài toán P_1 (P_2) có thể đưa về dạng Prob(s_0, ĐICH). Thực chất quá trình giải là tìm dãy trạng thái s_0,..., s_n ĐICH sao cho s_i có thể biến đổi trực tiếp thành s_{i+1}. Do vậy, một cách tiếp cận là xác định trạng thái trung gian cơ bản g_1, ..., g_k trong dãy s_0,..., s_n và bài toán Prob(s_0, ĐICH) được quy về các bài toán con:



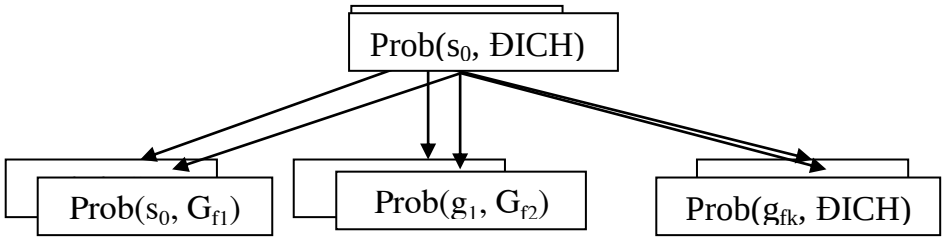
Song trên thực tế việc tìm ra ngay dãy $g_1,..., g_k$ là rất khó. Có thể làm như sau:

Xác định tập G_1 các trạng thái có khả năng là điểm khớp đầu tiên. Giải bài toán Prob(s_0, G_1) để tìm ra g_1[redacted]$_1$, sau đó xác lập G_2 và giải bài toán Prob(g_1, G_2) để tìm ra g_2[redacted]$_2$, ... Cuối cùng giải bài toán Prob(g_k, ĐICH)



Có hai cách tiếp cận cơ bản:

Phương pháp dựa vào toán tử khoá: Thực chất của bài toán P_1 là xác định dãy toán tử $o_1,..., o_n$ sao cho $o_n(o_{n-1}(...o_1(s_0)...))$[redacted]ICH. Trên thực tế ta mong muốn xác định một tập con các toán tử quan trọng (khoá) $f_1,..,f_k$. Gọi G_{fi} là tập trạng thái có thể áp dụng f_i. Khi đó, giải bài toán Prob(g_{f1}, G_{f1}) sẽ cho trạng thái g_{f2}[redacted]$_{f2}$...



Phương pháp xác định sự khác biệt f: So sánh sự khác biệt giữa s_0 và tập ĐICH, tìm ra toán tử f_1 quan trọng nhất cho phép giảm bớt sự khác biệt lớn nhất...

d. Các phương pháp tìm kiếm trong đồ thị VÀ/HOẶC

Trong các thuật toán tìm kiếm chúng ta phải dùng đến hai thủ tục sau:

- Thủ tục gán nhãn giải được cho các đỉnh GD(n[redacted])
- Thủ tục gán nhãn không giải được cho các đỉnh KGD(n[redacted]).

Với mỗi đỉnh n [redacted], ta dùng các ký pháp:

+ Nếu n là đỉnh kết thúc thì kt(n)=True, ngược lại kt(n)=False.

+ Nhan(n) = { "gđ" nếu n là đỉnh giải được
"kgđ" nếu n là đỉnh không giải được
"kxđ" nếu n không xác định, không đủ thông tin hoặc chưa được xét tới.

+ kieu(n)=true nếu các đỉnh con của n là đỉnh VÀ

Procedure GĐ (n[redacted])

{ if nhan(n)= "kxđ" then

if kt(n) then nhan(n)="gđ"

else if n [redacted]O [redacted]ONG then nhan(n)="kgđ"

else if kieu(n) then {bien=True;

While B(n) [redacted]nd bien do

{m [redacted]et(B(n));

gd(m);

bien=(nhan(m)="gđ")}

if bien then nhan(n)="gđ" else nhan(n)="kgđ"}

else {bien=false;

Repeat

{m[redacted]et(B(n));

gđ(m);

bien=(nhan(m)="gđ")}

Until bien or B(n)=[redacted]

if bien then nhan(n)="gđ" else nhan(n)="kgđ"}

**** Thủ tục tìm kiếm theo chiều rộng***

Thủ tục TKRM

Vào: Cây VA/HOAC G=(N, A) với đỉnh đầu n_0.

Ra: Thông báo "không thành công" nếu n_0 không giải được, "thành công" nếu n_0 giải được và đưa ra cây lời giải.

Phương pháp: /* sử dụng hai danh sách FIFO: MO, ĐONG */

{MO ={n_0};

While MO [redacted]o

{n[redacted]t(MO);

ĐONG[redacted]} [redacted]ONG;

if B(n) [redacted]en

{MO[redacted]O [redacted](n); /* đưa B(n) vào ***cuối*** danh sách MO */

bool = false;

For each m [redacted](n) do

if kt(m) then {nhan="gđ"; bool=true}

if bool then

{gđ(n_0);

if nhan(n_0)="gđ" then exit("thành công"

else loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được

/* *nếu m [redacted]B(n) thì n được gọi là tổ tiên của m* */

}}

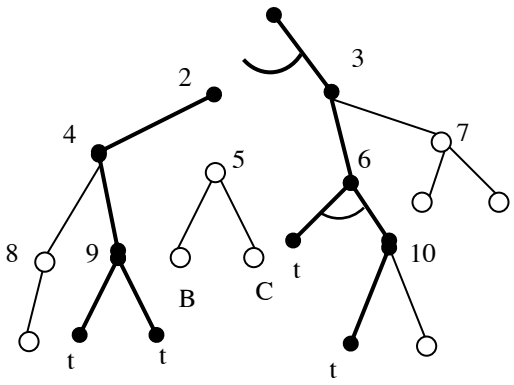
else {nhan(n)="kgđ"; kgđ(n_0);

if nhan(n_0) = "kgđ" then exit ("không thành công"

else loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được;}}}

Kết quả 2.7. Nếu cây lời giải tồn tại thì thủ tục TKMR sẽ dừng và đưa ra cây lời giải có độ cao nhỏ nhất.

Ví dụ 2.19. Xét cây VA/ HOAC sau đây (Hình 2.17), trên đó thứ tự đưa các nút vào danh sách ĐONG được chỉ ra bởi các số ở bên cạnh các đỉnh. Các đỉnh tô đậm là các đỉnh giải được. Cây lời giải được xác định qua các cung tô đậm.



Hình 2.20. Đồ thị VÀ/ HOẶC

**** Thủ tục tìm kiếm theo chiều sâu***

Thủ tục TKSM

Vào: Cây VA/HOAC G=(N,A) với đỉnh đầu n_0.

Giới hạn sâu D.

Ra: Cây lời giải nếu tồn tại.

Phương pháp: /* Sử dụng danh sách FIFO tên là ĐONG, và danh sách LIFO tên là MO */

{MO = {n_0};

While MO [redacted]o

{n [redacted]et(MO); /*Lấy n là đỉnh ***đầu*** của MO */

ĐONG[redacted]ONG [redacted]n};

if d(n) [redacted]D and B(n) [redacted]hen

{MO[redacted]O [redacted]B(n); /* Đưa B(n) vào ***đầu*** danh sách MO */

bool = false;

For each m [redacted](n) do

{if kt(m) then {nhan="gđ"; bool=true}}

if bool then

{gđ(n_0);

if nhan(n_0)="gđ" then exit("thành công")

else loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được}}

else {nhan(n)=”kgđ”; kgđ(n_0);

if nhan(n_0) = “kgđ” then exit (“không thành công”)

else loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được;}}}

Nhận xét: Cũng như ở trên, nếu tất cả các cây lời giải có độ cao lớn hơn D thì chúng sẽ bị bỏ qua trong quá trình tìm kiếm. Do vậy thực tế tồn tại cây lời giải song thuật toán lại không tìm được.

Có thể sửa đổi thủ tục TKSM thành thủ tục duyệt sâu dần áp dụng cho cây VA/HOAC.

**** Thủ tục tìm cây lời giải cực tiểu hoá giá thành.***

Định nghĩa h(n) giá tối ưu của cây lời giải gốc n:

- Nếu n là đỉnh kết thúc h(n)=0.
- Nếu n không là đỉnh kết thúc và các con $n_1,..,n_k$ là đỉnh HOAC thì h(n)=min(c(n,n_i)+h(n_i)).

Nếu n không là đỉnh kết thúc và các con $n_1,..., n_k$ là đỉnh VA thì:

- h(n)= $\sum_i$ (c(n, n_i) + h(n_i)) đối với giá tổng cộng.
- h(n)= $\max_i$ (c(n,n_i)+h(n_i)) đối với giá cực đại.
- h(n) không xác định nếu n là đỉnh không giải được.

Mục đích tìm kiếm:

- Cây lời giải được xây dựng dần trong quá trình mở rộng cây lựa chọn.
- Đối với đỉnh n ta sử dụng $h^*(n)$ là ước lượng của h(n).
- Các đỉnh lá của cây lựa chọn thuộc một trong ba dạng:
  + Các đỉnh kết thúc.
  + Các đỉnh không kết thúc
  + Các đỉnh chưa được xét đến.

Các đỉnh này được gọi là các mút.

Ta xây dựng ước lượng h^* đối với h như sau:

n là đỉnh mút.

- Nếu n là đỉnh kết thúc $h^*(n)=0$.
- Nếu n không là đỉnh kết thúc và không có con thì $h^*(n)$ không xác định.
- Nếu n chưa được xét thì $h^*(n)$ có thể là một ước lượng Heuristics nào đó của h(n).

n không là mút:

- n có các đỉnh con $n_1,...,n_k$ là đỉnh HOAC

  $h^*(n)=\min(c(n,n_i)+h^*(n_i))$.
- n có các đỉnh con $n_1,..., n_k$ là đỉnh VA

  $h^*(n)=$ [redacted]$(n,n_i)+h^*(n))$ - Đối với giá tổng cộng,

  $h^*(n)=\max(c(n,n_i)+h^*(n_i))$ - Đối với giá cực đại.

Ta nhận thấy là trong quá trình tìm kiếm, ở mỗi bước có thể chứa môt tập các cây con có gốc n_0 sao cho chúng có thể trở thành phần trên của cây lời giải cuối cùng. Ta gọi các cây này là cây lời giải tiềm tàng gốc n_0.

Từ cây lựa chọn ta xác định cây lời giải tiềm tàng T_0 gốc n_0 có nhiều khả năng là phần trên của cây lời giải như sau:

- Đỉnh đầu n_0[redacted]$_0$.
- Nếu n_0[redacted]$_0$ có các đỉnh con $n_1,..., n_k$ là:

  a/ Đỉnh HOAC thì chọn đỉnh n_i sao cho $c(n,n_i)+h(n_i)$[redacted]n.

  b/ Đỉnh VA thì chọn đỉnh $n_1,.., n_k$ thuộc vào T_0.

Thủ tục TKCTM

Vào: Cây VA/ HOAC G=(N,A) với đỉnh gốc n_0.

Ra: Cây lời giải tối ưu.

Phương pháp

{MO[redacted]$_0$}; T_0[redacted]$_0$};

While MO [redacted]o

{Xây dựng cây T_0;

n[redacted]et(MO) [redacted]Mut(T_0); /*Mut(T_0) là tập các lá của T_0 */

ĐONG[redacted]ONG [redacted]n};

if kt(n) then { nhan(n)= "gđ"; get(n_0);

if nhan(n_0)=”gđ” then exit(“thành công”);

else loại khỏi MO các đỉnh có tổ tiên là giải được;}

else if B(n) [redacted]en {MO[redacted]O [redacted](n);

for each m [redacted](n) do h^*(m);

for each m[redacted]O[redacted]ONG do h^*(m) }

else {nhan(n)= “kgđ”; kgđ(n_0);

if nhan(n_0) =”kgđ” then exit (“không thành công “)

else Loại khỏi MO các đỉnh có tổ tiên là không giải được}}}

Nhận xét. Nếu cây VA/ HOAC chỉ chứa đỉnh HOẶC thì thủ tục này chính là thủ tục TKCT, nếu c[redacted] và h^*[redacted] đối với mọi đỉnh mút và sử dụng giá cực đại thì thủ tục trở thành TKRM.

Kết quả 2.8. Nếu h^*(n)[redacted](n) đối với mọi đỉnh nằm trong MO và [redacted] c(a)[redacted]hì thủ tục TKCTM sẽ dừng và cho kết quả là cây lời giải tối ưu.

2.6.3 Biểu diễn vấn đề nhờ logic hình thức và các phương pháp chứng minh

a. Logic mệnh đề

Một mệnh đề p là một phát biểu chỉ nhận giá trị đúng (T, True, 1) hoặc sai (F, False, 0).

Ví dụ 2.20.

+ 5 nhân 2 hai là 10.

+ Hà Nội là thủ đô của nước Việt Nam.

+ Bạn A học giỏi hơn bạn B.

Các biểu thức logic mệnh đề được xây dựng trên cơ sở các tên mệnh đề và các phép toán logic theo quy tắc cú pháp nhất định.

Tên mệnh đề: thường được ký hiệu bởi các chữ cái la tinh thường a, b, p, q...

Các phép toán logic bao gồm:

- Hội ([redacted]and, và).
- Tuyển ([redacted]or, hoặc).

- Phủ định (not, không).
- Kéo theo (
- Tương đương (

Thứ tự ưu tiên các phép toán: phủ định, kéo theo, tương đương, hội, tuyển.

Giá trị chân lý của một biểu thức được tính dựa theo bảng chân lý (truth table) sau:

a	b	a	a		a	a
0	0	0	0	1	1	1
0	1	0	1	1	1	0
1	0	0	1	0	0	0
1	1	1	1	0	1	1

Khẳng định: i) Mọi biểu thức logic mệnh đề đều có thể đưa về biểu thức tương đương chỉ chứa các phép à

ii) Các phép và ó tính chất giao hoán, kết hợp, phân phối và luỹ đẳng.

Các phép biến đổi tương đương:

R_1. A , A

R_2. A

R_3. A A

R_4. A B) A B) C, A B C) A B) C

R_5. A), A B)

R_6. A B

R_7. A B C) A B) A C)

A B C) A B) A C)

R_8. A A B)

A A B)

R_9.

R_{10}. A) , A)

R_{11}. [redacted]

R_{12}. A [redacted] A [redacted]

A [redacted] A [redacted]

R_{13}. [redacted]

Cách tính giá trị của một biểu thức logic mệnh đề:

+ Lập bảng giá trị chân lý. Nếu biểu thức có n biến mệnh đề, ta lập bảng với 2^n dòng.

+ Biến đổi biểu thức về biểu thức tương đương và dựa và bảng giá trị chân lý của các phép toán.

b. Logic vị từ

Vị từ p(x,...,y) là một phát biểu chứa các biến x,...,y sao cho khi x, y nhận giá trị cụ thể thì p(x,...,y) nhận giá trị đúng hoặc sai.

Ví dụ 2.21. p(x,y,z) có nghĩa là x.y=z. Khi đó tính chất giao hoán của phép nhân x.y=y.x được diễn tả dưới dạng: [redacted]y p(x,y,z) [redacted]y,x,z).

Các phép toán: [redacted]

Các lượng từ:

Lượng từ tồn tại ([redacted]: Khi viết [redacted] p(x, y, . . ., z) có nghĩa là tồn tại một giá trị x_0 của x sao cho p(x_0, y, . . .,z) đúng với mọi giá trị y, . . ., z.

Lượng từ mọi ([redacted] Khi viết [redacted]p(x, y, . . ., z) có nghĩa là đối với mọi giá trị của x ta có p(x, y, . . .,z) đúng với mọi giá trị y, . . ., z.

Logic vị từ cho phép diễn đạt hầu hết các khái niệm và nguyên lý của khoa học cơ bản, đặc biệt là toán học và vật lý.

Hai phạm vi ứng dụng nỗi bật của logic hình thức là:

+ Chứng minh định lý tự động

+ Giải quyết vấn đề.

c. Chứng minh định lý nhờ logic hình thức

Bài toán: *Cho các giả thiết dưới dạng các biểu thức mệnh đề (hoặc logic vị từ) GT_1, ...,GT_m. Hãy rút ra một trong các kết luận KL_1,..., KL_n.*

<1> Thuật giải của Wong H (dùng cho cả logic mệnh đề và logic vị từ).

Vào: Các biểu thức mệnh đề $GT_1,...,GT_m$.

Các kết luận $KL_1,...,KL_n$.

Ra: Thông báo "Thành công' nếu GT_1 [redacted] T_m [redacted] L_1 [redacted] L_n.

Phương pháp:

{ For i=1 to m do { Trans(GT_i); VT [redacted] T_i [redacted] T}

for i=1 to n do {Trans (KL_i); VP [redacted] L_i [redacted] P}

/* VT, VP là danh sách các biểu thức ở vế trái, vế phải dấu [redacted]

P[redacted] VT,VP)}; /* P là danh sách các kết quả cần chứng minh VT[redacted] P */

While P [redacted] o { (VT, VP) [redacted] et(P);

if VT [redacted] P=[redacted] hen {chuyển(VT, VP);

if VP [redacted] T =[redacted] hen

if not tach (VT,VP) then Exit "Không thành công"}}

Write("Thành công") }

Ở đây:

[redacted] Thủ tục Trans(BT): Đưa biểu thức BT về dạng chuẩn, nghĩa là biểu thức chỉ chứa các phép toán: [redacted] và thuộc một trong hai dạng sau:

+ [redacted]

+ [redacted] trong đó $l_{ij} = p_{ij}$ hoặc $l_{ij} = $ [redacted] và p_{ij} là các mệnh đề đơn.

Ví dụ: Với biểu thức BT=[redacted]), thủ tục Trans(BT) sẽ cho ta biểu thức (a[redacted]) [redacted]c[redacted]) hoặc (a[redacted]) [redacted]) [redacted]) [redacted]).

[redacted] Thủ tục chuyen(VT, VP): Chuyển vế các GT_i, KL_j có dạng phủ định, thay dấu [redacted]ên trong GT_i và dấu [redacted]ên trong KL_j bởi dấu phẩy.

Ví dụ: Với (VT,VP) là [redacted], p [redacted] q. Khi đó thủ tục chuyen(VT, VP) cho ta: [redacted], p, e [redacted].

[redacted] Hàm tách(VT, VP): Tách thành hai danh sách con nếu có dấu [redacted]rong một GT_i hoặc dấu [redacted]rong một KL_j nào đó. Nếu tách được, hàm cho giá trị True, ngược lại cho giá trị False.

Ví dụ: Với (VT, VP) là [redacted], p, e [redacted] hàm tách sẽ tách thành hai danh sách con:

+ [redacted] p, e [redacted]

+ q, p, e [redacted]

và hàm cho giá trị True.

Kết quả: Thuật toán Wong H. dừng sau hữu hạn bước và cho thông báo "Thành công" nếu và chỉ nếu từ $GT_1,...,GT_m$ có thể suy ra một trong các kết luận $KL_1,...,KL_n$.

Nhận xét: - Nếu số phép toán liên kết [redacted]rong GT_i và [redacted]rong KL_j là m thì thuật giải Wong H sẽ sinh ra từ m đến 2^m dòng (VT, VP) [redacted].

Ví dụ 2.22. Chứng minh rằng từ (a[redacted]) [redacted] (b[redacted]) [redacted] a,b suy ra d.

Đưa các GT_i và KL_j về dạng chuẩn và xây dựng P=(VT, VP) ta có

[redacted], a, b [redacted]

Do VT [redacted]P = [redacted]à thủ tục chuyển không thực hiện được nên áp dụng thủ tục Tach(VT, VP) ta thu được 2 dòng:

1. (VT_1, VP) = [redacted], a, b [redacted]
2. (VT_2, VP) = [redacted], a, b [redacted]

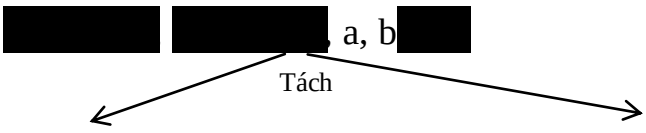
Áp dụng thủ tục chuyen(VT_1, VP) ta thu được

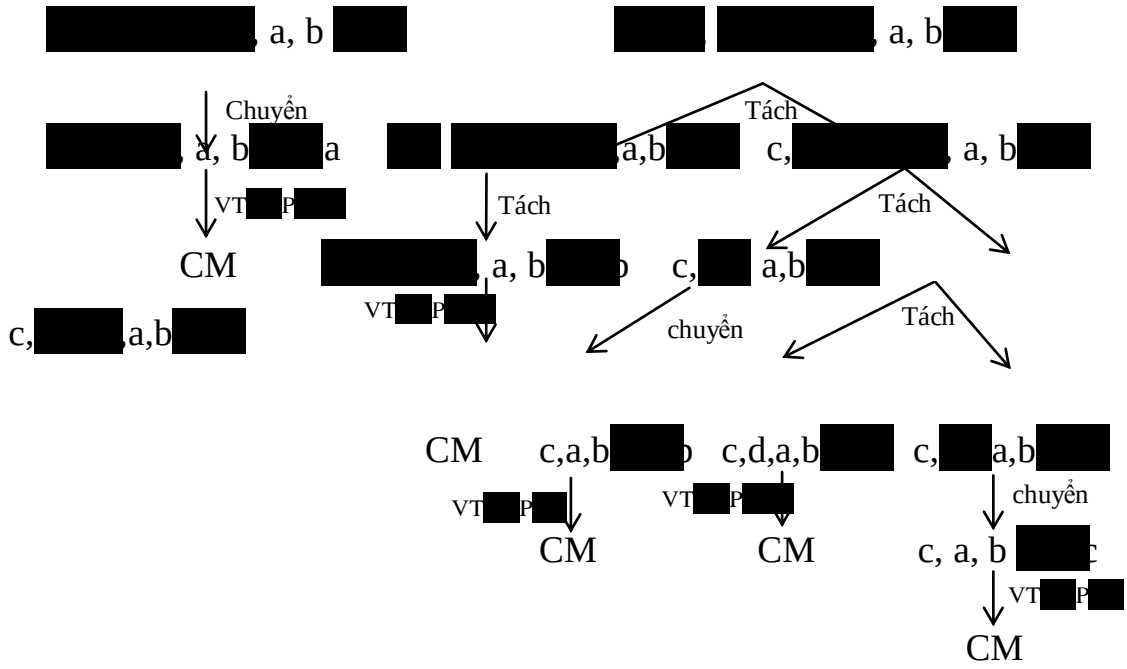
(VT_1, VP) = [redacted], a, b [redacted] a.

Vì VT_1 [redacted]P = [redacted]ên VT_1[redacted]P.

Đối với (VT_2, VP) không chuyển được nên ta tiếp tục tách.

Tiếp tục như vậy ta sẽ có lời giải của bài toán. Hình sau đây mô tả việc chứng minh bài toán:





Hình vẽ 2.21. Cây suy diễn trong thủ tục Wong .H

<2> Thủ tục hợp giải (Resolution) của Robinson

Bản chất của quá trình chứng minh bằng hợp giải của Robinson là chứng minh bằng phản chứng. Để chứng minh từ GT_1, ...,GT_m suy ra một trong các kết luận KL_1,..., KL_n ta lấy phủ định của KL_1,..., KL_n hợp với giả thiết ra suy ra mâu thuẫn.

* *<u>Thủ tục Robinson 1</u>* (Dùng cho logic mệnh đề)

Vào: Các biểu thức mệnh đề giả thiết GT_1,..., GT_m

Các biểu thức mệnh đề kết luận KL_1,..., KL_n

Ra: Thông báo "Thành công" nếu GT_1 [illegible] T_m [illegible] L_n.

Phương pháp:

{For i=1 to m do { Trans(GT_i); P [illegible] T_i;}

For i=1 to n do {Trans(KL_i); P [illegible] L_i;}

/* P=MĐ$_1$,...,MĐ$_k$, k=m+n*/

If mt(P) then exit("Thành công");

P_1=

While P [redacted]P_1 and [redacted]t(P) do {P_1=P; hopgiai(P);}

If mt(P) then exit (“Thành công”)

Else exit (“Không thành công”);}

Ở đây, hàm mt(P) kiểm tra xem trong P có hai mệnh đề p, q sao cho p=[redacted] hoặc q=[redacted] không?. Nó trả về giá trị True nếu tồn tại cặp p, q như thế và trả về giá trị False nếu ngược lại. Cụ thể:

Function mt(P);

{mt=false;

for each p[redacted] do

for each q[redacted] and q[redacted] do

if p=[redacted] or q=[redacted] then return (true)}

Còn thủ tục hopgiai(P) tìm cách hợp các mệnh đề dạng p=[redacted] với mệnh đề q=a[redacted] thành một mệnh đề r = b[redacted]

Procedure Hopgiai(P);

{for each p[redacted] do

for each q[redacted] and q[redacted] do

if p=[redacted] and q=a[redacted] then P[redacted]}}

Ví dụ 2.23. Chứng minh rằng từ (a[redacted])[redacted](b[redacted])[redacted] a,b suy ra d.

Đưa các giả GT_i và KL_j về dạng chuẩn và xây dựng P ta có

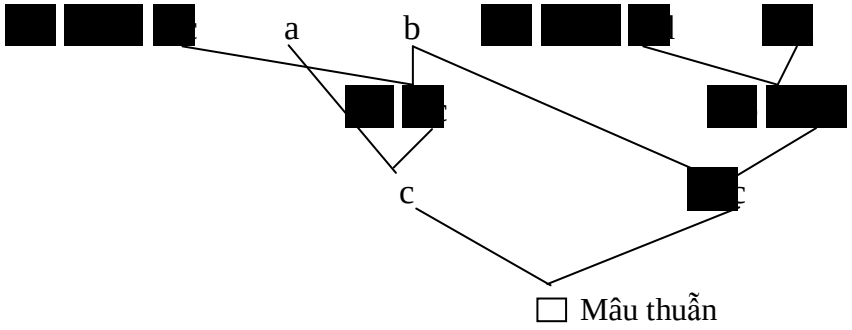
P ={[redacted], a, b, [redacted].

Viết các các xâu thành các dòng riêng biệt dưới dạng

1. [redacted]
2. [redacted]
3. a
4. b
5. [redacted]

Thực hiện hợp giải các mệnh đề:

6. [redacted] Res(1B, 4) /* Hợp giải dòng 1 với dòng 4. Chữ B chỉ*/
7. [redacted] Res(2C, 5) /* rằng thành phần thứ hai của dòng 1 */
8. c Res(3, 6A)
9. [redacted] Res(4, 7A)
10. Mâu thuẩn giữa 8 và 9.



Hình 2.22. Cây mô tả quá trình hợp giải

* *<u>Thủ tục Robinson 2</u>* (Dùng cho logic vị từ)

Thuật giải Resolution có thể mở rộng để giải quyết các bài toán chứng minh trong logic vị từ cấp 1. Việc hợp giải hai biểu thức vị từ được thực hiện như sau:

Giả sử có hai biểu thức: A= P [redacted]Q_1[redacted]Q_2 [redacted]Q_k và B=[redacted]R_1 [redacted]R_2 [redacted]R_t. Khi đó phép hợp giả cho ta biểu thức C = Q_1[redacted]Q_2 [redacted]Q_k [redacted]R_1 [redacted]R_2 [redacted]R_t.

Do các vị từ P_i, Q_i và R_j phụ thuộc và các biến nên để tạo ra các cặp đối ngẫu thực sự P và [redacted] ta phải thực hiện các phép gán.

Cách chọn phép gán: Để đưa vị từ $P(x_1, x_2, ...,x_n)$ về dạng $P(t_1, t_2, ...,t_n)$ ta chọn phép gán $q= \{ t_1/x_1, t_2/x_2, ...,t_n/x_n\}$ để thay thế mỗi ***biến*** x_i bởi t_i, trong đó t_i có thể là biến, hằng hoặc biểu thức.

Thủ tục Resolution cho logic vị từ có thể phát biểu như sau:

Bước 1: Đưa các GT_i và KL_j về dạng chuẩn sau:

[redacted]

sao cho mọi biến có mặt trong P_{ij} đều thuộc vào tập $\{x_1, x_2, \ldots, x_k\}$. Muốn vậy ta thực hiện các thao tác sau:

1. Khử bỏ các dấu kéo theo và tương đương nhờ A [redacted]

2. Đưa dấu phủ định vào trong cùng chừng nào có thể, nhờ các phép biến đổi:

- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]

3. Thay tên biến để cho mỗi lượng từ chỉ có một tên biến riêng.

4. Khử bỏ các lượng từ tồn tại nhờ:

- [redacted] P(x) chuyển thành P(a)
- [redacted] P(x,y) chuyển thành P(x,g(x)). Hàm g(x) được gọi là hàm Scholem.

5. Chuyển mọi lượng từ [redacted] về đầu biểu thức, phần biểu thức gọi là ma trận.

6. Đưa ma trận về dạng [redacted] nhờ áp dụng A[redacted]

7. Loại bỏ các lượng từ [redacted]

8. Thay thế các liên kết [redacted] bởi các dấu phẩy, mỗi dòng được gọi là một câu.

<u>Ví dụ:</u> Xét [redacted]

Áp dụng các bước để đưa về dạng chuẩn như sau:

1. [redacted]
2. [redacted]
3. [redacted]
4. [redacted]
5. [redacted]
6. [redacted]
7. [redacted]

8. Tách câu và viết thành các dòng

[illegible](x)[illegible](y)[illegible](f(x,y))

[illegible](x)[illegible](x,g(x))

[illegible](x)[illegible](g(x))

Bước 2. Nếu tìm được một cặp câu P_1, P_2 và một phép gán q sao cho P_{1q}=[illegible]$_{2q}$ thì thông báo "Thành công" và thuật toán dừng. Ngược lại sang bước 3.

Bước 3: Tìm cặp câu P = P_0 [illegible]P_1[illegible].. [illegible]$_k$ và Q = Q_0 [illegible]Q_1 [illegible]Q_2 [illegible]$_t$ và phép gán q sao cho P_{0q} = [illegible]$_{0q}$. Thực hiện hợp giải câu P với câu Q được câu

R= P_1[illegible].. [illegible]$_k$ [illegible]Q_1 [illegible]Q_2 [illegible]Q_2 [illegible]$_t$

và bổ sung câu R vào danh sách các câu.

Bước 4: Nếu không thể xây dựng thêm được các hợp giải và không có cặp câu đối ngẫu thì bài toán sai, ngược lại quay lại bước 3 hoặc bài toán được giải quyết xong và thông báo "Thành công".

<u>Kết quả:</u> Nếu từ GT_1[illegible]GT_2[illegible].. [illegible]T_m [illegible]L_1 [illegible]KL_2 [illegible]L_n thì thủ tục Resolution dừng và đưa ra thông báo thành công.

<u>Ví dụ 2.24</u>. Hãy chứng minh rằng An là sinh viên trường ĐHV biết rằng:

- An là sinh viên lớp TIN 40
- Lớp TIN 40 là thuộc khoa CNTT
- Khoa CNTT là thuộc trường ĐHV.

Để chứng minh điều khẳng định trên, ta đưa vào vị từ

P(x,y): nghĩa là "x thuộc y".

Khi đó ta có các giả thiết: P(An, T40), P(T40, CN), P(CN, DHV) và kết luận cần chứng minh là P(An, DHV). Ngoài ra P(x,y) là quan hệ có tính bắc cầu, nghĩa là [illegible] [illegible] [illegible] P(x,y)[illegible]P(y,z)[illegible]x,z).

Viết các biểu thức thành từng dòng và trong mỗi dòng thay dấu [illegible]bởi dấu phẩy:

1. [redacted](x,y) , [redacted](y,z), P(x,z)
2. P(An, T40)
3. P(T40, CN)
4. P(CN, DHV)
5. [redacted](An, DHV) /*Phủ định của kết luận*/

Ta có các bước hợp giải sau

6. [redacted](T40, z), P(An, z) Res(1A, 2) q_1 ={An/x, T40/y }
7. P(An, CN) Res(3, 6A) q_2 ={CN/z }
8. [redacted](An, y), [redacted](y,DHV) Res(1C, 5) q_3 ={An/x, DHV/z}
9. [redacted](CN, DHV} Res(7, 8A) q_4 ={CN/y}
10. Mâu thuẫn Res(4, 9).

Ví dụ 2.25. Chứng minh rằng nếu nửa nhóm có đơn vị mà mỗi phần tử đều luỹ linh thì nửa nhóm đó là giao hoán.

Nhắc lại một số khái niệm:

+ *Nửa nhóm:* Là tập X[redacted]ên đó có xác định một phép toán hai ngôi có tính chất kết hợp. Nếu phép toán là phép cộng (nhân) thì ta có nửa nhóm cộng (nhân). Trong ví dụ này ta sẽ xét nửa nhóm nhân.

+ *Phần tử đơn vị*: Phần tử e của nửa nhóm X được gọi là phần tử đơn vị nếu với mọi x[redacted]X ta có e.x=x.e=x.

+ *Phần tử luỹ linh*: Phần tử x[redacted]X được gọi là phần tử luỹ linh nếu x.x=e.

Để chứng minh khẳng định trên ta sử dụng vị từ:

P(x, y, z) : nghĩa là x.y=z.

Từ các giả thiết và kết luận của bài toán ta có:

1. P(e, x_1, x_1) /* e.x_1=x_1 */
2. P(x_1, e, x_1) /* x_1e=x_1 */

Từ tính kết hợp ta có: [redacted] (x_1.x_2).x_4=x_1.(x_2.x_4). Đặt x_1.x_2=x_3, x_2.x_4=x_5.

Khi đó, tính chất này có thể được hiểu là:

Nếu $x_1x_5=x_6$ thì $x_3x_4=x_6$ và nếu $x_3x_4=x_6$ thì $x_1x_5=x_6$. Do đó

$(P(x_1, x_2, x_3) \rightarrow P(x_2, x_4, x_5) \rightarrow (x_1, x_5, x_6)) \rightarrow x_3, x_4, x_6)$.

Đưa về dạng chuẩn ta có:

$\neg(x_1, x_2, x_3) \vee \neg(x_2, x_4, x_5) \vee \neg(x_1, x_5, x_6) \vee P(x_3, x_4, x_6)$.

Đưa về dạng câu ta có:

$\neg(x_1, x_2, x_3), \neg(x_2, x_4, x_5), \neg(x_1, x_5, x_6), P(x_3, x_4, x_6)$.

Vậy ta có

3. $\neg(x_1, x_2, x_3), \neg(x_2, x_4, x_5), \neg(x_1, x_5, x_6), P(x_3, x_4, x_6)$

Hoàn toàn tương tự ta có:

4. $\neg(x_1, x_2, x_3), \neg(x_2, x_4, x_5), \neg(x_3, x_4, x_6), P(x_1, x_5, x_6)$

5. $P(x_1,x_1,e)$ /* x_1. x_1= e */

Ta cần chứng minh tính chất giao hoán: $\forall \forall$ x.y=y.x. Lấy phủ định ta có $\exists$ x.y $\exists$.x. Thay biểu thức dạng $\forall$ P(x) bởi P(a) ta có P(a,b,c) và $\exists$(b,a,c). Vậy ta có:

6. P(a,b,c)
7. $\exists$(b,a,c)

Thực hiện các bước hợp giải:

8. $\exists(x_2,x_1,x_3), \exists(x_2,e,x_6), P(x_3,x_1,x_6)$ Res(3B,5), $q_1=\{x_1/x_2,x_1/x_4,e/x_5, x_2/x_1\}$
9. $\exists(x_1,x_4,x_5), \exists(e,x_4,x_6), P(x_1,x_5,x_6)$ Res(4A,5) $q_2=\{x_1/x_2, e/x_3\}$
10. $\exists(b,x_4,x_5), \exists(a,x_5,x_6), P(c,x_4,x_6)$ Res(3A,6) $q_3=\{a/x_1,b/x_2,c/x_3\}$
11. $\exists(x_2,x_1,x_5), P(x_2,x_5,x_1)$ Res(1,9B) $q_4=\{x_1/x_4, x_1/x_6, x_2/x_1\}$
12. $\exists(x_1,x_2,x_3), P(x_3,x_2,x_1)$ Res(2,8B) $q_5=\{x_1/x_2, x_1/x_6, x_2/x_1\}$
13. $\exists(a,e,x_6), P(c,b,x_6)$ Res(5,10A) $q_6=\{b/x_1, b/x_4, e/x_5\}$
14. P(c,b,a) Res(2,13A) $q_7=\{a/x_1, a/x_6\}$
15. P(c,a,b) Res(11A,14) $q_8=\{c/x_2, b/x_1, a/x_5\}$
16. P(b,a,c) Res(12A,15) $q_9=\{c/x_1, a/x_2, b/x_3\}$
17. Mâu thuẫn Res(7,16).

Nhận xét: Trong ví dụ trên ta chỉ sử dụng một vị từ. Trên thực tế đôi khi sử dụng nhiều vị từ là làm cho bài toán đơn giản hơn. Ta xét ví dụ sau:

Ví dụ 2.26. Nếu xem một ai đó đi lừa dối người khác là kẻ bịp bợm và bất kỳ ai đồng tình với kẻ bịp bợm cũng là bịp bợm, hơn nữa trong tập thể có một người nhút nhát đồng tình với kẻ lừa dối thì chắc chắn là có một kẻ bịp bợm tính tình nhút nhát.

Ta sử dụng các vị từ sau:

BB(x): x là kẻ bịp bợm.

LD(x): x là kẻ lừa dối.

NN(x): x là kẻ nhút nhát.

ĐT(x,y): x đồng tình với y.

Khi đó ta có:

1. [redacted]D(x), BB(x)
2. [redacted]T(x,y), [redacted]B(y), BB(x)
3. NN(a)
4. LD(b)
5. ĐT(a,b)
6. [redacted]B(x), [redacted]N(x)
7. [redacted]B(a) Res(3, 6B) q_1 ={a/x}
8. BB(b) Res(1A, 4), q_2 ={ b/x}
9. [redacted]N(b) Res(6A, 8), q_3 ={b/x}
10. [redacted]B(b), BB(a) Res(2A, 5), q_4 ={a/x, b/y}
11. [redacted]B(b) Res(7, 10B)
12. Mâu thuẫn Res(8,11).

Tóm lại:

Phương pháp của Robinson để chứng minh định lý có nội dung như sau:

[redacted] Lấy các giả thiết hợp cùng phủ định của kết luận và suy ra mâu thuẫn. Muốn vậy ta dùng kỹ thuật hợp giải: Sử dụng các phép gán q={t_1/x_1, ...,t_n/x_n} để

gán cho các biến $x_1,...x_n$ các biểu thức t_i, từ đó tìm ra cặp đối ngẫu thực sự P và [redacted]. Nếu xuất hiện cặp đối ngẫu P và [redacted] thì dừng và khẳng định định lý đúng.

[redacted] Thực chất quá trình hợp giải là đi tìm các cặp câu $P_1=P_1^0$ [redacted] và $P_2=P_2^0$ [redacted] và phép gán P_{1q}^0 [redacted] P_{2q}^0. Xây dựng hợp giải $(Q_1$ [redacted] R_1 [redacted] $)_q$ và bổ sung vào danh sách các câu.

d. Giải quyết vấn đề nhờ logic vị từ

Trong mục A chúng ta đã nghiên cứu hai phương pháp chứng minh định lý tiêu biểu, ở đây sẽ nghiên cứu việc xác định các phép gán trị cho các biến để từ các giả thiết GT_1 [redacted] T_m suy ra KL_1 [redacted] L_n.

Hai cách giải quyết:

1. Lưu lại vết các phép gán giá trị nhận được cho đến khi đưa đến mâu thuẫn.

Ta đưa vào khái niệm hợp các phép gán. Giả sử [redacted] là hai phép gán trị, hợp của chúng được ký hiệu bởi [redacted] sao cho đối với mọi biểu thức P ta có: P[redacted]=(P[redacted]

Ví dụ 2.27. Giả sử Mai và Dương rất thân nhau. Đi đâu Mai và Dương cũng có nhau. Hơn nữa ta biết rằng hiện nay Mai đang ở thư viện. Hỏi Dương đang ở đâu?

Ta đưa vào vị từ:

P(x,y): x đang ở vị trí y.

Khi đó:

[redacted] P(Mai, x) [redacted] Dương, x)

P(Mai, Thư viện)

Cần tìm giá trị của P(Dương,x).

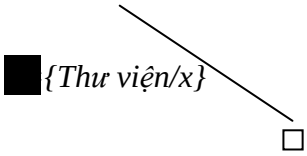
Ta có:

[redacted] (Mai,x) [redacted] (Dương,x) [redacted] (Dương,x) P(Mai, Thư viện)

[redacted]

[redacted] (Mai,x)

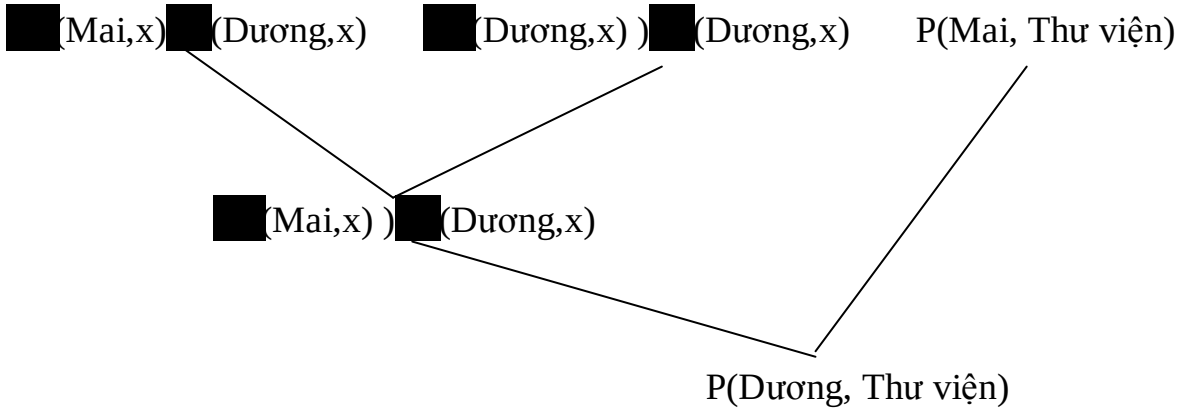
Hình 2.23. Đồ thị hợp giải



Nhờ hợp phép gán {thư viện/x}, ta có câu trả lời hiện nay Dương đang ở thư viện.

2. Cải biên đồ thị lời giải

Bên cạnh biểu thức là phủ định của kết luận KL_j cần chứng minh ta thêm vào phủ định của chính nó (tức là KL_j) và giữ nguyên các phép hợp giải giống như ở trong đồ thị hợp giải. Kết quả là nhận được P(Dương, Thư viện) có nghĩa là câu trả lời đã được xây dựng xong.



Hình 2.24. Đồ thị chứng minh cải biên

2.6.4. Biểu diễn tri thức và suy diễn

Như đã nói ở trong 2.5 về cấu trúc các hệ thống giải quyết vấn đề, có hai xu hướng giải quyết vấn đề sau:

-Tổng quát hoá cấu trúc bài toán, đưa về các dạng quen thuộc (không gian trạng thái, quy bài toán về bài toán con, sử dụng logic hình thức). Ở đây mới chú ý đến các thông tin về bài toán. Sau đó sử dụng các chiến lược tìm kiếm.

-Cụ thể hoá bài toán trên cơ sở sử dụng các tri thức đặc tả của từng bài toán và của các chuyên gia giải quyết vấn đề để xây dựng các tri thức suy diễn (Xem chương 3)

2.6.5. Phương pháp GPS (General Problem Solving)

Phương pháp GPS còn được gọi là phương pháp Mục đích-Phương tiện (Endo-Means) gồm 3 yếu tố cơ bản:

- Xác định không gian biểu diễn vấn đề W và tập các trạng thái, trạng thái đầu S_0, trạng thái cuối S_* và tập các phép biến đổi trạng thái $O=\{O_i \mid O_i: S_i \rightarrow S_j\}$.

- Xác định tập các kiểu khác biệt giữa các trạng thái trong không gian

$$\Delta\{\Delta_1, \ldots, \Delta_n\}$$

- Xây dựng ma trận M với các cột ứng với các toán tử, các hàng ứng với các kiểu khác biệt, $M=(m_{ij})_{m\times n}$.

$$m_{ij}=\begin{cases}1 \text{ nếu phép biến đổi } O_j \text{ làm giảm sự khác biệt } \Delta_i\\ 0 \text{ nếu ngược lại.}\end{cases}$$

Thủ tục GPS

Vào: Tập các kiểu khác biệt giữa các trạng thái $\Delta\{\Delta_1, \ldots, \Delta_n\}$

Tập các toán tử $O=\{O_1, O_2, \ldots, O_n\}$.

Trạng thái đầu S_0, trạng thái cuối S_*.

Ra: Thông báo "Thành công" nếu có thể đưa S_0 về S_*.

Phương pháp:

```
{ S=S0; OP=Δ
While S ≠ S* do
  {D=match(S, S*);    /*Giả sử D={Δ1,..., Δn} */
   Δ=chọn(D); /*Chọn ra sự khác biệt quan trọng nhất */
              /* Giả sử Δi là sự khác biệt thứ i trong Δ và OP={Oj |mij=1} */
```

For j=1 to n do

if m_{ij}=1 then OP=OP [redacted]O_j};

O[redacted]t(OP); /* chọn ra toán tử O_j làm giảm sự khác biệt [redacted]nhiều nhất */

S=O(S) }}

Ở đây thủ tục match(S, S_*) cho phép xác định những sự khác biệt giữa S và S_*.

Nếu S và S_* là các đỉnh trong đồ thị, một ví dụ đơn giản về sự khác biệt giữa S và S_* là độ dài đường đi ngắn nhất từ S tới S_*.

Ví dụ 2.28. Chứng minh rằng R [redacted]) [redacted]Q [redacted]P) [redacted]R. Một thao tác đơn giản là quy bài toán về dạng:

1) R [redacted]) [redacted]Q [redacted]P) [redacted]R

và 2) (Q [redacted]P) [redacted]R [redacted]

Sau đó áp dụng thuật giải Wong .H, hoặc Robinson.

Ở đây, ta minh hoạ việc giải bài toán bằng cách áp dụng thủ tục GPS.

+ Không gian biểu diễn vấn đề là không gian biểu diễn các mệnh đề logic.

Trạng thái đầu S_0 = R [redacted])

Trạng thái đích S_* = (Q [redacted]P) [redacted]R

Các phép toán là các phép biến đổi tương đương. (xem trang 50)

+ Tập các kiểu khác biệt gồm có 6 kiểu khác biệt:

[redacted]: ở biểu thức này có một biến v nào đó mà ở biểu thức khác không có.

[redacted]: Một biến v nào đó thuộc vào hai biểu thức với số lần xuất hiện khác nhau.

[redacted]: Có sự khác biệt về dấu [redacted]ủa biểu thức. Một trong hai biểu thức được bắt đầu bởi dấu [redacted]

[redacted]: Khác nhau về số các phép toán.

[redacted]: Khác nhau về phương pháp nhóm các mệnh đề.

[redacted]: Khác nhau về vị trí các thành phần trong hai biểu thức.

+ Xây dựng ma trận M: Để đơn giản, ta chỉ lấy các cột tương ứng với R_1 [redacted] $_6$, R_9,

	R_1	R_2	R_3	R_4	R_5	R_6	R_9
[redacted]							
[redacted]			1				
[redacted]		1			1	1	1
[redacted]					1	1	
[redacted]				1			
[redacted]	1	1					

Đặt $L_1 = S_0$ = R [redacted]), $L_0 = S_*$ = (Q [redacted]) [redacted]R

Đích 1: Biến đổi L_1 về L_0; Xác định sự khác biệt [redacted].

Đích 2: Loại bỏ sự khác biệt [redacted] giữa L_1 và L_0.

Xác định tập các toán tử chấp nhận được {R_1, R_2}

Đích 3: Áp dụng R_1 vào L_1.

Đích 4: Biến đổi L_1 sang dạng thích hợp cho việc áp dụng R_1.

Xác định được A = R, B = [redacted].

Tạo ra biểu thức mới L_2= ([redacted]) [redacted]R.

Đích 5: Biến đổi L_2 về L_0.

Xác định sự khác biệt [redacted].

Đích 6: Loại bỏ sự khác biệt [redacted] giữa L_2 và L_0.

Xác định tập các toán tử chấp nhận được {R_5, R_6}.

Đích 7: Áp dụng R_5 vào L_2.

Đích 8: Biến đổi L_2 về dạng thích hợp để có thể áp dụng R_5.

Không xác định được A và B, nên toán tử R_5 bị loại. Chuyển sang R_6.

Đích 9: Áp dụng toán tử R_6 vào L_2.

Đích 10: Biến đổi L_2 về dạng thích hợp để có thể áp dụng R_6.

Xác định được A= B=Q.

Tạo ra biểu thức mới L_3=(Q) R.

Đích 11. Biến đổi L_3 về L_0.

Xác định sự khác biệt .

Đích 12: Xoá bỏ sự khác biệt giữaL_3 và L_0.

Tập toán tử chấp nhận được {R_1, R_2}.

Đích 13: Áp dung R_1 vào L_3.

Đích 14: Biến đổi L_3 về dạng thích hợp để có thể áp dụng R_1.

Ta xác định A = B = Q.

Xây dựng L_4=(Q R.

Đích 15: Biến đổi L_4 về L_0.

Xác định sự khác biệt giữa L_4 và L_0 .

Đích 16: Loại bỏ sự khác biệt giữa L_4 và L_0.

Xác định tập các toán tử chấp nhận được {R_2, R_5, R_6, R_9}

Đích 17: Áp dụng R_2 vào L_4.

Đích này bị loại vì không thể đưa về đạng có thể áp dụng R_2.

Chuyển sang áp dụng R_5.

Đích 18: Áp dụng R_5 vào L_4.

Đích này bị loại. Chuyển sang áp dụng R_6.

Đích 19: Áp dụng R_6 vào L_4.

Đích này bị loại và chuyển sang R_9.

Đích 20: Áp dụng R_9 vào L_4.

Đích 21: Biến đổi L_4 về dạng thích hợp để có thể áp dụng R_9.

Ta xác định được A=P

Xây dựng L_5= (Q) R

L_5 trùng với L_0. Quá trình đã được chứng minh.

2.7. GIẢI QUYẾT VẤN ĐỀ VÀ CÁC KỸ THUẬT HEURISTICS

Một số giải thích về Heuristics:

1. Theo từ điển tiếng anh Oxford: "*Heuristics là nghệ thuật tìm kiếm chân lý. Nói riêng, Heuristics là đặc trưng của quá trình học nhờ đó các học sinh học được cách tự tìm ra cách giải thích các hiện tượng tự nhiên*".
2. Theo Feigenbaum Feldman: "*Heuristics là các quy tắc, phương pháp, chiến lược, mẹo giải hay phương cách nào đó nhằm làm giảm khối lượng tìm kiếm lời giải trong không gian bài toán cực lớn*".

3. Trong các tài liệu trí tuệ nhân tạo: Heuristics được coi là khái niệm đối lập với thuật giải (Algorithm). Các dấu hiệu đặc trưng của bài toán cho phép xác định nhanh chóng lời giải được gọi là các Heuristics và các quy tắc sử dụng các Heuristics được gọi là các quy tắc Heuristics.

Các khía cạnh quan trọng của lập trình Heuristics:

1. Đảm bảo tính tổng quát
2. Điều khiển quá trình tìm kiếm
3. Điều khiển quá trình suy diễn
4. Các hàm đánh giá
5. Đối sánh các cấu trúc thông tin
6. Khả năng học
7. Khả năng đặt kế hoạch.

Trong đó các hàm đánh giá được coi là công cụ có hiệu lực nhất. Khi lập trình Heuristics cần chú ý hai vấn đề sau:

+ Tách các dấu hiệu có ích

+ Tổ hợp các dấu hiệu này để nhận được các hàm đánh giá tốt.

Một kỹ thuật Heuristics được coi là hợp lý nếu nó cho phép đánh giá được các khả năng để làm rõ khả năng nào tốt hơn các khả năng còn lại.

Các kỹ thuật Heuristics được chia thành 2 lớp chính:

+ Lớp định tính: Các luật ***nếu*** ... ***thì***...

+ Lớp định lượng: Các hàm ước lượng Heuristics.

Hai cách đưa các thông tin Heuristics đặc tả vào các thủ tục tìm kiếm:

+ Các kỹ thuật Heuristic được nạp ngay trong các biểu diễn của bài toán.

+ Các hàm đánh giá Heuristic nhằm lượng hoá "giá trị" của mỗi khả năng thực hiện.

2.8. MỘT SỐ PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ KHÁC

2.8.1. Phương pháp tạo và kiểm tra (Generate and test)

Nội dung:

1.Tạo một lời giải thử nghiệm nào đó. Trong không gian bài toán điều này có nghĩa là đưa ra một đường đi bộ phận xuất phát từ trạng thái đầu.

2. Kiểm tra xem nó có phải là lời giải thực sự hay không?

3. Nếu đúng thì dừng và đưa ra thông báo thành công. Ngược lại chuyển về bước 1.

Nhận xét:

-Về thực chất thủ tục tạo và kiểm tra sẽ cho lời giải nếu quả thật nó tồn tại và do tính tạo sinh ngẫu nhiên nên nó phải duyệt không gian bài toán một cách hệ thống.

-Thủ tục này tổng quát hơn thủ tục tìm kiếm theo chiều sâu, theo chiều rộng, sâu dần,...

2.8.2. Phương pháp leo đồi (Hill climbing)

Phương pháp này là một biến thể của phương pháp tạo và kiểm tra trong đó thông tin phản hồi khi kiểm tra được dùng để trợ giúp quá trình tạo sinh lời giải ở bước tiếp theo.

Nội dung:

1. Tạo sinh lời giải thử nghiệm giống như trong thủ tục tạo và kiểm tra. Nếu nó là lời giải thực sự thì dừng và đưa ra thông báo thành công. Ngược lại tiếp tục thực hiện bước sau.

2. Từ lời giải này, áp dụng một số luật nào đó để sinh ra một tập mới các lời giải thử nghiệm.

3. Với mọi phần tử trong tập các lời giải này, ta làm như sau:

- Nếu nó là lời giải thực sự thì dừng.

- Nếu ngược lại, ta xem phần tử này có phải là phần tử gần nhất so với lời giải thực sự cho đến thời điểm hiện tại không. Nếu đúng ta giữ lại. Ngược lại, sang bước 4.

4. Lấy phần tử tốt nhất nhận được ở trên và dùng nó như lời giải thử nghiệm mới và chuyển sang bước 2.

2.8.3. Phương pháp thoả mãn các ràng buộc (constraint satisfaction method)

Phương pháp này bổ trợ cho các phương pháp tìm kiếm đã nêu ở trên. Mục đích đặt ra là tìm các trạng thái của bài toán sao cho thoả mãn một tập ràng buộc nào đó. Quá trình tìm kiếm lời giải bao gồm hai phần:

+ Tìm kiếm trong không gian các ràng buộc

+ Tìm kiếm trong không gian bài toán ban đầu.

Nội dung:

Thực hiện các bước sau đây cho đến khi tìm được lời giải đầy đủ của bài toán hoặc tất cả các đường đi đều đã được duyệt nhưng không thu được kết quả.

1. Chọn một đỉnh chưa được xét trong đồ thị tìm kiếm.

2. Áp dụng các luật suy dẫn đối với các ràng buộc vào đỉnh đã chọn để sinh ra tất cả các ràng buộc mới có thể có.

3. Nếu tập các ràng buộc mới có chứa mâu thuẫn thì đưa ra thông báo bế tắc.

4. Nếu tập các ràng buộc mô tả lời giải đầy đủ của bài toán thì dừng và đưa ra thông báo thành công. Ngược lại sang bước 5.

5. Áp dụng các luật trong không gian trạng thái để tạo ra các lời giải bộ phận tương hợp với tập các ràng buộc hiện thời. Thêm các lời giải bộ phận này vào đồ thị tìm kiếm.

Ví dụ 2.30. Xét bài toán điền chữ

```
   S END
+ MORE
MONEY
```

Ràng buộc:

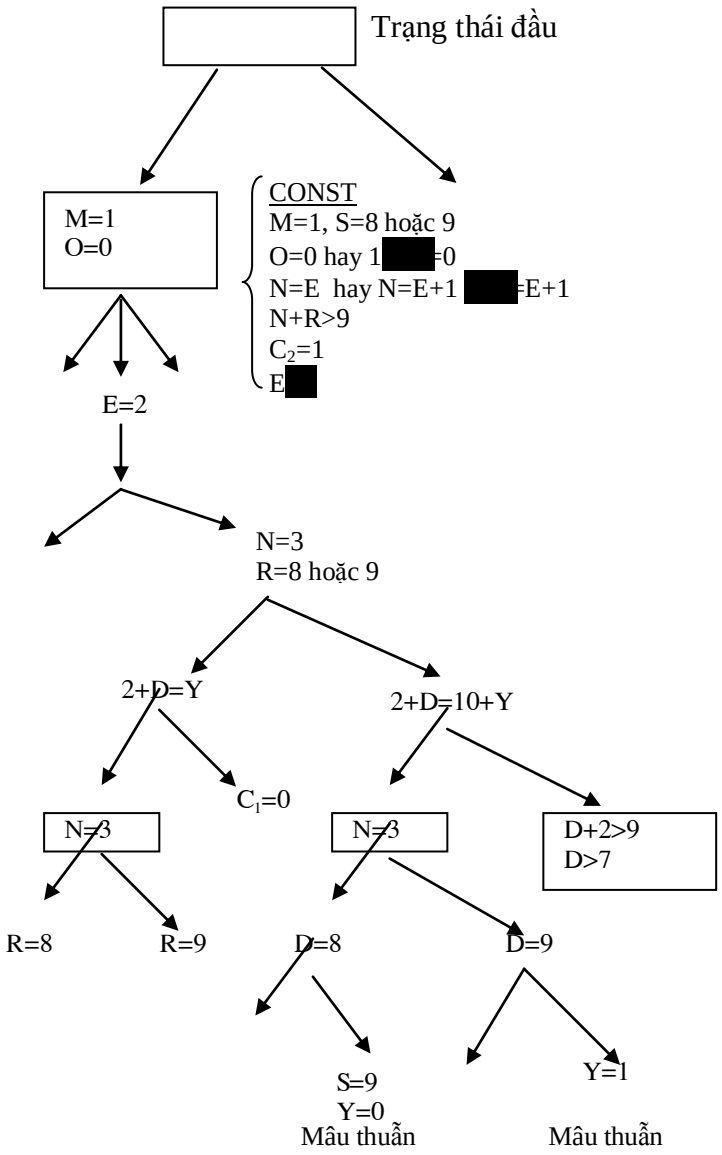
- Hai chữ khác nhau có giá trị khác nhau.

- Các ràng buộc về phép toán số học.

Các trạng thái:

- Trạng thái đầu: S, E, N, D, M, O, R, Y chưa xác định.

- C_1, C_2, C_3, C_4 chưa xác định /*C_i là các giá trị nhớ của các cột tính từ phải sang trái */



Hình 2.25. Đồ thị giải kết hợp và các ràng buộc

2.9. CÁC CHIẾN LƯỢC ĐIỀU KHIỂN

Như đã chỉ ra trong 2.5, quá trình giải quyết vấn đề bao gồm:

- Biểu diễn vấn đề.
- Các kỹ thuật tìm kiếm.
- Các chiến lược điều khiển.
- Các kỹ thuật Heuristics.

Các chiến lược điều khiển bao gồm các quyết định:

- Hướng tìm kiếm trong không gian bài toán.
- Xử lý cạnh tranh.

Hướng tìm kiếm

Ba cách lựa chọn hướng tìm kiếm:

- Xuất phát từ hình trạng đầu đi đến hình trạng cuối, gọi là chiến lược điều khiển dựa trên dữ liệu hay suy diễn tiến (data-driven control strategy, forward chaining strategy). Các kỹ thuật nêu trong 2.7.1, 2.7.2, 2.7.3 thuộc loại này.

- Xuất phát từ hình trạng cuối để ngược trở lại đi tới hình trạng đầu, gọi là chiến lược điều khiển dựa trên luật hay suy diễn lùi, hướng đích (Rule-driven control Strategy, backward chaining strategy goal-driven strategy).

- Xuất phát từ cả hình trạng đầu và hình trạng cuối (mixed control strategy).

Ba nhân tố ảnh hưởng tới quá trình lựa chọn hướng tìm kiếm:

- Số hình trạng đầu và số hình trạng đích. Ưu tiên hướng tìm kiếm từ tập ít các tình huống đi tới tập các tình huống nhiều hơn.

- Theo một hướng nào đó, hằng số phân nhánh (Branching factor) là số trung bình các nút có thể đạt tới từ một nút nào đó. Ưu tiên hướng tìm kiếm với hằng số phân nhánh nhỏ hơn.

- Yêu cầu giải thích quá trình lập luận đối với người sử dụng. Nếu có thì ưu tiên hướng tìm kiếm gần gũi với cách nghĩ của họ.

Ví dụ 2.31. - Đối với bài toán tìm đường đi từ nhà đến một nơi nào đó không quen biết, chiến lược điều khiển tốt là backward chaining

- Đối với bài toán tích phân nên dùng forward chaining.

Xử lý cạnh tranh

Xử lý cạnh tranh xuất hiện khi có hơn một khả năng có thể áp dụng.

Cách xử lý cạnh tranh:

- Chọn ngẫu nhiên.
- Chọn theo một tiêu chuẩn nào đó.
- Đưa ra một số ràng buộc. Các kỹ thuật đối sánh xem các đỉnh có thoả mãn các ràng buộc hay không:
  + Xây dựng chỉ dẫn.
  + Đối sánh các giá trị các biến.
  + Đối sánh xấp xỉ.
  + Lọc các đỉnh đã được đối sánh.

2.10. CÁC HỆ THỐNG GIẢI QUYẾT VẤN ĐỀ

Có thể đưa ra danh sách các hệ thống giải quyết vấn đề được nghiên cứu và cài đặt sau đây:

- Các chương trình chơi cờ tướng, cờ GO, cờ Đam,... Các chương trình trò chơi cờ carô, chơi bài.
- Các chương trình giải bài toán hình học phẳng, hình học không gian. Các chương trình giải quyết các bài toán thuộc lĩnh vực toán:
  + Tính tích phân bất định.
  + Chứng minh định lý sử dụng logic toán học sơ cấp.
  + Kiểm tra các chứng minh toán học.
  + Trợ giúp xử lý các biểu thức toán học.
  + Xác định những đặc điểm tương tự về hình học.
  + Chương trình giải tích hồi quy Heuristic.
  + Chứng minh định lý tự động sử dụng phương pháp Resolution.
  + Tìm các hàm tuyến tính để đánh giá và nhận dạng
- Các hệ thống MULTIPLE, GPS.
- Các hệ thống tự động dẫn xuất các câu trả lời.

Những hệ thống giải quyết vấn đề lớn:

- Hệ chuyên gia.
- Hệ trợ giúp quyết định.

Chương III

BIỂU DIỄN TRI THỨC VÀ SUY DIỄN

(KNOW LEDYE REPRESENTATION AND ENGINEERING)

3.1. NHẬP MÔN

Khái niệm biểu diễn tri thức được đề cập nhiều trong các tài liệu về TTNT. Tuy nhiên, chỉ đến khi xuất hiện các hệ chuyên gia và ứng dụng của nó, khái niệm này mới thực sự có cơ sở lý luận và thực tế vững chắc.

Ta hiểu biểu diễn tri thức là thể hiện các mô tả về thế giới bên ngoài dưới dạng sao cho các máy thông minh có thể đưa tới những kết luận về môi trường xung quanh nó, trên cơ sở một cách hình thức các mô tả này. Tri thức và suy diễn là hai thành phần trong bất kỳ một hệ xử lý dựa trên tri thức nào.

Các phương pháp biểu diễn tri thức chủ yếu:

- Biểu diễn nhờ logic hình thức
- Biểu diễn nhờ hệ sản xuất
- Biểu diễn nhờ mạng ngữ nghĩa
- Biểu diễn nhờ các Frame
- Biểu diễn dùng bộ ba OAV

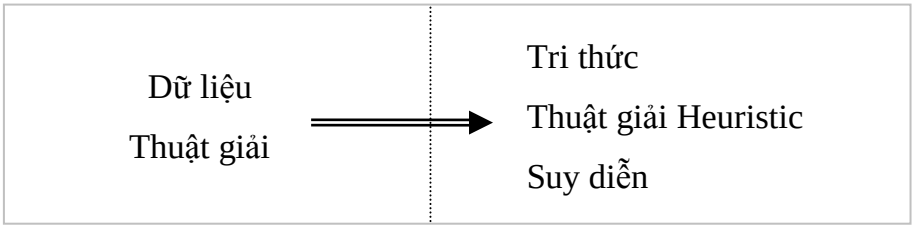
Trong bất kỳ phương pháp biểu diễn và xử lý tri thức nào chúng ta cần lưu tâm tới hai vấn đề:

- Các sự kiện (facts): Các thông tin về đối tượng
- Các phương pháp biểu diễn các sự kiện trong một hệ phát biểu hình thức nào đó.

3.2. TRI THỨC VÀ DỮ LIỆU

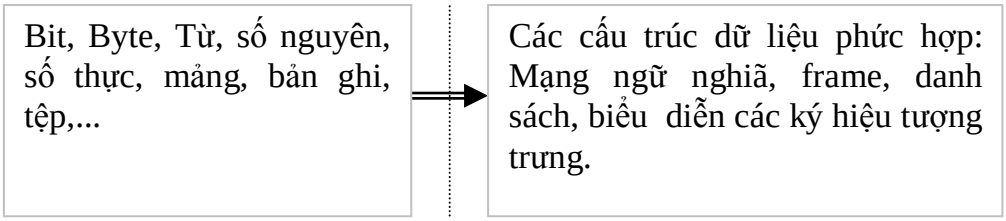
Trên đây ta đã đưa ra những so sánh căn bản giữa lập trình truyền thống (conventional programming) và lập trình TTNT (A.I. programming).

Sự tiến hoá có thể tóm tắt trong sơ đồ sau:

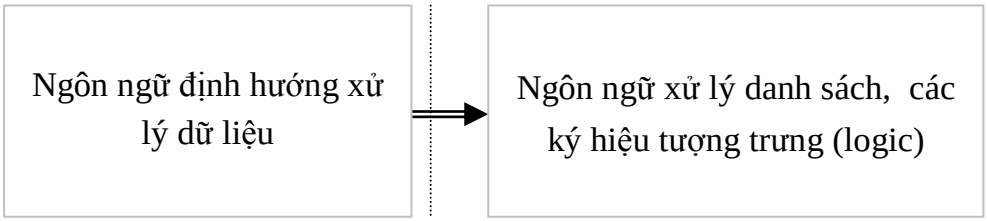


Lập trình truyền thống ⋯⋯▸ *Lập trình TTNT*

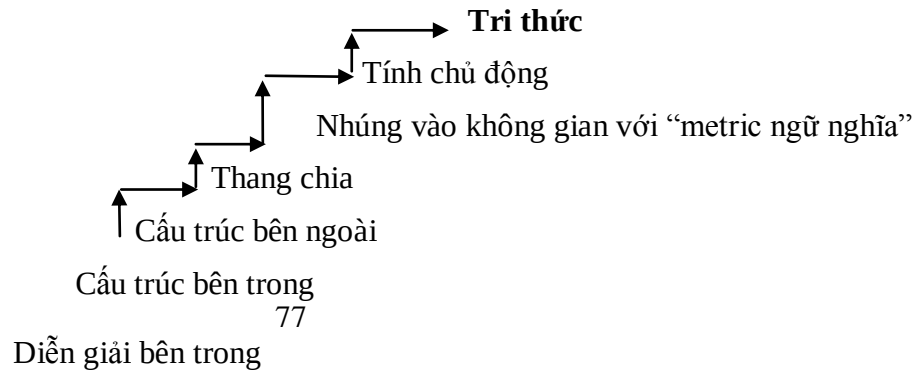
Theo quan điểm cấu trúc dữ liệu:

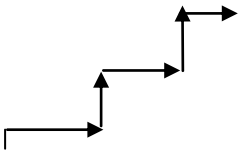


Theo quan điểm ngôn ngữ lập trình:



Như vậy dữ liệu và tri thức cùng là những dạng khác nhau của thông tin, nên rất khó có thể phân biệt rạch ròi giữa tri thức và dữ liệu. Tuy vậy, có thể dẫn ra các cung bậc khác nhau dẫn từ dữ liệu đến tri thức:





3.3. PHÂN LOẠI TRI THỨC

3.2.1. Các dạng tri thức

Tri thức tồn tại dưới ba dạng cơ bản sau:

- Tri thức mô tả: Cho ta một sự kiện mà không có thông tin về cấu trúc, mối liên hệ bên trong, bên ngoài cũng như phương pháp sử dụng tri thức đó.

Ví dụ 3.1: Vinh là thành phố đẹp.

- Tri thức thủ tục: Mô tả phương pháp kiến thiết, ghép nối và suy diễn các tri thức đã có, tạo nên những tri thức mới.

Ví dụ 3.2: Nếu a đúng và có luật suy diễn từ a [redacted] thì b đúng.

- Tri thức điều khiển: Dùng để chỉ ra phương pháp điều khiển sử dụng các tri thức mô tả và các tri thức thủ tục.

Ví dụ 3.3: - Các tri thức mô tả là các tri thức về đỉnh, cung.

- Các tri thức thủ tục là các thủ tục tìm kiếm.
- Các tri thức điều khiển là các chiến lược điều khiển.

3.2.2. Các cấp độ của tri thức

- Tri thức tĩnh, thuần nhất: Các tri thức bất biến đối với không gian, thơi gian.

Ví dụ 3.4: Nếu hàm số f(x) có đạo hàm trên đoạn [a, b] thì liên tục trên đoạn đó.

- Tri thức động phụ thuộc vào không gian, tình huống.

Ví dụ 3.5: Sỹ Thuỷ là vua phá lưới giải vô địch Quốc gia mùa bóng 1999-2000.

- Tri thức động phụ thuộc vào thời gian, quan hệ nhân quả.

Ví dụ 3.6: Kỷ lục nhảy cao của Việt nam là 2,02m.

3.4. BẢN CHẤT CỦA TRI THỨC CHUYÊN GIA

Chuyên gia là người có uy tín cao trong một lĩnh vực nào đó, được đông đảo mọi người công nhận là người có khả năng giải một lớp các bài toán hẹp nào đó mà phần lớn những người khác không thể giải quyết được như vậy.

Các chuyên gia có khả năng như vậy là vì họ có một lượng lớn các tri thức tích luỹ cùng với các tri thức đặc tả trong lĩnh vực cụ thể được lưu trong bộ nhớ dài hạn. Chẳng hạn, các chuyên gia cờ thế giỏi hoặc các nhà bác học nhận giải Nobel thường có thể chứa 5.10^4 đến 10^5 các bộ thông tin heuristic. (Theo các chuyên gia tâm lý học, để làm được như vậy phải mất ít nhất 10 năm).

Tri thức thu nạp được là thông tin nhận được được tổ chức, tạo chỉ dẫn và lưu trữ sao cho dễ dàng truy nhập tới. Việc thu nạp tri thức chính là quá trình tạo các lớp thông tin. Các tri thức thu nạp được bao gồm hai dạng:

- Các nguyên lý, các lý thuyết tổng quát.
- Các heuristic và các lý thuyết đặc tả.

3.5. CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

3.5.1. Biểu diễn tri thức sử dụng logic hình thức

Trong chương 2 (Phần 2.7.3) ta đã đưa vào các khái niệm cơ bản về logic mệnh đề và logic vị từ. Ở đó các tri thức về bài toán cần giải quyết được thể hiện dưới dạng các biểu thức logic.

Trong phần này ta quan tâm đến việc biểu diễn bằng logic vị từ các tri thức của các chuyên gia giải quyết vấn đề.

Một cách tổng quát các tri thức chuyên gia được thể hiện dưới dạng luật: GT[redacted]L, ở đây GT và KL là các biểu thức logic.

Ta thường xét tới các luật dưới dạng Horn:

$p_1 \wedge \ldots \wedge p_n \Rightarrow q_1 \vee \ldots \vee q_m$

Dạng 1: n=0, m=0, luật có dạng:

$(t_1,\ldots,t_k)$

nếu $t_1,\ldots,t_k$ là các hạng thức cụ thể nào đó thì nó cho ta một sự kiện trong CSDL. Ngược lại nếu t_i là biến thì nó cho ta một tập các sự kiện.

Dạng 2: n[illegible], m=0, luật có dạng:

$p_1 \wedge \ldots \wedge p_n \Rightarrow$

Dạng này thường được dùng để diễn đạt các câu hỏi trong các phương pháp chứng minh nhờ phản chứng nhằm dẫn ra mâu thuẫn (xem phương pháp Resolution).

Dạng 3: n[illegible], m=1, luật có dạng:

$p_1 \wedge \ldots \wedge p_n \Rightarrow$

Đây là dạng hay gặp nhất trong thực tế. Dạng này có nghĩa là nếu các sự kiện $p_1,\ldots,p_n$ đúng thì sự kiện q cũng đúng.

Dạng 4: n=0, m>1, luật có dạng:

$\Rightarrow q_1 \vee \ldots \vee q_m$

Dạng này thường được dùng khi thiếu thông tin.

Dạng 5: n[illegible], m[illegible]

$p_1 \wedge \ldots \wedge p_n \Rightarrow q_1 \vee \ldots \vee q_m$

Đây là dạng tổng quát nhất. Chẳng hạn: CHAME(x,y)[illegible]HA(x,y)[illegible]IE(x,y). Trong thực tế xử lý các dạng 4, 5 thường không sử dụng do tính phức tạp.

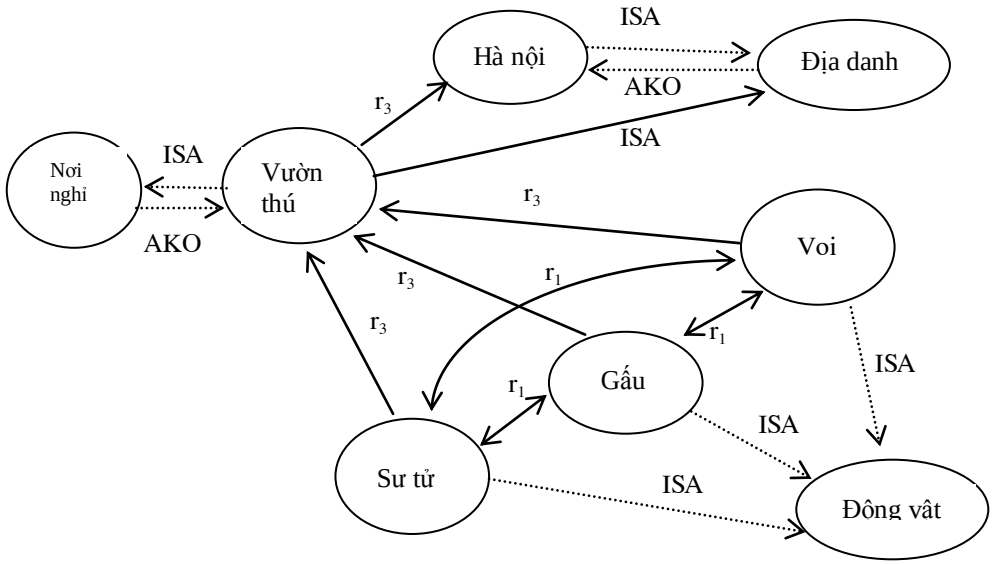
Tuy nhiên, một luật dạng này tương đương với một tập các luật dạng 3, nghĩa là nó tương đương với:

$$\begin{cases} p_1 \wedge \ldots \wedge p_n \wedge q_2 \wedge \ldots \wedge q_m \Rightarrow q_1 \\ p_1 \wedge \ldots \wedge p_n \wedge q_1 \wedge q_3 \wedge \ldots \wedge q_m \Rightarrow q_2 \\ \ldots\ldots \\ p_1 \wedge \ldots \wedge p_n \wedge q_1 \wedge \ldots \wedge q_{m-1} \Rightarrow q_m \end{cases}$$

3.5.2. Biểu diễn tri thức nhờ mạng ngữ nghĩa

Một cách hình thức mạng ngữ nghĩa là đồ thị định hướng G=(N,A), trong đó N là tập các đối tượng, các sự kiện hay khái niệm cụ thể, A là tập các mối liên hệ giữa các cặp đối tượng, sự kiện hay khái niệm.

Ví dụ 3.7: Về mạng ngữ nghĩa. A[redacted][2]



Hình 3.1. Một ví dụ về mạng ngữ nghĩa

Trong đó Voi, Gấu, Sư tử là những thể hiện cụ thể của frame "động vật", được mô tả bởi: (<Loài, .>,<Sống ở, .>,<Ăn, .>)

Các tên: Loài, sống ở, Ăn được gọi là Slot.

Hà nội, Vườn thú là những thể hiện của các frame "Địa danh" và "Nơi nghỉ" tương ứng.

- r_1 là quan hệ cùng có 4 chân.
- r_3 là quan hệ "ở tại".

- ISA là quan hệ loài đặc biệt "là" (is a)
- AKO là quan hệ đặc biệt "bao gồm" (a kind of)

3.5.3. Biểu diễn tri thức sử dụng các luật sản xuất

Các sản xuất có dạng: $p_1 \ldots p_m \ldots$ Tuỳ thuộc vào bản chất của lĩnh vực đang quan tâm mà có những ngữ nghĩa khác nhau về sản xuất.

- Trong logic vị từ $p_1,...,p_m$, q là những biểu thức logic, còn [redacted]ính là phép toán kéo theo theo nghĩa thông thường.
- Trong văn phạm: G=(N,T,P,S), N là tập các ký hiệu không kết thúc, T là tập ký hiệu kết thúc, S [redacted] là ký hiệu đầu, P là tập các sản xuất có dạng [redacted] ở đây [redacted]N U T)*. Mỗi sản xuất p: [redacted]ó nghĩa là nếu có xuất hiện xâu [redacted] thì có thể thay nó bởi xâu [redacted]Do vậy có thể định nghĩa: L(G)={[redacted]* [redacted]S[redacted]
- Trong ngôn ngữ lập trình, các sản xuất có nghĩa là các câu lệnh

  If $p_1 \ldots p_m$ **then** q

- Theo nghĩa thường gặp trong các hệ chuyên gia (expert system) cơ sở tri thức bao gồm hai bộ phận:

+ Cơ sở dữ liệu các sự kiện F (facts) F={$f_1,...,f_m$}.

+ Cơ sở các luật sản xuất R (rules).

Các luật sản xuất P có dạng: $f_{i1} \ldots f_{ik}$ [redacted]có nghĩa là nếu trong F đã có các sự kiện $f_{i1},...,f_{ik}$ thì nó chứa thêm sự kiện q.

3.5.4. Biểu diễn tri thức sử dụng các frame

Tư tưởng về frame (khung) do Miminsky đưa ra năm 1975. Frame về thực chất là sự tổng quát hoá của các bản ghi. Mỗi kiểu frame được mô tả bởi:

Tên frame

<slot-1>:<giá trị>

<slot-2>:<giá trị>

. . .

<slot-n>: <giá trị>

Trong đó <giá trị> có thể là:

- * Nếu lấy giá trị ngầm định (default)
- PART OF có nghĩa là frame này tạo thành một phần cấu thành của một frame khác.
- IS A có nghĩa là fame này là một biến thể của frame khác.

Lưu ý: Các frame được nối với nhau tạo thành một cấu trúc danh sách phức hợp giống như mạng ngữ nghĩa, song đối với các liên kết kiểu PART OF, ISA có sự xuất hiện của việc di truyền các thuộc tính.

3.6. CÁC KHÍA CẠNH XỬ LÝ TRI THỨC

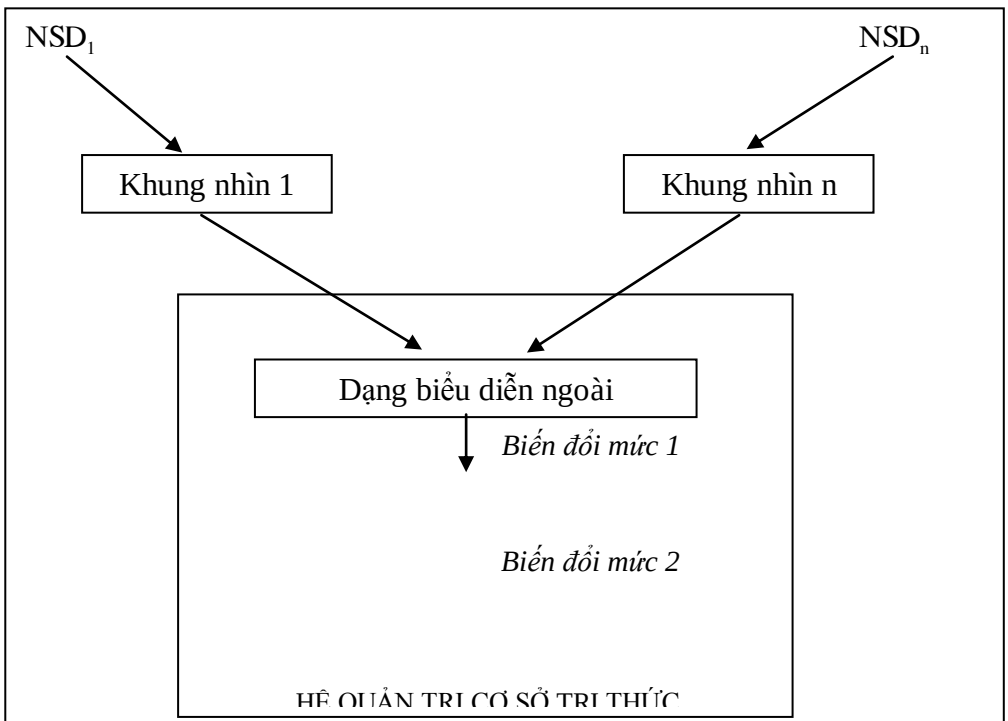
Ba khía cạnh cơ bản trong xử lý tri thức là: Suy diễn, tổng hợp và chuyển đổi tri thức.

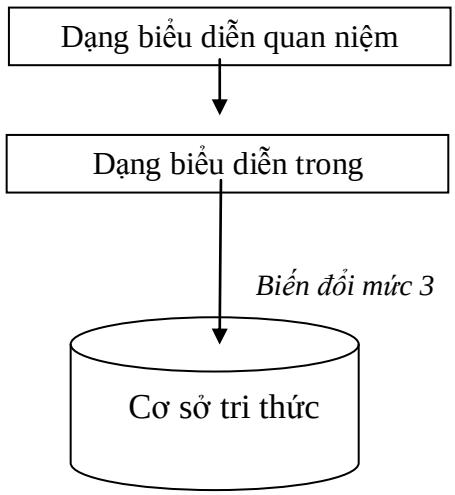
3.6.1. Chuyển đổi tri thức

Thông thường các hệ cơ sở tri thức thường biểu diễn tri thức theo 4 mức:

- Dạng biểu diễn ngoài
- Dạng biểu diễn quan niệm
- Dạng biểu diễn trong
- Dạng biểu diễn vật lý

Có ba loại phép chuyển đổi tri thức được mô tả trong hình sau:





Hình 3.2. Các mức biểu diễn tri thức và chuyển

3.6.2. Tổng hợp tri thức

Tổng hợp tri thức là quá trình liên kết, sắp xếp, khái quát hoá các nội dung tri thức làm cơ sở cho khả năng học, tạo ra những tri thức mới ở cấp độ cao hơn.

Có hai lớp kỹ thuật tổng hợp tri thức:

Lớp 1: Bao gồm các kỹ thuật xây dựng tri thức từ các dữ liệu

Lớp 2: Bao gồm các kỹ thuật xây dựng tri thức điều khiển từ các tri thức mô tả và tri thức thủ tục ban đầu.

Các kỹ thuật cơ bản bao gồm:

- ***Học của máy:***

- Học ngẫu nhiên
- Học thông qua mạng nơron
- Học thông qua xác định các tham số
- Học nhờ phương pháp GPS
- Học các quan niệm.
- Học tương tự.
- Học thông qua khám phá quy luật.

- ***Quy nạp:***

- Quy nạp không hoàn toàn
- Quy nạp hoàn toàn
- Quy nạp cấu trúc

■ ***Hướng dữ liệu:***

- Dựa trên dữ liệu mẫu
- Giải thuật di truyền

■ ***Tạo kiểm tra***

3.6.3. Suy diễn tri thức

Suy diễn tri thức bao gồm các cơ chế liên kết các tri thức đã có để suy dẫn ra các tri thức mới. Phương thức biểu diễn tri thức có ảnh hưởng lớn đến cơ chế suy diễn tri thức.

Các phương pháp suy diễn tri thức:

■ **Phương pháp suy diễn**

Trong logic mệnh đề, logic vị từ, dựa trên hai phương thức:

Modus ponens $\frac{A,\ A \rightarrow B}{B}$. Nghĩa là nếu A đúng và A → B đúng thì B đúng.

Modus tollens $\frac{A \rightarrow B}{\neg A}$. Nghĩa là nếu B sai và A → B đúng thì A sai.

Suy diễn tiến: Xuất phát từ các vị từ đã cho ban đầu, sử dụng các luật cho đến khi đưa ra kết luận mong muốn.

Suy diễn lùi: Xuất phát từ các kết luận mong muốn, xem những luật có khả năng suy ra chúng, thêm các tiền đề vào danh sách các kết luận cần chứng minh và tiếp tục như vậy.

■ **Phương pháp quy diễn**

Suy diễn trên cơ sở tương tự giữa các sự kiện.

■ **Phương pháp quy nạp**

- Quy nạp hoàn toàn.
- Quy nạp không hoàn toàn.
- Quy nạp cấu trúc.

3.7. CÁC HỆ CƠ SỞ TRI THỨC

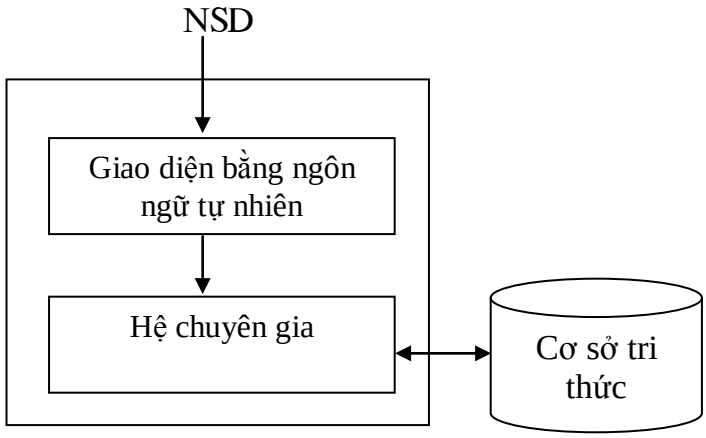
Vai trò của các kỹ sư xử lý tri thức

Các kỹ sư tri thức có trách nhiệm:

- Phát triển phần mềm hệ thống, quản trị phần mềm hệ thống, đảm bảo cập nhật, sửa đổi và suy diễn tri thức.
- Phân tích phương cách giải quyết vấn đề của chuyên gia con người.
- Trao đổi với các chuyên gia con người nhằm giúp họ có thể diễn đạt tri thức và cơ chế suy diễn dưới dạng có thể mã hoá trong máy tính.
- Bảo trì hệ thống, đảm bảo tính nhất quán (tính phi mâu thuẫn), tính không dư thừa của tri thức.

Các thành phần trong hệ cơ sở tri thức

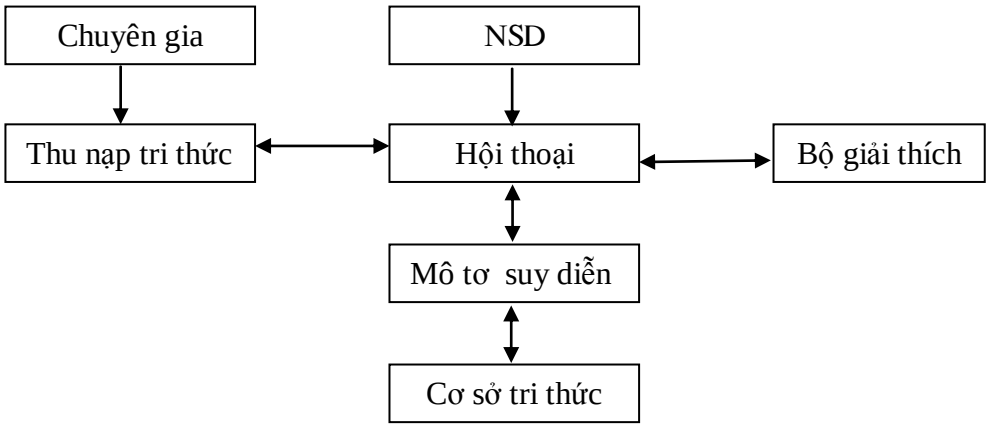
Các thành phần trong một hệ cơ sở tri thức và mối quan hệ giữa chúng được mô tả bởi hình vẽ sau:



Các loại hệ cơ sở tri thức:

- Hệ dựa trên tri thức có giao tiếp ngôn ngữ tự nhiên.
- Hệ dựa trên tri thức trong một phạm vi cụ thể.
- Hệ chuyên gia.

Các hệ chuyên gia



3.8. NGÔN NGỮ PROLOG

3.8.1. Giới thiệu

Prolog là ngôn ngữ cấp cao, theo dự tính đây là ngôn ngữ của hệ máy tính thứ 5. Các đặc điểm nổi bật của Prolog là:

- Đây là ngôn ngữ phi thủ tục.
- Cho phép hướng tới xử lý và lập trình logic.
- Cho phép hướng tới các cơ sở dữ liệu.
- Cho phép xử lý ngôn ngữ tự nhiên.

Cũng giống như các ngôn ngữ lập trình khác, Prolog có trình biên dịch được dùng phổ biến là Turbo Prolog. Sau đây là một số ứng dụng của Turbo Prolog:

+ Nhanh chóng tạo nên hình mẫu cho một chương trình ứng dụng.

+ Điều khiển và chỉ đạo các quá trình sản xuất công nghiệp.

+ Cài đặt các cơ sở dữ liệu quan hệ.

+ Dịch ngôn ngữ, hoặc ngôn ngữ tự nhiên của con người hoặc từ ngôn ngữ lập trình này sang ngôn ngữ lập trình khác.

+ Xây dựng các giao diện bằng ngôn ngữ tự nhiên cho các hệ thống phần mềm có sẵn.

+ Xây dựng hệ chuyên gia.

+ Viết các chương trình trí tuệ nhân tạo...

3.8.2. Các yếu tố cơ bản của Turbo Prolog

Vì Turbo Prolog là ngôn ngữ khai báo (ta phải mô tả hay khai báo thì chương trình mới suy diễn cho ra kết quả) nên trong chương trình ta phải khai báo các yếu tố sau:

- Các đối tượng (object)
- Các quan hệ giữa các đối tượng
- Các sự kiện (facts) và quy tắc (rules)

a. Các đối tượng: Bao gồm biến và hằng.

+ Các hằng có giá trị cho sẵn ở đầu chương trình hoặc trong quá trình viết chương trình ta đưa vào.

+ Các biến được gán giá trị khi chạy chương trình và có thể thay đổi giá trị của biến. Tên của các biến phải là một xâu ký tự hoa hoặc bắt đầu bằng ký tự hoa.

Biến tự do: Là biến chưa được gán giá trị.

Biến bị chặn: Là biến đã đạt giá trị sau khi ghép đúng với quy tắc hoặc sự kiện.

Biến vô danh: Là biến mà ta không cần để ý đến giá trị của nó và ta có thể thay nó bằng một biến bất kỳ. Biến vô danh thường được ký hiệu bằng một gạch dưới dòng. Ví dụ: likes(_, X)

Ví dụ: X, Y, Name,... là các tên biến đúng.

a=15, pi=3.1416,...là các giá trị hằng.

b. Quan hệ giữa các đối tượng: Các quan hệ được dùng dưới hình thức vị từ.

Ví dụ: likes(X, Y) là vị từ diễn tả câu "X likes Y" trong ngôn ngữ tự nhiên.

blue(car) là vị từ biểu diễn câu "car is blue" trong ngôn ngữ tự nhiên.

Các thành phần trong ngoặc của vị từ gọi là đối số của nó. Số các đối số của một vị từ gọi là arity của nó.

Chẳng hạn trong hai ví dụ trên thì vị từ likes có arity 2, vị từ blue có arity 1.

c. Các sự kiện và quy tắc:

+ Các sự kiện là các vị từ cho giá trị đúng.

+ Quy tắc là mệnh đề kéo theo cho giá trị đúng.

Các sự kiện và quy tắc được kê khai trong phần clauses.

Một số ví dụ chương trình Prolog:

Ví dụ:

```
clauses
        blue (bird)
        blue (car)
        likes (an, X): - blue (X)
```

có nghĩa là:

```
        bird is blue
        car is blue
        an likes X if X is blue
```

Ví dụ:

```
                                /*LIKES.PRO*/
domains
    anh, car, bird = symbol
predicates
    blue (symbol)
    likes (symbol, symbol)
clauses
    blue (symbol)
    blue (car)
    likes (anh, X) : -blue (X)
```

goal

likes (anh, X)

write (X), nl, fail

Trong chương trình này ta có dùng vị từ **fail** có arity 0 để yêu cầu máy quay lại và cho hai giải đáp: bird và car.

3.8.3. CÁC PHẦN CỦA MỘT CHƯƠNG TRÌNH

Một chương trình Turbo Prolog thường 3 hay 4 phần cơ bản: clauses, predicates, domains, và goal được bố cục như sau:

```
constans
   /* khai báo hằng*/
domain
   /*....khai báo kiểu các đối tượng .....*/
database
   /* khai báo cơ sở dữ liệu cần dùng*/
predicates
   /*khai báo các vị từ*/
goal
   /*khai báo các đích*/
clauses
   /*các sự kiện và quy tắc*/
```

+ ***Phần clauses***: Gồm các sự kiện và các quy tắc mà máy phải tuân theo để thoả mãn phần goal.

+ ***Phần predicates:*** Nếu trong phần clauses có sử dụng vị từ nào thì trong phần này phải khai báo (trừ các vị từ mà Turbo Prolog đã định nghĩa sẵn như write, n1, fail,...). Khi khai báo mỗi vị từ mới, phải liệt kê kiểu của các đối số. Ví dụ: is (symbol, symbol). Nếu trong phần domains đã khai báo kiểu của đối số thì ở đây chỉ cần viết tên của đối số.

+ ***Phần domains***: Phần này khai báo kiểu của các đối số trong các vị từ của chương trình. Nếu các vị từ được viết trực tiếp qua kiểu của đối số (chẳng hạn is(symbol, symbol) thì trong phần domains không cần khai báo.

+ ***Phần goal***: Nếu goal được viết sẵn trong chương trình thì nó được gọi là goal nội. Nếu trong chương trình không có goal thì khi chạy chương trình máy sẽ yêu cầu cho goal goal ngoại ở cửa sổ Dialog dạng: Goal: -

Khi đó ta đưa goal ngoại vào và ấn enter mới có kết quả.

Goal nội và goal ngoại khác nhau ở chỗ goal ngoại cho tất cả giải đáp còn goal nội khi đã đạt được một giải đáp rồi thì ngừng. Vì vậy khi dùng goal nội mà muốn có tất cả giải đáp thì ta dùng vị từ fail.

Ngoài ra trong chương trình Turbo Prolog còn có thể có các phần:

+ *Phần include*: Yêu cầu khi chạy, Turbo đưa thêm vào một tệp tin có sẵn trong ổ đĩa A:, B:, hoặc C:

Ví dụ: Include "A:\GRAPDECL.PRO"

+ *Phần Constans*: Cho tên và giá trị các hằng số

Ví dụ: pi=3.1416

ega=3

+ *Phần chỉ thị dịch:* Bao gồm các chỉ thị biên dịch:

- Trace: Yêu cầu chạy chương trình từng bước
- Nowarnings (không lưu ý): Yêu cầu không cho warnings khi một biến chỉ xuất hiện một lần.

Ngoài ra còn nhiều chỉ thị dịch khác.

3.8.4. CÁCH VIẾT MỘT SỐ CHƯƠNG TRÌNH ĐƠN GIẢN

Trước hết chúng ta phải biết cách biểu diễn một câu trong ngôn ngữ tự nhiên thành hình thức vị từ.

Ví dụ: A fast car is fun được viết thành fun (fast_car).

A big car is nice được viết thành nice (big_car).

Bill likes a car if the car is fun được viết thành likes (Bill, Car_) : - fun (car).

the car is a blue mercedes được viết thành car (mercedes, blue, station vagon)

Ví dụ 3.8.

/* PHONE.PRO */

```
domains
    name=symbol
    number= real
predicates
    phone (name, number)
    demo
clause
    phone (“Gs. Nhụy”, 843330).
    phone( “Gs. Xuân”, 840368).
    phone(“A. Cần”, 855668).
    phone(“A. Hào”, No): - phone(“A. Cần”, No).
    demo: -
    write(“ Gõ vào một tên:”)
    readln(name),
    phone (name, No),
    write(“ Số điện thoại của”, name,”là”, No), nl.
```

Ở đây ta dùng vị từ demo không có đối số. Chương trình không có goal nội. Goal ngoại là: demo, khi chạy chương trình sẽ thấy ở cửa sổ Dialog như sau:

Gõ vào một tên:

bạn gõ vào

Gs. Nhuỵ

và thấy

Số điện thoại của G.s Nhuỵ là 843330

Yes

Quy tắc phone (“A.Hào”, No): - phone (A.Cần, No) có nghĩa là số điện thoại của A.Hào là No, nếu số điện thoại của A.Cần là No, mà sự kiện phone(“A.Cần”, 855668) cho biết số điện thoại của A.Cần là 855668, nên khi gõ vào tên A. Hào thì cũng được: Số điện thoại của A.Hào là 445761

Các tên người được viết trong dấu ngoặc kép “...” để chỉ đó là một symbol.

No trong phone (“A.Dung”, No) là một biến nên N phải viết hoa.

Ví dụ 3.9.

```
/* EATS.PRO */
domains
    name, food=symbol
predicates
    eats (symbol, symbol)
clauses
    eats (“An”, banana).
    eats (“Ba”, apple).
    eats (“An”, apricot).
    eats (“Công”, grapes).
```

a. Nếu bạn cho Goal: **eats (X, banana), eats (X, apricot)** sẽ được giải đáp:

   X= An

   1 solution

b. Nếu bạn cho Goal: **eats (X, banana); eats (X, apple)** ta sẽ được giải đáp:

   X=An

   X=Ba

2 solutions

c. Nếu cho goal: **eats (X, banana), eats (grapes)** sẽ được giải đáp:

No solution

Trường hợp a, goal: **eats (X, banana), eats (X, apricot)** gồm hai subgoals nối liền nhau bằng dấu “,” tức là “và”, ta gọi đó là giao của hai subgoals, nó giống như mệnh đề giao trong đại số. Cả hai subgoals đều phải thoả nên ta có gải đáp X=An

Trường hợp b, goal ngoại gồm hai subgoals nối nhau bằng dấu “;” tức là “hay”, ta gọi đó là hợp của hai subgoals. Nó giống như mệnh đề hợp của hai mệnh đề, chỉ cần ít nhất một trong hai subgoals thoả và ta thấy eats(“An”, banana) hay (“Ba”, apple) thoả nên ta được cả hai giải đáp X=An hay X=Ba.

Trường hợp c, goal ngoại là giao của hai subgoals không đồng thời thoả được nên không có giải đáp:

No solution

Ví dụ 3.10.

```
/ * MAZE.PRO */
predicates
find_goal (integer, integer)
path (integer, integer)
clauses
    find_ goal (Start, End) : -
    path (Start, End),
    write (“Goal is”, End), nl.
find_ goal (Start, End) : -
    path (Start, G), !,
    write( “Move from”, Start,”to”, G), nl,
    find_ goal (G, End).
    path (3,6).
```

path (6,7).

path (7,8).

path (8,12).

path (12,16).

path (16, 15).

Bạn cho goal: find_ goal(6,16) thì được giải đáp:

Move from 6 to 7

Move from 7 to 8

Move from 8 to 12

Goal is 16

Yes

Đây là chương trình chỉ đường đi trong một mê cung (maze). Chúng ta có thể một số Goal khác.

3.8.4. CÁC KIỂU ĐỐI TƯỢNG

a. Các kiểu tiêu chuẩn

1. *char:* Kỹ tự viết trong cặp nháy đơn.
2. *integer:* Số nguyên từ -32768 đến 32767
3. *real:* Số thực dạng dấu chấm tĩnh hay động. Từ 0^{-307} đến 0^{307}.
4. *string:* Chuỗi ký tự viết trong nháy kép.
5. *symbol:* Có hai dạng:

   + Dãy các chữ, số và dấu gạch dưới viết liên tiếp, ký tự đầu là chữ thường.

   + Dãy ký tự viết trong cặp nháy kép.
6. *file:* Kiểu tệp
7. *Kiểu phức hợp:* Các đối tượng mà bản thân nó chứa các đối tượng khác.

Ví dụ: Muốn diễn tả Mary có một cuốn sách tên là English for Today ta viết:

owns(“Mary”, book(“English for Today”))

8. *Danh sách:* Danh sách giống như mảng trong Pascal nhưng cách viết lại giống như tập hợp. Ví dụ:

[1, 3, 5, 6] - Danh sách gồm các số 1,3, 5, 6.

[“Mary”, “John”] - Danh sách gồm xâu Mary, John.

[dog, cat, canary] - Danh sách các symbol dog, cat, canary.

[] - Danh sách rỗng.

Mỗi danh sách được xem gồm hai phần:

- Phần đầu (head): Là phần tử đầu tiên
- Phần đuôi (tail): Gồm các phần tử còn lại.

Để khai báo danh sách, trong domai ta viết kiểu_đối_tượng *

Ví dụ 3.11. Ví dụ về danh sách

```
/*HEADTAIL.PRO*/
domain
   ilist=integer *
predicates
   headtail(ilist)
clauses
   headtail([H | T]):-
   write(“The head is H=”, H), n1
   write(“The tail is T=”), T), n1
goal
   headtail([1,3,5,6])
```

Khi chạy chương trình ta thu được

H= 1

T=[3,5,6]

3.8.5. CÁC PHÉP TOÁN

Các phép toán số học

Toán hạng 1	*Toán tử*	*Toán hạng 2*	*Kết quả*
integer	+, -, *	integer	integer
integer	+, -, *	real	real
real	+, -, *	integer	real
real	+, -, *	real	real
integer or real	/	integer or real	real
integer	div	integer	integer
integer	mod	integer	ineger

Các phép so sánh: =, <, <=, >, >=, <>

Các hàm số học: Tương tự trong Pascal.

3.8.6. CÁCH NHẬP XUẤT

Lệnh xuất:

write(Arg1, Arg2, . . . ,ArgN)

trong đó Arg1, Arg2, . . . ,ArgN là các chuỗi, biến, hằng.

Muốn xuất có khuôn dạng ta dùng vị từ

write(Formatsting, Arg1, Arg2, . . . ,ArgN)

ở đây Formatstring để chỉ khuôn dạng.

Lệnh nhập:

Turbo Prolog dùng một số vị từ sau để nhập:

- Đọc cả một dòng ký tự.
- Đọc các giá trị số nguyên, số thực, các ký tự từ bàn phím.
- Đọc từ một tên tệp trên đĩa.

readln(Biến_dòng): Biến_dòng là biến có kiểu xâu hoặc kiểu ký tự.

readint(X): X là kiểu Integer.

readreal(X): X là biến kiểu real.

file_str(Tên_tệp, X): Miền dữ liệu cho Tên_tệp và X phải là symbol hoặc string.

readchar(Biến_ký_tự): Biến ký tự là kiểu char.

Ví dụ 3.12. Chương trình tính tổ hợp n chập r, với n và r nhập sau:

/* COMBINAT.PRO*/

```
predicates
        com (real, real, real)
        facto(real, real)
clauses
        com(N,R,C): - facto (N, M), facto( R, M1), L=N-R, facto( L, M2),
C=M/(M1*M2).
        facto(1,1).
        facto (N, M): -N>1,N1=N-1, facto (N1, M1), M=N*M1.
goal
        write("n = "), readreal(N),
        write("r=  "), readreal(R),
    com( N,R,C), write("C ("",n,"","",r,"") = "",C)
```

MỤC LỤC

TÀI LIỆU THAM KHẢO

1. Bạch Hưng Khang, Hoàng Kiếm. ***Trí tuệ nhân tạo: Các phương pháp và ứng dụng***. Nhà xuất bản khoa học kỹ thuật, 1989.
2. Nguyễn Thanh Thuỷ. ***Các phương pháp giải quyết vấn đề và kỹ thuật xử lý tri thức***. Nhà xuất bản giáo dục, 1996.
3. Đỗ Trung Tuấn. ***Trí tuệ nhân tạo***. Nhà xuất bản giáo dục, 1998.
4. John Durkin. ***Expert Systems***. Ed. Prentice Hall, 1996.
5. N. Nilson. ***Artificial Intelligence***. Ed. McGrawhill, 1971.
6. Viện khó học tính toán và điều khiển. ***Prolog và hệ chuyên gia***. 1998.
7. Phan Trương Dần. ***Lập trình Turbo Prolpog 2.0***. Nhà xuất bản khoa học kỹ thuật. 1997.

TRÍ TUỆ NHÂN TẠO

Hoàng Kiếm

CHƯƠNG 1 : THUẬT TOÁN – THUẬT GIẢI

I. TỔNG QUAN THUẬT TOÁN – THUẬT GIẢI

Trong quá trình nghiên cứu giải quyết các vấn đề – bài toán, người ta đã đưa ra những nhận xét như sau:

> Có nhiều bài toán cho đến nay vẫn chưa tìm ra một cách giải theo kiểu thuật toán và cũng không biết là có tồn tại thuật toán hay không.
>
> Có nhiều bài toán đã có thuật toán để giải nhưng không chấp nhận được vì thời gian giải theo thuật toán đó quá lớn hoặc các điều kiện cho thuật toán khó đáp ứng.
>
> Có những bài toán được giải theo những cách giải vi phạm thuật toán nhưng vẫn chấp nhận được.

Từ những nhận định trên, người ta thấy rằng cần phải có những đổi mới cho khái niệm thuật toán. Người ta đã mở rộng hai tiêu chuẩn của thuật toán: tính xác định và tính đúng đắn. Việc mở rộng tính xác định đối với thuật toán đã được thể hiện qua các giải thuật đệ quy và ngẫu nhiên. Tính đúng của thuật toán bây giờ không còn bắt buộc đối với một số cách giải bài toán, nhất là các cách giải gần đúng. Trong thực tiễn có nhiều trường hợp người ta chấp nhận các cách giải thường cho kết quả tốt (nhưng không phải lúc nào cũng tốt) nhưng ít phức tạp và hiệu quả. Chẳng hạn nếu giải một bài

toán bằng thuật toán tối ưu đòi hỏi máy tính thực hiên nhiều năm thì chúng ta có thể sẵn lòng chấp nhận một giải pháp gần tối ưu mà chỉ cần máy tính chạy trong vài ngày hoặc vài giờ.

Các cách giải chấp nhận được nhưng không hoàn toàn đáp ứng đầy đủ các tiêu chuẩn của thuật toán thường được gọi là các thuật giải. Khái niệm mở rộng này của thuật toán đã mở cửa cho chúng ta trong việc tìm kiếm phương pháp để giải quyết các bài toán được đặt ra.

Một trong những thuật giải thường được đề cập đến và sử dụng trong khoa học trí tuệ nhân tạo là các cách giải theo kiểu Heuristic.

II. THUẬT GIẢI HEURISTIC

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau:

Thường tìm được lời giải tốt (nhưng không chắc là lời giải tốt nhất)

Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.

Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ bản như sau:

Nguyên lý vét cạn thông minh: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.

Nguyên lý tham lam (Greedy): Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.

Nguyên lý thứ tự: Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.

Hàm Heuristic: Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các hàm Heuristic. Đó là các hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.

Bài toán hành trình ngắn nhất – ứng dụng nguyên lý Greedy

Bài toán: Hãy tìm một hành trình cho một người giao hàng đi qua n điểm khác nhau, mỗi điểm đi qua một lần và trở về điểm xuất phát sao cho tổng chiều dài đoạn đường cần đi là ngắn nhất. Giả sử rằng có con đường nối trực tiếp từ giữa hai điểm bất kỳ.

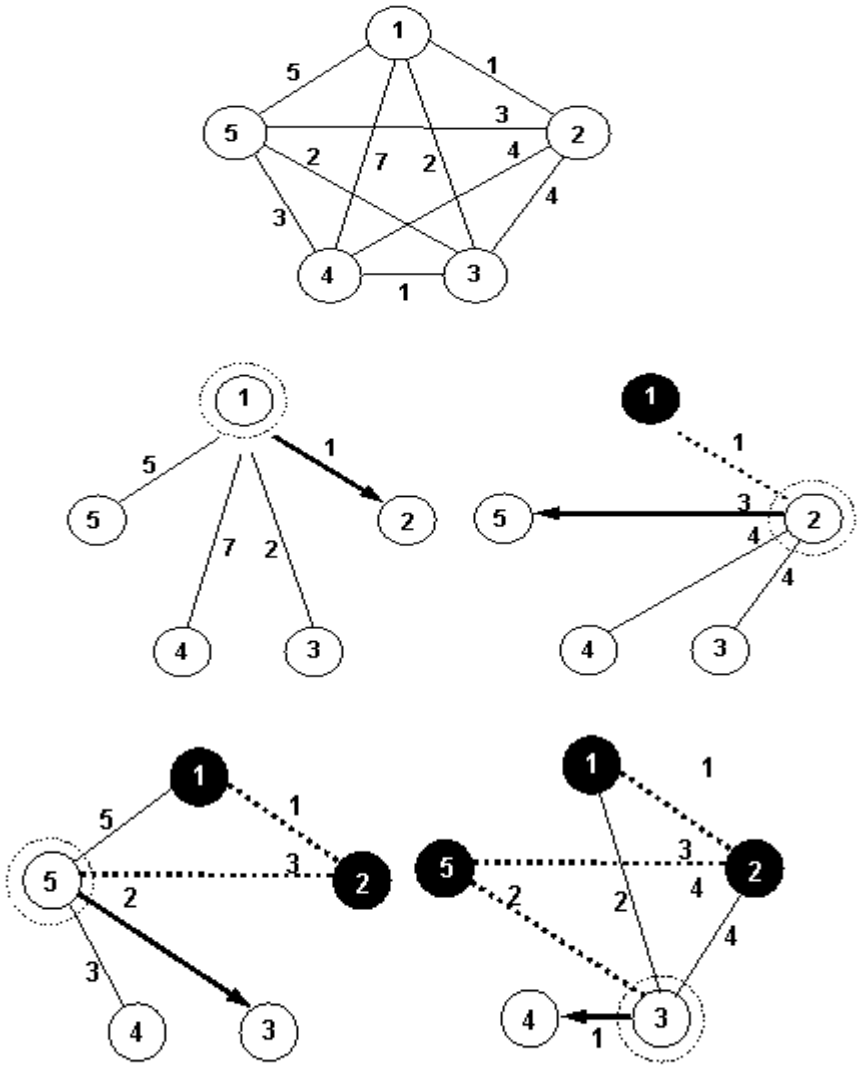
Tất nhiên ta có thể giải bài toán này bằng cách liệt kê tất cả con đường có thể đi, tính chiều dài của mỗi con đường đó rồi tìm con đường có chiều dài ngắn nhất. Tuy nhiên, cách giải này lại có độ phức tạp 0(n!) (một hành trình là một *hoán vị* của n điểm, do đó, tổng số hành trình là số lượng

hoán vị của một tập n phần tử là n!). Do đó, khi số đại lý tăng thì số con đường phải xét sẽ tăng lên rất nhanh.

Một cách giải đơn giản hơn nhiều và thường cho kết quả tương đối tốt là dùng một thuật giải Heuristic ứng dụng nguyên lý Greedy. Tư tưởng của thuật giải như sau:

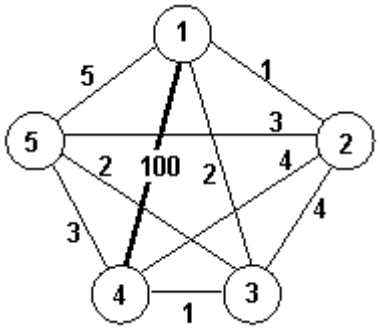
> Từ điểm khởi đầu, ta liệt kê tất cả quãng đường từ điểm xuất phát cho đến n đại lý rồi chọn đi theo con đường ngắn nhất.
>
> Khi đã đi đến một đại lý, chọn đi đến đại lý kế tiếp cũng theo nguyên tắc trên. Nghĩa là liệt kê tất cả con đường từ đại lý ta đang đứng đến những đại lý chưa đi đến. Chọn con đường ngắn nhất. Lặp lại quá trình này cho đến lúc không còn đại lý nào để đi.

Bạn có thể quan sát hình sau để thấy được quá trình chọn lựa. Theo nguyên lý Greedy, ta lấy tiêu chuẩn hành trình ngắn nhất của bài toán làm tiêu chuẩn cho chọn lựa cục bộ. *Ta hy vọng rằng, khi đi trên n đoạn đường ngắn nhất thì cuối cùng ta sẽ có một hành trình ngắn nhất.* Điều này không phải lúc nào cũng đúng. Với điều kiện trong hình tiếp theo thì thuật giải cho chúng ta một hành trình có chiều dài là 14 trong khi hành trình tối ưu là 13. Kết quả của thuật giải Heuristic trong trường hợp này chỉ lệch 1 đơn vị so với kết quả tối ưu. Trong khi đó, độ phức tạp của thuật giải Heuristic này chỉ là $0(n^2)$.



Hình : Giải bài toán sử dụng nguyên lý Greedy

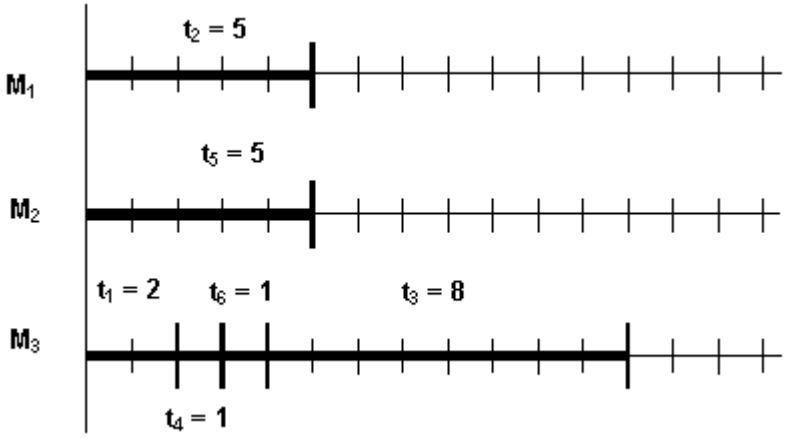
Tất nhiên, thuật giải theo kiểu Heuristic đôi lúc lại đưa ra kết quả không tốt, thậm chí rất tệ như trường hợp ở hình sau.



Bài toán phân việc – ứng dụng của nguyên lý thứ tự

Một công ty nhận được hợp đồng gia công m chi tiết máy J_1, J_2, … Jm. Công ty có n máy gia công lần lượt là P_1, P_2, … Pn. Mọi chi tiết đều có thể được gia công trên bất kỳ máy nào. Một khi đã gia công một chi tiết trên một máy, công việ sẽ tiếp tục cho đến lúc hoàn thành, không thể bị cắt ngang. Để gia công một việc J_1 trên một máy bất kỳ ta cần dùng một thời gian tương ứng là t_1. Nhiệm vụ của công ty là phải làm sao gia công xong toàn bộ n chi tiết trong thời gian sớm nhất.

Chúng ta xét bài toán trong trường hợp có 3 máy P_1, P_2, P_3 và 6 công việc với thời gian là t_1=2, t_2=5, t_3=8, t_4=1, t_5=5, t_6=1. ta có một phương án phân công (L) như hình sau:



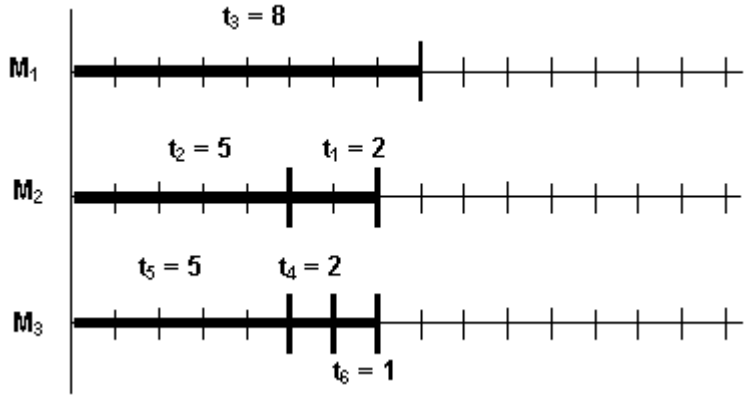
Theo hình này, tại thời điểm t=0, ta tiến hành gia công chi tiết J_2 trên máy P_1, J_5 trên P_2 và J_1 tại P_3. Tại thời điểm t=2, công việc J_1 được hoàn thành, trên máy P_3 ta gia công tiếp chi tiết J_4. Trong lúc đó, hai máy P_1 và P2 vẫn đang thực hiện công việc đầu tiên mình … Sơ đồ phân việc theo hình ở trên được gọi là lược đồ GANTT. Theo lược đồ này, ta thấy thời gian để hoàn thành toàn bộ 6 công việc là 12. Nhận xét một cách cảm tính ta thấy rằng phương án (L) vừa thực hiện là một phương án không tốt. Các máy P_1 và P_2 có quá nhiều thời gian rãnh.

Thuật toán tìm phương án tối ưu L_0 cho bài toán này theo kiểu vét cạn có độ phức tạp cỡ O(mn) (với m là số máy và n là số công việc). Bây giờ ta xét đến một thuật giải Heuristic rất đơn giản (độ phức tạp O(n)) để giải bài toán này.

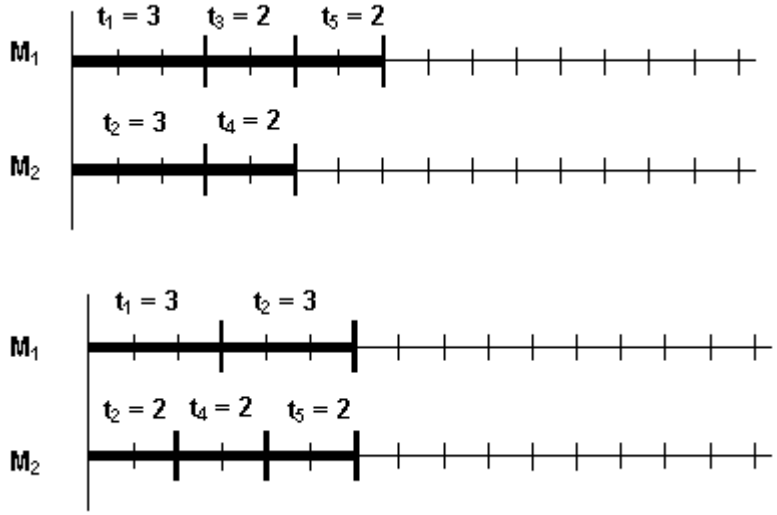
Sắp xếp các công việc theo thứ tự giảm dần về thời gian gia công.

Lần lượt sắp xếp các việc theo thứ tự đó vào máy còn dư nhiều thời gian nhất.

Với tư tưởng như vậy, ta sẽ có một phương án L* như sau:



Rõ ràng phương án L* vừa thực hiện cũng chính là phương án tối ưu của trường hợp này vì thời gian hoàn thành là 8, đúng bằng thời gian của công việc J_3. Ta hy vọng rằng một giải Heuristic đơn giản như vậy sẽ là một thuật giải tối ưu. Nhưng tiếc thay, ta dễ dàng đưa ra được một trường hợp mà thuật giải Heuristic không đưa ra được kết quả tối ưu.



Nếu gọi T* là thời gian để gia công xong n chi tiết máy do thuật giải Heuristic đưa ra và T^0 là thời gian tối ưu thì người ta đã chứng minh được rằng

$$\frac{T^*}{T^0} \le \frac{4}{3} - \frac{1}{M}$$, M là số máy

Với kết quả này, ta có thể xác lập được sai số mà chúng ta phải gánh chịu nếu dùng Heuristic thay vì tìm một lời giải tối ưu. Chẳng hạn với số máy là 2 (M=2) ta có $\frac{T^*}{T^0} \le \frac{7}{6}$, và đó chính là sai số cực đại mà trường hợp ở trên đã gánh chịu. Theo công thức này, số máy càng lớn thì sai số càng lớn.

Trong trường hợp M lớn thì tỷ số 1/M xem như bằng 0 . Như vậy, sai số tối đa mà ta phải chịu là T* [redacted]4/3 T^0, nghĩa là sai số tối đa là 33%. Tuy nhiên, khó tìm ra được những trường hợp mà sai số đúng bằng giá trị cực đại, dù trong trường hợp xấu nhất. Thuật giải Heuristic trong trường hợp này rõ ràng đã cho chúng ta những lời giải tương đối tốt.

III. CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC

Qua các phần trước chúng ta tìm hiểu tổng quan về ý tưởng của thuật giải Heuristic (nguyên lý Greedy và sắp thứ tự). Trong mục này, chúng ta sẽ đi sâu vào tìm hiểu một số kỹ thuật tìm kiếm Heuristic – một lớp bài toán rất quan trọng và có nhiều ứng dụng trong thực tế.

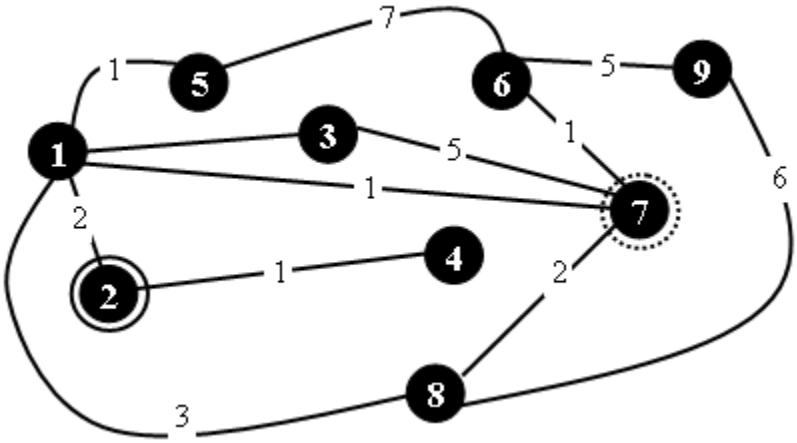
III.1. Cấu trúc chung của bài toán tìm kiếm

Để tiện lợi cho việc trình bày, ta hãy dành chút thời gian để làm rõ hơn "đối tượng" quan tâm của chúng ta trong mục này. Một cách chung nhất, nhiều vấn đề-bài toán phức tạp đều có dạng "tìm đường đi trong đồ thị" hay nói một cách hình thức hơn là *"xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến một đỉnh nào đó"*. Một phát biểu khác thường gặp của dạng bài toán này là :

Cho trước hai trạng thái T_0 và TG hãy xây dựng chuỗi trạng thái T_0, **T_1, T_2, ..., Tn_{-1}**, Tn = TG sao cho :

$\sum_1^n cost(T_{i-1}, T_i)$ thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).

Trong đó, Ti thuộc tập hợp S (gọi là không gian trạng thái – state space) bao gồm tất cả các trạng thái có thể có của bài toán và **cost(T_{i-1}, T_i)** là *chi phí* để *biến đổi* từ trạng thái Ti_{-1} sang trạng thái Ti. Dĩ nhiên, từ một trạng thái Ti ta có nhiều cách để biến đổi sang trạng thái Ti_{+1}. Khi nói đến một biến đổi cụ thể từ Ti_{-1} sang Ti ta sẽ dùng thuật ngữ *hướng đi (*với ngụ ý nói về sự lựa chọn).



Hình : Mô hình chung của các vấn đề-bài toán phải giải quyết bằng phương pháp tìm kiếm lời giải. Không gian tìm kiếm là một tập hợp trạng thái - tập các nút của đồ thị. Chi phí cần thiết để chuyển từ trạng thái T này sang trạng thái Tk được biểu diễn dưới dạng các con số nằm trên cung nối giữa hai nút tượng trưng cho hai trạng thái.

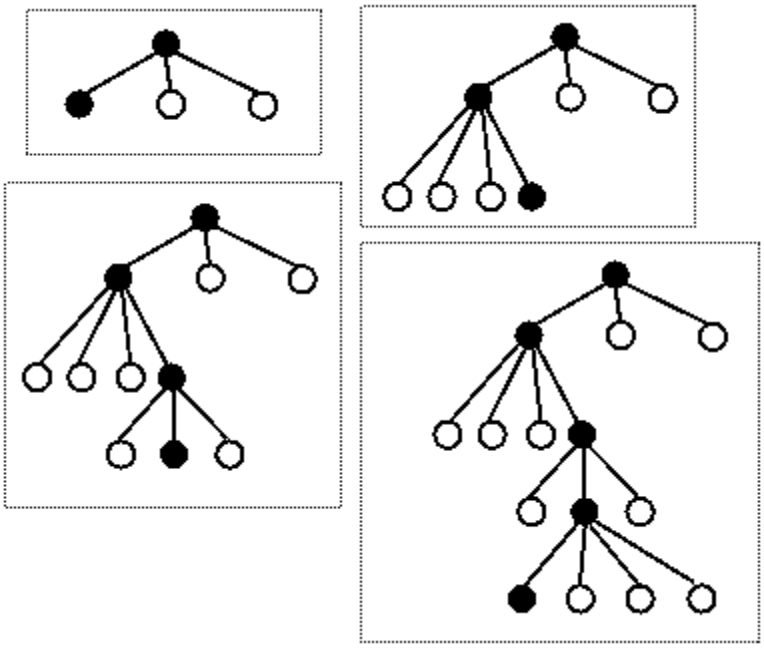
Đa số các bài toán thuộc dạng mà chúng ta đang mô tả đều có thể được biểu diễn dưới dạng đồ thị. Trong đó, một trạng thái là một đỉnh của đồ thị. Tập hợp S bao gồm tất cả các trạng thái chính là tập hợp bao gồm tất cả đỉnh của đồ thị. Việc biến đổi từ trạng thái Ti_{-1} sang trạng thái Ti là việc đi từ đỉnh đại diện cho Ti_{-1} sang đỉnh đại diện cho Ti theo cung nối giữa hai đỉnh này.

III.2. Tìm kiếm chiều sâu và tìm kiếm chiều rộng

Để bạn đọc có thể hình dung một cách cụ thể bản chất của thuật giải Heuristic, chúng ta nhất thiết phải nắm vững hai *chiến lược* tìm kiếm cơ bản là tìm kiếm theo chiều sâu (Depth First Search) và tìm kiếm theo chiều rộng (Breath First Search). Sở dĩ chúng ta dùng từ *chiến lược* mà không phải là *phương pháp* là bởi vì trong thực tế, người ta hầu như chẳng bao giờ vận dụng một trong hai kiểm tìm kiếm này một cách trực tiếp mà không phải sửa đổi gì.

III.2.1. Tìm kiếm chiều sâu (Depth-First Search)

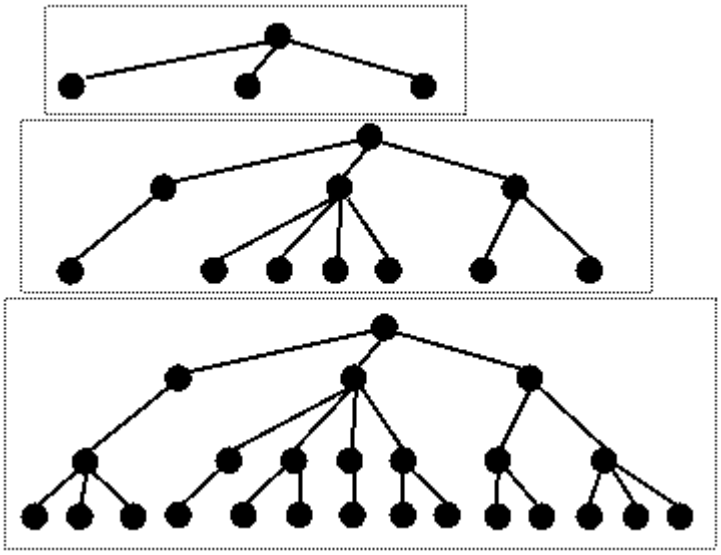
Trong tìm kiếm theo chiều sâu, tại trạng thái (đỉnh) hiện hành, ta chọn một trạng thái kế tiếp (trong tập các trạng thái có thể biến đổi thành từ trạng thái hiện tại) làm trạng thái hiện hành cho đến lúc trạng thái hiện hành là trạng thái đích. Trong trường hợp tại trạng thái hiện hành, ta không thể biến đổi thành trạng thái kế tiếp thì ta sẽ quay lui (back-tracking) lại trạng thái trước trạng thái hiện hành (trạng thái biến đổi thành trạng thái hiện hành) để chọn đường khác. Nếu ở trạng thái trước này mà cũng không thể biến đổi được nữa thì ta quay lui lại trạng thái trước nữa và cứ thế. Nếu đã quay lui đến trạng thái khởi đầu mà vẫn thất bại thì kết luận là không có lời giải. Hình ảnh sau minh họa hoạt động của tìm kiếm theo chiều sâu.



Hình : Hình ảnh của tìm kiếm chiều sâu. Nó chỉ lưu ý "mở rộng" trạng thái được chọn mà không "mở rộng" các trạng thái khác (nút màu trắng trong hình vẽ).

III.2.2. Tìm kiếm chiều rộng (Breath-First Search)

Ngược lại với tìm kiếm theo kiểu chiều sâu, tìm kiếm chiều rộng mang hình ảnh của vết dầu loang. Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, *ứng với mỗi* trạng thái Tk trong tập S, ta xây dựng tập Sk bao gồm các trạng thái kế tiếp của Tk rồi lần lượt bổ sung các Sk vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả Sk.



Hình : Hình ảnh của tìm kiếm chiều rộng. Tại một bước, mọi trạng thái đều được mở rộng, không bỏ sót trạng thái nào.

	Chiều sâu	Chiều rộng
Tính hiệu quả	Hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác. Hiệu quả của chiến lược phụ thuộc vào phương án chọn hướng đi. Phương án càng kém hiệu quả thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm chỉ một lời giải.	Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm. Hiệu quả của chiến lược phụ thuộc vào độ sâu của lời giải. Lời giải càng xa gốc thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm nhiều lời giải.
Lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Chỉ lưu lại các trạng thái chưa xét đến.	Phải lưu toàn bộ các trạng thái.
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ.
Trường hợp tốt nhất	Phương án chọn hướng đi *tuyệt đối* chính xác. Lời giải được xác định một cách trực tiếp.	Vét cạn toàn bộ.

Tìm kiếm chiều sâu và tìm kiếm chiều rộng đều là các phương pháp tìm kiếm có hệ thống và chắc chắn tìm ra lời giải. Tuy nhiên, do bản chất là vét cạn nên với những bài toán có không gian lớn thì ta không thể dùng hai chiến lược này được. Hơn nữa, hai chiến lược này đều có tính chất "mù quáng" vì chúng không chú ý đến những thông tin (tri thức) ở trạng thái hiện thời và thông tin về đích cần đạt tới cùng mối quan hệ giữa chúng. Các tri thức này vô cùng quan trọng và rất có ý nghĩa để thiết kế các thuật giải hiệu quả hơn mà ta sắp sửa bàn đến.

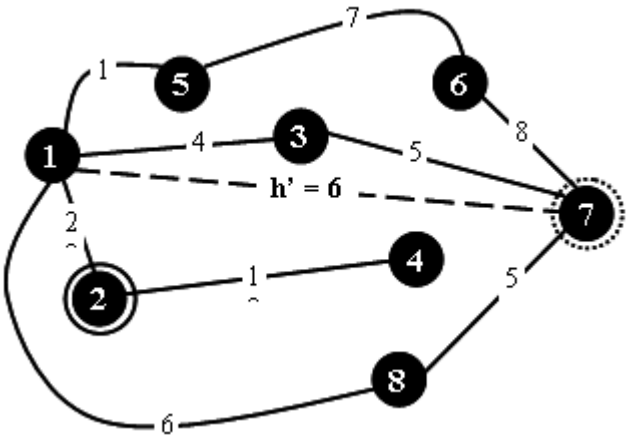
III.3. Tìm kiếm leo đồi

III.3.1. Leo đồi đơn giản

Tìm kiếm leo đồi theo đúng nghĩa, nói chung, thực chất chỉ là một trường hợp đặc biệt của tìm kiếm theo chiều sâu nhưng không thể quay lui. Trong tìm kiếm leo đồi, việc lựa chọn trạng thái tiếp theo được quyết định dựa trên một hàm Heuristic.

Hàm Heuristic là gì ?

Thuật ngữ "hàm Heuristic" muốn nói lên điều gì? Chẳng có gì ghê gớm. Bạn đã quen với nó rồi! Đó đơn giản chỉ là một ***ước lượng về khả năng dẫn đến lời giải*** tính từ trạng thái đó (*khoảng cách* giữa trạng thái hiện tại và trạng thái đích). Ta sẽ quy ước gọi hàm này là **h** trong suốt giáo trình này. Đôi lúc ta cũng đề cập đến ***chi phí tối ưu thực sự*** từ một trạng thái dẫn đến lời giải. Thông thường, giá trị này là không thể tính toán được (vì tính được đồng nghĩa là đã biết con đường đến lời giải !) mà ta chỉ dùng nó như một cơ sở để suy luận về mặt lý thuyết mà thôi ! Hàm **h**, ta quy ước rằng, luôn trả ra kết quả là một số không âm. Để bạn đọc thực sự nắm được ý nghĩa của hai hàm này, hãy quan sát hình sau trong đó minh họa chi phí tối ưu thực sự và chi phí ước lượng.



Hình Chi phí ước lượng h' = 6 và chi phí tối ưu thực sự h = 4+5 = 9 (đi theo đường 1-3-7)

Bạn đang ở trong một thành phố xa lạ mà không có bản đồ trong tay và ta muốn đi vào khu trung tâm? Một cách suy nghĩ đơn giản, chúng ta sẽ nhắm vào *hướng* những tòa cao ốc của khu trung tâm!

Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T_0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi không tồn tại một trạng thái tiếp theo hợp lệ (Tk) của trạng thái hiện hành :

a. Đặt Tk là một trạng thái tiếp theo hợp lệ của trạng thái hiện hành Ti.

b. Đánh giá trạng thái Tk mới :

b.1. Nếu là trạng thái kết thúc thì trả về trị này và thoát.

b.2. Nếu không phải là trạng thái kết thúc nhưng **tốt** hơn trạng thái hiện hành thì cập nhật nó thành trạng thái hiện hành.

b.3. Nếu nó không tốt hơn trạng thái hiện hành thì tiếp tục vòng lặp.

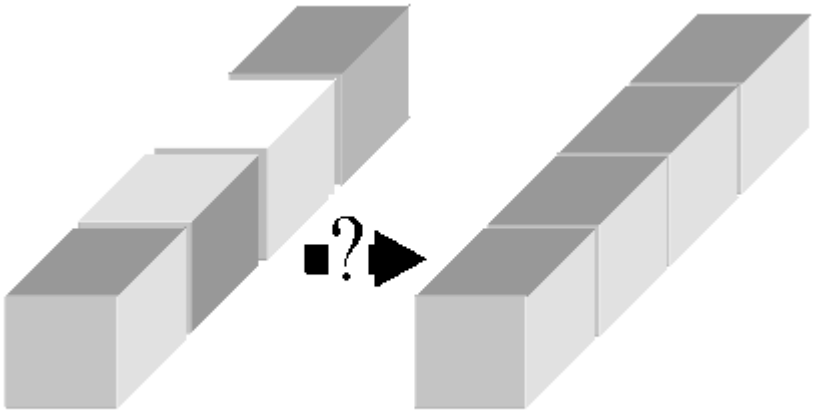
Mã giả

```
Ti := T0; Stop :=FALSE;

WHILE Stop=FALSE DO BEGIN

    IF Ti [redacted]TG THEN BEGIN

                <tìm được kết quả >; Stop:=TRUE;

    END;

    ELSE BEGIN

        Better:=FALSE;

        WHILE (Better=FALSE) AND
        (STOP=FALSE) DO BEGIN

            IF <không tồn tại trạng thái kế tiếp hợp lệ của
            Ti> THEN BEGIN

                <không tìm được kết quả >;
                Stop:=TRUE; END;

            ELSE BEGIN

                Tk := <một trạng thái kế tiếp hợp lệ
                của Ti>;

                IF <h(Tk) tốt hơn h(Ti)> THEN
                BEGIN

                    Ti :=Tk; Better:=TRUE;

                END;

            END;

        END; {WHILE}

    END; {ELSE}

END;{WHILE}
```

Mệnh đề "h'(Tk) tốt hơn h'(Ti)" nghĩa là gì? Đây là một khái niệm chung chung. Khi cài đặt thuật giải, ta phải cung cấp một định nghĩa tường minh về *tốt hơn.* Trong một số trường hợp, tốt hơn là nhỏ hơn : h'(Tk) < h'(Ti); một số trường hợp khác tốt hơn là lớn hơn h'(Tk) > h'(Ti)...Chẳng hạn, đối với bài toán tìm đường đi ngắn nhất giữa hai điểm. Nếu dùng hàm h' là hàm cho ra *khoảng cách theo đường chim bay* giữa vị trí hiện tại (trạng thái hiện tại) và đích đến (trạng thái đích) thì tốt hơn nghĩa là nhỏ hơn.

Vấn đề cần làm rõ kế tiếp là thế nào là <một trạng thái kế tiếp hợp lệ của Ti>? Một trạng thái kế tiếp hợp lệ là trạng thái chưa được xét đến. Giả sử h của trạng thái hiện tại Ti có giá trị là h(Ti) = 1.23 và từ Ti ta có thể biến đổi sang một trong 3 trạng thái kế tiếp lần lượt là Tk_1, Tk_2, Tk_3 với giá trị các hàm h tương ứng là $h(Tk_1) = 1.67$, $h(Tk_2) = 2.52$, $h'(Tk_3) = 1.04$. Đầu tiên, Tk sẽ được gán bằng Tk_1, nhưng vì $h'(Tk) = h'(T_{k1}) > h'(Ti)$ nên Tk không được chọn. Kế tiếp là Tk sẽ được gán bằng Tk_2 và cũng không được chọn. Cuối cùng thì Tk_3 được chọn. Nhưng giả sử $h'(Tk_3) = 1.3$ thì cả Tk_3 cũng không được chọn và mệnh đề <*không thể sinh ra trạng thái kế tiếp của Ti*> sẽ có giá trị TRUE. Giải thích này có vẻ hiển nhiên nhưng có lẽ cần thiết để tránh nhầm lẫn cho bạn đọc.

Để thấy rõ hoạt động của thuật giải leo đồi. Ta hãy xét một bài toán minh họa sau. Cho 4 khối lập phương *giống nhau* A, B, C, D. Trong đó các mặt (M1), (M2), (M3), (M4), (M5), (M6) có thể được tô bằng 1 trong 6 màu (1), (2), (3), (4), (5), (6). Ban đầu các khối lập phương được xếp vào một hàng. Mỗi một bước, ta chỉ được xoay một khối lập phương quanh một trục (X,Y,Z) 90^0 theo chiều bất kỳ (nghĩa là ngược chiều hay thuận chiều kim đồng hồ cũng được). Hãy xác định số bước quay ít nhất sao cho tất cả các mặt của khối lập phương trên 4 mặt của hàng là có cùng màu như hình vẽ.



Hình : Bài toán 4 khối lập phương

Để giải quyết vấn đề, trước hết ta cần định nghĩa một hàm **G** dùng để đánh giá một tình trạng cụ thể có phải là lời giải hay không? Bạn đọc có thể dễ dàng đưa ra một cài đặt của hàm G như sau :

IF (Gtrái + Gphải + Gtrên + Gdưới + Gtrước + Gsau) = 16 **THEN**

G:=TRUE

ELSE

G:=FALSE;

Trong đó, Gphải là *số lượng các mặt* có cùng màu của mặt bên phải của hàng. Tương tự cho Gtrái, Gtrên, Ggiữa, Gtrước, Gsau. Tuy nhiên, do các khối lập phương A,B,C,D là hoàn toàn tương tự nhau nên tương quan giữa các mặt của mỗi khối là giống nhau. Do đó, nếu có 2 mặt không đối nhau trên hàng đồng màu thì 4 mặt còn lại của hàng cũng đồng màu. Từ đó ta chỉ cần hàm G được định nghĩa như sau là đủ :

IF Gphải + Gdưới = 8 **THEN**

G:=TRUE

ELSE

G:=FALSE;

Hàm **h** (ước lượng khả năng dẫn đến lời giải của một trạng thái) sẽ được định nghĩa như sau :

h = Gtrái + Gphải + Gtrên + Gdưới

Bài toán này đủ đơn giản để thuật giải leo đồi có thể hoạt động tốt. Tuy nhiên, không phải lúc nào ta cũng may mắn như thế!

Đến đây, có thể chúng ta sẽ nảy sinh một ý tưởng. Nếu đã chọn trạng thái *tốt hơn* làm trạng thái hiện tại thì tại sao không chọn trạng thái *tốt nhất* ? Như vậy, *có lẽ* ta sẽ nhanh chóng dẫn đến lời giải hơn! Ta sẽ bàn luận về vấn đề: "liệu cải tiến này có thực sự giúp chúng ta dẫn đến lời giải nhanh hơn hay không?" ngay sau khi trình bày xong thuật giải leo đồi dốc đứng.

III.3.2. Leo đồi dốc đứng

Về cơ bản, leo đồi dốc đứng cũng giống như leo đồi, chỉ khác ở điểm là leo đồi dốc đứng sẽ duyệt tất cả các hướng đi có thể và chọn đi theo trạng thái *tốt nhất* trong số các trạng thái kế tiếp có thể có (trong khi đó leo đồi chỉ chọn đi theo trạng thái kế tiếp đầu tiên *tốt hơn* trạng thái hiện hành mà nó tìm thấy).

Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T_0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi (Ti) không tồn tại một trạng thái kế tiếp (Tk) nào tốt hơn trạng thái hiện tại (Ti)

a) Đặt S bằng tập tất cả trạng thái kế tiếp có thể có của T_i *và* tốt hơn Ti.

b) Xác định Tkmax là trạng thái tốt nhất trong tập S

Đặt Ti = Tkmax

Mã giả

```
Ti := T0;
Stop :=FALSE;
WHILE Stop=FALSE DO BEGIN
    IF Ti [redacted]TG THEN BEGIN
        <tìm được kết quả >;
        STOP :=TRUE;
        END;
```

```
ELSE BEGIN
        Best:=h'(Ti);
        Tmax := Ti;
        WHILE <tồn tại trạng thái kế tiếp hợp lệ của Ti> DO
        BEGIN
            Tk := <một trạng thái kế tiếp hợp lệ của Ti>;
            IF <h'(Tk) tốt hơn Best> THEN BEGIN
                Best :=h'(Tk);
                Tmax := Tk;
            END;
        END;
        IF (Best>Ti) THEN
            Ti := Tmax;
        ELSE BEGIN
            <không tìm được kết quả >;
            STOP:=TRUE;
        END;
    END; {ELSE IF}
END;{WHILE STOP}
```

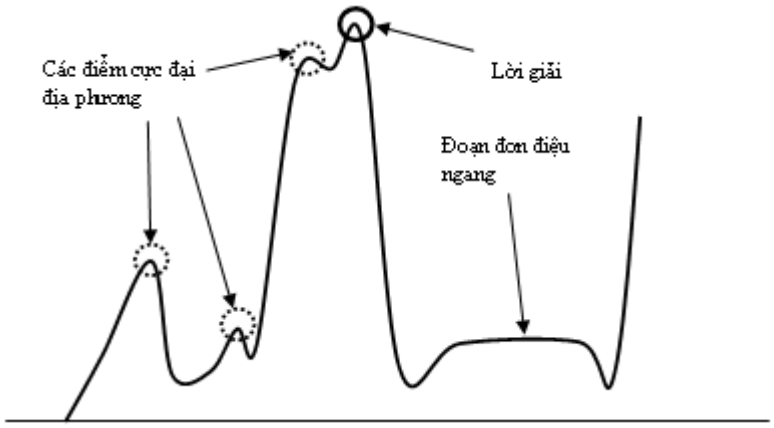
III.3.3. Đánh giá

So với leo đồi đơn giản, leo đồi dốc đứng có ưu điểm là luôn luôn chọn hướng có triển vọng nhất để đi. Liệu điều này có đảm bảo leo đồi dốc đứng luôn tốt hơn leo đồi đơn giản không? Câu trả lời là không. Leo đồi dốc đứng chỉ tốt hơn leo đồi đơn giản trong một số trường hợp mà thôi. Để chọn ra được hướng đi tốt nhất, leo đồi dốc đứng phải duyệt qua *tất cả* các hướng đi có thể có tại trạng thái hiện hành. Trong khi đó, leo đồi đơn giản chỉ chọn đi theo trạng thái *đầu tiên* tốt hơn (so với trạng thái hiện hành) mà nó tìm ra được. Do đó, thời gian cần thiết để leo đồi dốc đứng chọn được một hướng đi sẽ lớn hơn so với leo đồi đơn giản. Tuy vậy, do lúc nào cũng chọn hướng đi tốt nhất nên leo đồi dốc đứng thường sẽ tìm đến lời giải sau một số bước ít hơn so với leo đồi đơn giản. Nói một cách ngắn gọn, leo đồi dốc đứng sẽ tốn nhiều thời gian hơn cho một bước nhưng lại đi ít bước hơn; còn leo đồi đơn giản tốn ít thời gian hơn cho một bước đi nhưng lại phải đi nhiều bước hơn. Đây chính là yếu tố được và mất giữa hai thuật giải nên ta phải cân nhắc kỹ lưỡng khi lựa chọn thuật giải.

Cả hai phương pháp leo núi đơn giản và leo núi dốc đứng đều có khả năng thất bại trong việc tìm lời giải của bài toán mặc dù lời giải đó thực sự hiện hữu. Cả hai giải thuật đều có thể kết thúc khi đạt được một trạng thái mà không còn trạng thái nào tốt hơn nữa có thể phát sinh nhưng trạng thái này không phải là trạng thái đích. Điều này sẽ xảy ra nếu chương trình đạt đến một điểm cực đại địa phương, một đoạn đơn điệu ngang.

Điểm cực đại địa phương (a local maximum) : là một trạng thái tốt hơn tất cả lân cận của nó nhưng không tốt hơn một số trạng thái khác ở xa hơn. Nghĩa là tại một điểm cực đại địa phương, mọi trạng thái *trong một lân cận* của trạng thái hiện tại đều *xấu hơn* trạng thái hiện tại. Tuy có dáng vẻ của lời giải nhưng các cực đại địa phương không phải là lời giải thực sự. Trong trường hợp này, chúng được gọi là những ngọn đồi thấp.

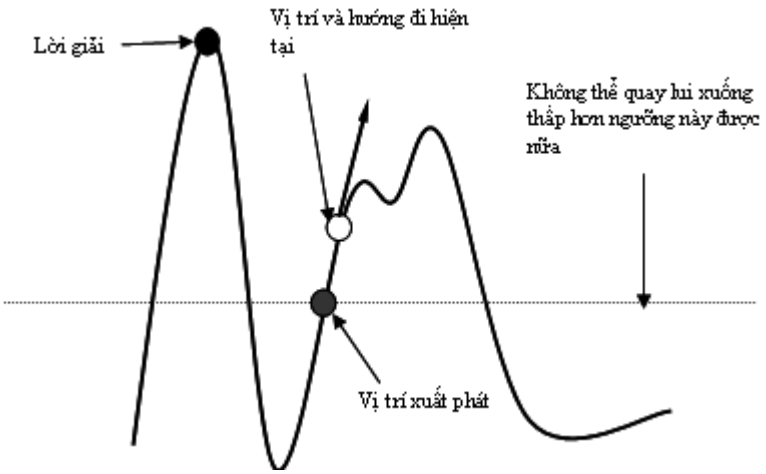
Đoạn đơn điệu ngang (a plateau) : là một vùng bằng phẳng của không gian tìm kiếm, trong đó, toàn bộ các trạng thái lân cận đều có cùng giá trị.



Hình : Các tình huống khó khăn cho tìm kiếm leo đèo.

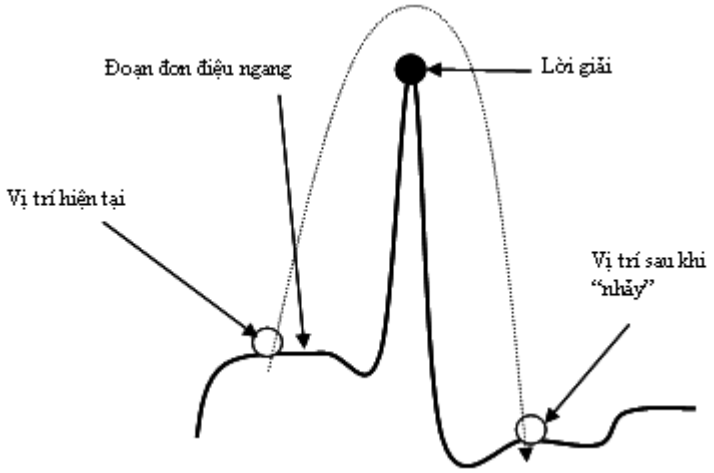
Để đối phó với các các điểm này, người ta đã đưa ra một số giải pháp. Ta sẽ tìm hiểu 2 trong số các giải pháp này. Những giải này, không thực sự giải quyết trọn vẹn vấn đề mà chỉ là một phương án cứu nguy tạm thời mà thôi.

Phương án đầu tiên là kết hợp leo đồi và quay lui. Ta sẽ quay lui lại các trạng thái trước đó và thử đi theo hướng khác. Thao tác này hợp lý nếu tại các trạng thái trước đó có một hướng đi tốt mà ta đã bỏ qua trước đó. Đây là một cách khá hay để đối phó với các điểm cực đại địa phương. Tuy nhiên, do đặc điểm của leo đồi là "bước sau cao hơn bước trước" nên phương án này sẽ thất bại khi ta xuất phát từ một điểm quá cao hoặc xuất phát từ một đỉnh đồi mà để đến được lời giải cần phải đi qua một "thung lũng" thật sâu như trong hình sau.



Hình : Một trường hợp thất bại của leo đèo kết hợp quay lui.

Cách thứ hai là thực hiện một bước *nhảy vọt* theo hướng nào đó để thử đến một vùng mới của không gian tìm kiếm. Nôm na là "bước" liên tục nhiều "bước" (chẳng hạn 5,7,10, …) mà tạm thời "quên" đi việc kiểm tra "bước sau cao hơn bước trước". Tiếp cận có vẻ hiệu quả khi ta gặp phải một đoạn đơn điệu ngang. Tuy nhiên, nhảy vọt cũng có nghĩa là ta đã bỏ qua cơ hội để tiến đến lời giải thực sự. Trong trường hợp chúng ta đang đứng khá gần lời giải, việc nhảy vọt sẽ đưa chúng ta sang một vị trí hoàn toàn xa lạ, mà từ đó, có thể sẽ dẫn chúng ta đến một rắc rối kiểu khác. Hơn nữa, số bước nhảy là bao nhiêu và nhảy theo hướng nào là một vấn đề phụ thuộc rất nhiều vào đặc điểm không gian tìm kiếm của bài toán.



Hình Một trường hợp khó khăn cho phương án "nhảy vọt".

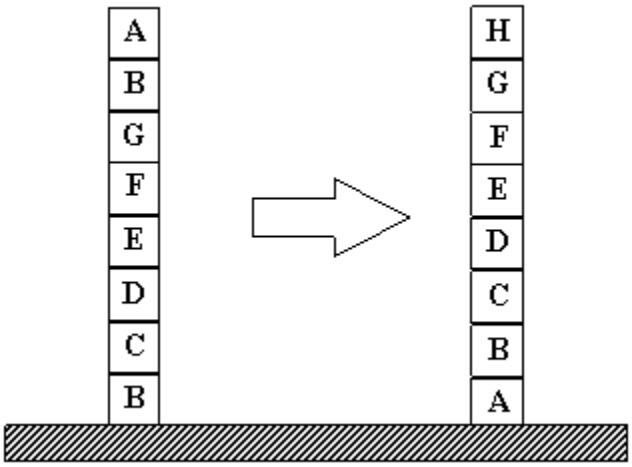
Leo núi là một phương pháp cục bộ bởi vì nó quyết định sẽ làm gì tiếp theo dựa vào một đánh giá về trạng thái hiện tại và các trạng thái kế tiếp có thể có (*tốt hơn* trạng thái hiện tại, trạng thái *tốt nhất* tốt hơn trạng thái hiện tại) thay vì phải xem xét một cách toàn diện trên tất cả các trạng thái đã đi qua. Thuận lợi của leo núi là ít gặp sự bùng nổ tổ hợp hơn so với các phương pháp toàn cục. Nhưng nó cũng giống như các phương pháp cục bộ khác ở chỗ là không chắc chắn tìm ra lời giải trong trường hợp xấu nhất.

Một lần nữa, ta khẳng định lại vai trò quyết định của hàm Heuristic trong quá trình tìm kiếm lời giải. Với cùng một thuật giải (như leo đồi chẳng hạn), nếu ta có một hàm Heuristic tốt hơn thì kết quả sẽ được tìm thấy nhanh hơn. Ta hãy xét bài toán về các khối được trình bày ở hình sau. Ta có hai thao tác biến đổi là:

+ Lấy một khối ở đỉnh một cột bất kỳ và đặt nó lên một chỗ trống tạo thành một cột mới. Lưu ý là chỉ có thể tạo ra tối đa 2 cột mới.

+ Lấy một khối ở đỉnh một cột và đặt nó lên đỉnh một cột khác

Hãy xác định số thao tác ít nhất để biến đổi cột đã cho thành cột kết quả.



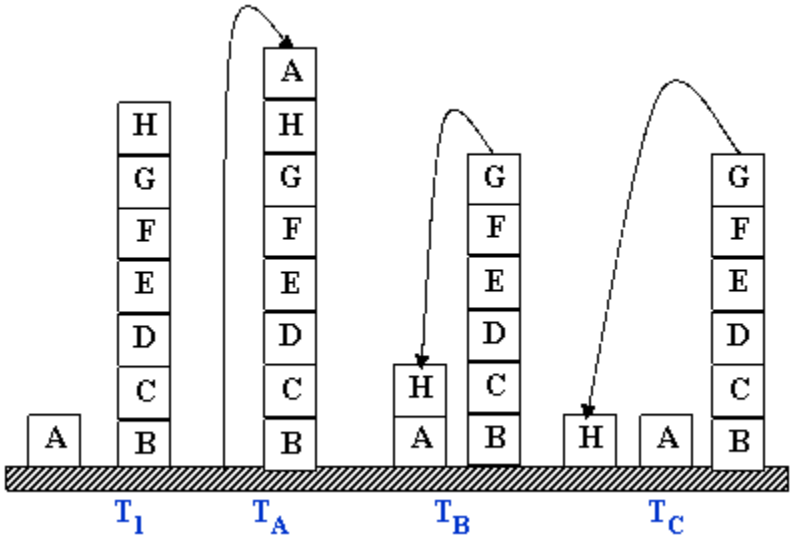
Hình : Trạng thái khởi đầu và trạng thái kết thúc

Giả sử ban đầu ta dùng một hàm Heuristic đơn giản như sau :

> **H_1** : Cộng 1 điểm cho mỗi khối ở vị trí đúng so với trạng thái đích. Trừ 1 điểm cho mỗi khối đặt ở vị trí sai so với trạng thái đích.

Dùng hàm này, trạng thái kết thúc sẽ có giá trị là 8 vì cả 8 khối đều được đặt ở vị trí đúng. Trạng thái khởi đầu có giá trị là 4 (vì nó có 1 điểm cộng cho các khối C, D, E, F, G, H và 1 điểm trừ cho các khối A và B). Chỉ có thể có một di chuyển từ trạng thái khởi đầu, đó là dịch chuyển khối A xuống tạo thành một cột mới (T_1).

Điều đó sinh ra một trạng thái với số điểm là **6** (vì vị trí của khối A bây giờ sinh ra 1 điểm cộng hơn là một điểm trừ). Thủ tục leo núi sẽ chấp nhận sự dịch chuyển đó. Từ trạng thái mới T_1, có ba di chuyển có thể thực hiện dẫn đến ba trạng thái **Ta, Tb, Tc** được minh họa trong hình dưới. Những trạng thái này có số điểm là : h'(Ta)= 4; h'(Tb) = 4 và h'(Tc) = 4



Hình Các trạng thái có thể đạt được từ T_1

Thủ tục leo núi sẽ tạm dừng bởi vì tất cả các trạng thái này có số điểm thấp hơn trạng thái hiện hành. Quá trình tìm kiếm chỉ dừng lại ở một trạng thái cực đại địa phương mà không phải là cực đại toàn cục.

Chúng ta có thể đổ lỗi cho chính giải thuật leo đồi vì đã thất bại do không đủ tầm nhìn tổng quát để tìm ra lời giải. Nhưng chúng ta cũng có thể đổ lỗi cho hàm Heuristic và cố gắng sửa đổi nó. Giả sử ta thay hàm ban đầu bằng hàm Heuristic sau đây :

> **H_2 :** Đối với mỗi khối phụ trợ đúng (khối phụ trợ là khối nằm bên dưới khối hiện tại), cộng 1 điểm, ngược lại trừ 1 điểm.

Dùng hàm này, trạng thái kết thúc có số điểm là **28** vì B nằm đúng vị trí và không có khối phụ trợ nào, C đúng vị trí được 1 điểm cộng với 1 điểm do khối phụ trợ B nằm đúng vị trí nên C được 2 điểm, D được 3 điểm, ...Trạng thái khởi đầu có số điểm là **–28**. Việc di chuyển A xuống tạo thành một cột mới làm sinh ra một trạng thái với số điểm là $h'(T_1) = \mathbf{-21}$ vì A không còn 7 khối sai phía dưới nó nữa. Ba trạng thái có thể phát sinh tiếp theo bây giờ có các điểm số là : h'(Ta)=–28; h'(Tb)=–16 và h'(Tc) = –15. Lúc này thủ tục leo núi dốc đứng sẽ chọn di chuyển đến trạng thái Tc, ở đó có một khối đúng. Qua hàm H_2 này ta rút ra một nguyên tắc : *tốt hơn* không chỉ có nghĩa là có *nhiều ưu điểm* hơn mà còn phải *ít khuyết điểm* hơn. Hơn nữa, khuyết điểm không có nghĩa chỉ là sự sai biệt ngay tại một vị trí mà còn là sự khác biệt trong tương quan giữa các vị trí. Rõ ràng là đứng về mặt kết quả, cùng một thủ tục leo đồi nhưng hàm H_1 bị thất bại (do chỉ biết đánh giá ưu điểm) còn hàm H_2 mới này lại hoạt động một cách hoàn hảo (do biết đánh giá cả ưu điểm và khuyết điểm).

Đáng tiếc, không phải lúc nào chúng ta cũng thiết kế được một hàm Heuristic hoàn hảo như thế. Vì việc đánh giá ưu điểm đã khó, việc đánh giá khuyết điểm càng khó và tinh tế hơn. Chẳng hạn, xét lại vấn đề muốn đi vào khu trung tâm của một thành phố *xa lạ*. Để hàm Heuristic hiệu quả, ta cần phải đưa các thông tin về các đường một chiều và các ngõ cụt, mà trong trường hợp một thành phố hoàn toàn xa lạ thì ta khó hoặc không thể biết được những thông tin này.

Đến đây, chúng ta hiểu rõ bản chất của hai thuật giải tiếp cận theo chiến lược tìm kiếm chiều sâu. Hiệu quả của cả hai thuật giải leo đồi đơn giản và leo đồi dốc đứng phụ thuộc vào :

+ Chất lượng của hàm Heuristic.

+ Đặc điểm của không gian trạng thái.

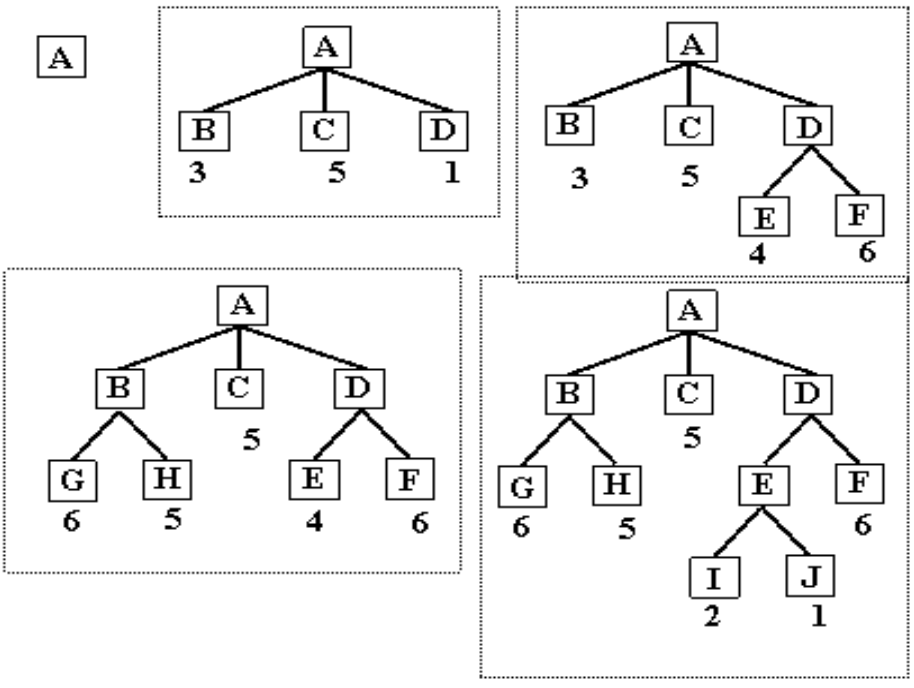
+ Trạng thái khởi đầu.

Sau đây, chúng ta sẽ tìm hiểu một tiếp cận theo mới, kết hợp được sức mạnh của cả tìm kiếm chiều sâu và tìm kiếm chiều rộng. Một thuật giải rất linh động và có thể nói là một thuật giải kinh điển của Heuristic.

III.4. Tìm kiếm ưu tiên tối ưu (best-first search)

Ưu điểm của tìm kiếm theo chiều sâu là không phải quan tâm đến sự mở rộng của tất cả các nhánh. Ưu điểm của tìm kiếm chiều rộng là không bị sa vào các đường dẫn bế tắc (các nhánh cụt). Tìm kiếm ưu tiên tối ưu sẽ kết hợp 2 phương pháp trên cho phép ta đi theo một con đường duy nhất tại một thời điểm, nhưng đồng thời vẫn "quan sát" được những hướng khác. Nếu con đường đang đi "có vẻ" không triển vọng bằng những con đường ta đang "quan sát" ta sẽ chuyển sang đi theo một trong số các con đường này. Để tiện lợi ta sẽ dùng chữ viết tắt BFS thay cho tên gọi tìm kiếm ưu tiên tối ưu.

Một cách cụ thể, tại mỗi bước của tìm kiếm BFS, ta chọn đi theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét *cho đến thời điểm đó.* (khác với leo đồi dốc đứng là chỉ chọn trạng thái có khả năng cao nhất trong số các trạng thái kế tiếp có thể đến được từ trạng thái hiện tại). Như vậy, với tiếp cận này, ta sẽ ưu tiên đi vào những nhánh tìm kiếm có khả năng nhất (giống tìm kiếm leo đồi dốc đứng), nhưng ta sẽ không bị lẩn quẩn trong các nhánh này vì nếu càng đi sâu vào một hướng mà ta phát hiện ra rằng hướng này càng đi thì càng tệ, đến mức nó xấu hơn cả những hướng mà ta chưa đi, thì ta sẽ không đi tiếp hướng hiện tại nữa mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi. Đó là tư tưởng chủ đạo của tìm kiếm BFS. Để hiểu được tư tưởng này. Bạn hãy xem ví dụ sau :



Hình Minh họa thuật giải Best-First Search

Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B,C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai nút G, H này được đánh giá ít khả năng hơn **E**, vì thế sự chú ý lại trở về E. E được mở rộng và các nút được sinh ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất. Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.

Lưu ý rằng tìm kiếm này rất giống với tìm kiếm leo đồi dốc đứng, với 2 ngoại lệ. Trong leo núi, một trạng thái được chọn và tất cả các trạng thái khác bị loại bỏ, không bao giờ chúng được xem xét lại. Cách xử lý dứt khoát này là một đặc trưng của leo đồi. Trong BFS, tại một bước, cũng có một di chuyển được chọn nhưng những cái khác vẫn được giữ lại, để ta có thể trở lại xét sau đó khi trạng thái hiện tại trở nên kém khả năng hơn những trạng thái đã được lưu trữ. Hơn nữa, ta chọn trạng thái tốt nhất mà không quan tâm đến nó có *tốt hơn* hay không các trạng thái trước đó. Điều này tương phản với leo đồi vì leo đồi sẽ dừng nếu không có trạng thái tiếp theo nào tốt hơn trạng thái hiện hành.

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, người ta thường cần dùng 2 tập hợp sau :

OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, **OPEN** là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử *tốt nhất*. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap. Các bạn có thể tham khảo thêm trong các tài liệu về Cấu trúc dữ liệu về loại dữ liệu này.

CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

Thuật giải BEST-FIRST SEARCH

1. Đặt **OPEN** chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái **tốt nhất** (Tmax) trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

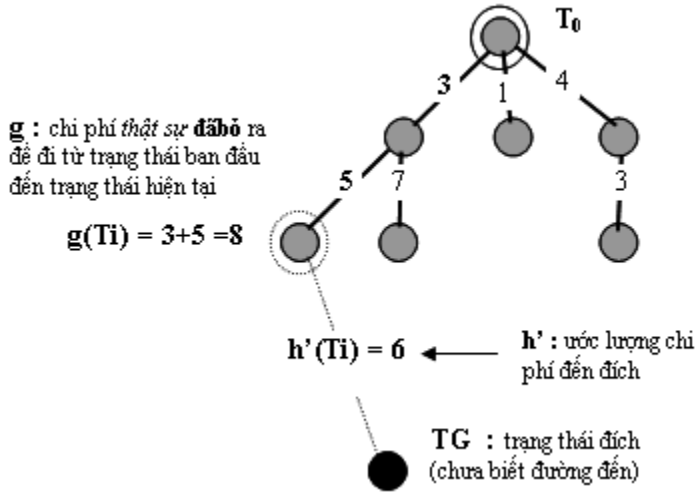
2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

Tính f(Tk); Thêm Tk vào OPEN

BFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BFS một cách trực tiếp. Thông thường, người ta thường dùng các phiên bản của BFS là AT, AKT và A^*

Thông tin về quá khứ và tương lai

Thông thường, trong các phương án tìm kiếm theo kiểu BFS, độ tốt **f** của một trạng thái được tính dựa theo 2 hai giá trị mà ta gọi là là **g** và **h'**. **h'** chúng ta đã biết, đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai). Còn **g** là "chiều dài quãng đường" *đã* đi từ trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ). Lưu ý rằng **g** là chi phí thực sự (không phải chi phí ước lượng). Để dễ hiểu, bạn hãy quan sát hình sau :



Hình 6.14 Phân biệt khái niệm g và h'

Kết hợp g và h' thành **f'** (f' = g + h') sẽ thể hiện một ước lượng về "tổng chi phí" cho con đường từ trạng thái bắt đầu đến trạng thái kết thúc dọc theo con đường đi qua trạng thái hiện hành. Để thuận tiện cho thuật giải, ta quy ước là **g** và **h'** đều không âm và càng nhỏ nghĩa là càng tốt.

III.5. Thuật giải AT

Thuật giải AT là một phương pháp tìm kiếm theo kiểu BFS với độ tốt của nút là giá trị hàm **g** – tổng chiều dài con đường đã đi từ trạng thái bắt đầu đến trạng thái hiện tại.

Thuật giải AT

1. Đặt **OPEN** chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái (Tmax) có **giá trị g nhỏ nhất** trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk);

Thêm Tk vào OPEN.

* Vì chỉ sử dụng hàm g (mà không dùng hàm ước lượng h') fsđể đánh giá độ tốt của một trạng thái nên ta cũng có thể xem AT chỉ là một thuật toán.

III.6. Thuật giải AKT

(Algorithm for Knowlegeable Tree Search)

Thuật giải AKT mở rộng AT bằng cách sử dụng thêm thông tin ước lượng h'. Độ tốt của một trạng thái f là tổng của hai hàm g và h'.

Thuật giải AKT

1. Đặt **OPEN** chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái (Tmax) có **giá trị f nhỏ nhất** trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk);

Tính h'(Tk)

f(Tk) = g(Tk) + h'(Tk);

Thêm Tk vào OPEN.

III.7. Thuật giải A*

A^* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị. Thuật giải A* có sử dụng thêm tập hợp **CLOSE** để lưu trữ những trường hợp đã được xét đến. A^* mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi "mở" một nút mà nút này đã có sẵn trong OPEN hoặc CLOSE. Khi xét đến một trạng thái Ti bên cạnh việc lưu trữ 3 giá trị cơ bản g,h', f' để phản ánh độ tốt của trạng thái đó, A^* còn lưu trữ thêm hai thông số sau :

1. *Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti)* : cho biết trạng thái dẫn đến trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến T_i thì chọn Cha(Ti) sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất, nghĩa là :

g(Ti) = g(Tcha) + cost(Tcha, Ti) là thấp nhất.

2. *Danh sách các trạng thái kế tiếp của Ti :* danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti từ trạng thái ban đầu là thấp nhất. Thực chất thì danh sách này có thể được tính ra từ thuộc tính Cha của các trạng thái được lưu trữ. Tuy nhiên, việc tính toán này có thể mất nhiều thời gian (khi tập OPEN, CLOSE được mở rộng) nên người ta thường lưu trữ ra một danh sách riêng. Trong thuật toán sau đây, chúng ta sẽ không đề cập đến việc lưu trữ danh sách này. Sau khi hiểu rõ thuật toán, bạn đọc có thể dễ dàng điều chỉnh lại thuật toán để lưu trữ thêm thuộc tính này.

1. Đặt OPEN chỉ chứa T0. Đặt g(T0) = 0, h'(T0) = 0 và f'(T0) = 0. Đặt CLOSE là tập hợp rỗng.

2. Lặp lại các bước sau cho đến khi gặp điều kiện dừng.

2.a. Nếu OPEN rỗng : bài toán vô nghiệm, thoát.

2.b. Ngược lại, chọn Tmax trong OPEN sao cho f'(Tmax) là nhỏ nhất

2.b.1. Lấy Tmax ra khỏi **OPEN** và đưa Tmax vào **CLOSE.**

2.b.2. Nếu Tmax chính là TG thì thoát và thông báo lời giải là Tmax.

2.b.3. Nếu Tmax không phải là TG. Tạo ra danh sách *tất cả* các trạng thái kế tiếp của Tmax. Gọi một trạng thái này là Tk. Với mỗi Tk, làm các bước sau :

2.b.3.1. Tính g(Tk) = g(Tmax) + cost(Tmax, Tk).

2.b.3.2. Nếu tồn tại Tk' trong OPEN trùng với Tk

Nếu g(Tk) < g(Tk') thì

Đặt g(Tk') = g(Tk)

Tính lại f'(Tk')

Đặt Cha(Tk') = Tmax

2.b.3.3. Nếu tồn tại Tk' trong CLOSE trùng với Tk

Nếu g(Tk) < g(Tk') thì

Đặt g(Tk') = g(Tk)

Tính lại f'(Tk')

Đặt Cha(Tk') = Tmax

Lan truyền sự thay đổi giá trị g, f' cho tất cả các trạng thái kế tiếp của Ti (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

2.b.3.4. Nếu Tk chưa xuất hiện trong cả **OPEN** lẫn **CLOSE** thì :

Thêm Tk vào OPEN

Tính : f' (Tk) = g(Tk)+h'(Tk).

Có một số điểm cần giải thích trong thuật giải này. Đầu tiên là việc sau khi đã tìm thấy trạng thái đích TG, làm sao để xây dựng lại được "con đường" từ T_0 đến TG. Rất đơn giản, bạn chỉ cần lần ngược theo thuộc tính Cha của các trạng thái đã được lưu trữ trong **CLOSE** cho đến khi đạt đến T_0. Đó chính là "con đường" tối ưu đi từ TG đến T_0 (hay nói cách khác là từ T_0 đến TG).

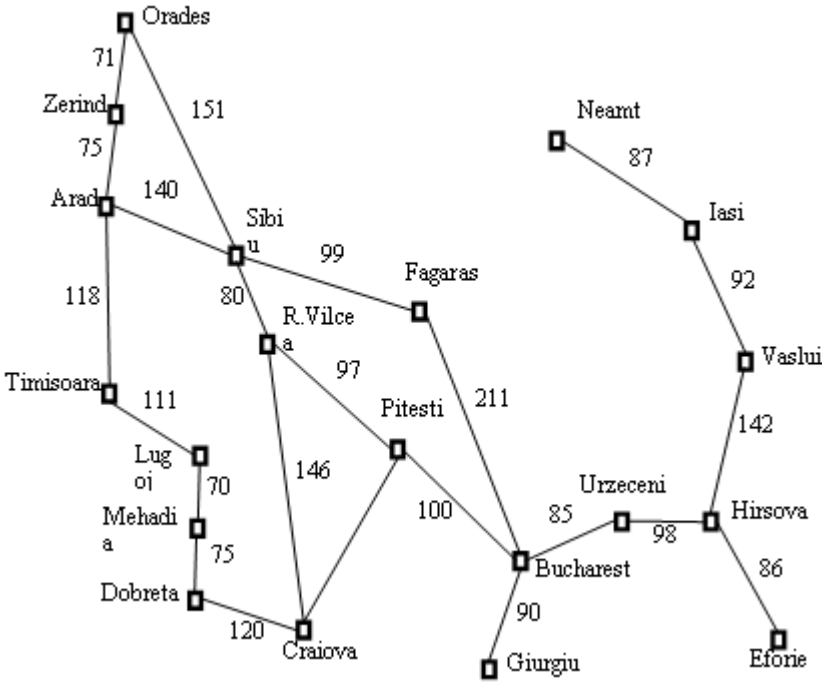
Điểm thứ hai là thao tác cập nhật lại g(Tk') , f'(Tk') và Cha(Tk') trong bước 2.b.3.2 và 2.b.3.3. Các thao tác này thể hiện tư tưởng : "luôn chọn con đường tối ưu nhất". Như chúng ta đã biết, giá trị g(Tk') nhằm lưu trữ chi phí tối ưu *thực sự* tính từ T_0 đến Tk'. Do đó, nếu chúng ta phát hiện thấy một "con đường" khác tốt hơn thông qua Tk (có chi phí nhỏ hơn) con đường hiện tại được lưu trữ thì ta phải chọn "con đường" mới tốt hơn này. Trường hợp 2.b.3.3 phức tạp hơn. Vì từ Tk' nằm trong tập CLOSE nên từ Tk' ta đã lưu trữ các trạng thái con kế tiếp xuất phát từ Tk'. Nhưng g(Tk') thay đổi dẫn đến giá trị g của các trạng thái con này cũng phải thay đổi theo. Và đến lượt các trạng thái con này lại có thể có các các trạng thái con tiếp theo của chúng và cứ thế cho đến khi mỗi nhánh kết thúc với một trạng thái trong **OPEN** (nghĩa là không có trạng thái con nào nữa). Để thực hiện quá trình cập nhật này, ta hãy thực hiện quá trình duyệt theo chiều sâu với điểm khởi đầu là Tk'. Duyệt đến đâu, ta cập nhật lại **g** của các trạng thái đến đó (dùng công thức *g(T) = g(Cha(T)) +cost(Cha(T), T))* và vì thế giá trị **f'** của các trạng thái này cũng thay đổi theo.

Một lần nữa, xin nhắc lại rằng, bạn có thể cho rằng tập OPEN lưu trữ các trạng thái "sẽ được xem xét đến sau" còn tập CLOSE lưu trữ các trạng thái "đã được xét đến rồi".

Có thể bạn sẽ cảm thấy khá lúng túng trước một thuật giải dài như thế. Vấn đề có lẽ sẻ trở nên sáng sủa hơn khi bạn quan sát các bước giải bài toán tìm đường đi ngắn nhất trên đồ thị bằng thuật giải A* sau đây.

III.8. Ví dụ minh họa hoạt động của thuật giải A*

Chúng ta sẽ minh họa hoạt động của thuật giải A* trong việc tìm kiếm đường đi ngắn nhất từ thành phố *Arad* đến thành phố *Bucharest* của Romania. Bản đồ các thành phố của Romania được cho trong đồ thị sau. Trong đó mỗi đỉnh của đồ thị của là một thành phố, giữa hai đỉnh có cung nối nghĩa là có đường đi giữa hai thành phố tương ứng. Trọng số của cung chính là chiều dài (tính bằng km) của đường đi nối hai thành phố tương ứng, chiều dài theo đường chim bay một thành phố đến Bucharest được cho trong bảng kèm theo.



Hình : Bảng đồ của Romania với khoảng cách đường tính theo km

Bảng : Khoảng cách đường chim bay từ một thành phố đến Bucharest.

Chúng ta sẽ chọn hàm **h'** chính là khoảng cách đường chim bay cho trong bảng trên và hàm chi phí **cost(Ti, Ti$_{+1}$)** chính là chiều dài con đường nối từ thành phố Ti và Ti$_{+1}$.

Sau đây là từng bước hoạt động của thuật toán A* trong việc tìm đường đi ngắn nhất từ Arad đến Bucharest.

Ban đầu :

OPEN {(Arad,g[illegible],h'[illegible],f'[illegible])}

CLOSE {}

Do trong OPEN chỉ chứa một thành phố duy nhất nên thành phố này sẽ là thành phố tốt nhất. Nghĩa là Tmax Arad.Ta lấy Arad ra khỏi OPEN và đưa vào CLOSE.

OPEN {}

CLOSE [illegible](Arad,g[illegible]0,h'[illegible]0,f'[illegible]0)}

Từ Arad có thể đi đến được 3 thành phố là Sibiu, Timisoara và Zerind. Ta lần lượt tính giá trị f', g và h' của 3 thành phố này. Do cả 3 nút mới tạo ra này chưa có nút cha nên ban đầu nút cha của chúng đều là Arad.

h'(Sibiu) [illegible]253

g(Sibiu) [illegible]g(Arad)+cost(Arad,Sibiu)

[illegible]0+140[illegible]140

f'(Sibiu) [illegible]g(Sibiu)+h'(Sibiu)

[illegible]140+253 [illegible]393

Cha(Sibiu) [illegible]Arad

h'(Timisoara) [illegible]329

g(Timisoara) [illegible]g(Arad)+cost(Arad, Timisoara)

[illegible]0+118[illegible]118

f'(Timisoara) [illegible]g(Timisoara)+ h'(Timisoara)

[illegible]118+329 [illegible]447

Cha(Timisoara) [illegible]Arad

h'(Zerind) [illegible]374

g(Zerind) [illegible]g(Arad)+cost(Arad, Zerind)

[illegible]0+75[illegible]75

f'(Zerind) [illegible]g(Zerind)+h'(Zerind)

[illegible]75+374 [illegible]449

Cha(Zerind) [illegible]Arad

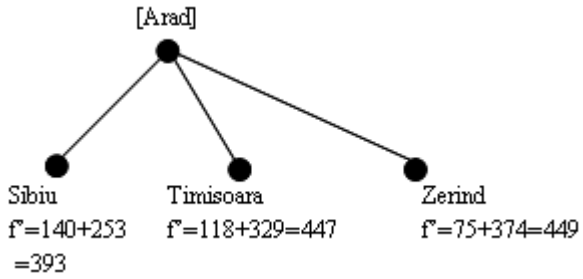
Do cả 3 nút Sibiu, Timisoara, Zerind đều không có trong cả OPEN và CLOSE nên ta bổ sung 3 nút này vào OPEN.

OPEN [illegible](Sibiu,g[illegible]140,h'[illegible]253,f'[illegible]393,Cha[illegible]Arad)

(Timisoara,g[illegible]118,h'[illegible]329,f'[illegible]447,Cha[illegible]Arad)

(Zerind,g[illegible]75,h'[illegible]374,f'[illegible]449,Cha[illegible]Arad)}

CLOSE [illegible](Arad,g[illegible]0,h'[illegible]0,f'[illegible]0)}



Hình : Bước 1, nút được đóng ngoặc vuông (như [Arad]) là nút trong tập CLOSE, ngược lại là trong tập OPEN.

Trong tập OPEN, nút Sibiu là nút có giá trị f' nhỏ nhất nên ta sẽ chọn Tmax [illegible]Sibiu. Ta lấy Sibiu ra khỏi OPEN và đưa vào CLOSE.

OPEN [illegible](Timisoara,g[illegible]118,h'[illegible]329,f'[illegible]447,Cha[illegible]Arad)

(Zerind,g[illegible]75,h'[illegible]374,f'[illegible]449,Cha[illegible]Arad)}

CLOSE [illegible](Arad,g[illegible]0,h'[illegible]0,f'[illegible]0)

(Sibiu,g[illegible]140,h'[illegible]253,f'[illegible]393,Cha[illegible]Arad)}

Từ Sibiu có thể đi đến được 4 thành phố là : Arad, Fagaras, Oradea, Rimnicu. Ta lần lượt tính các giá trị g, h', f' cho các nút này.

h'(Arad) [illegible]366

g(Arad) [illegible]g(Sibiu)+cost(Sibiu,Arad)

[illegible]140+140[illegible]280

f'(Arad) [illegible]g(Arad)+h'(Arad)

[illegible]280+366 [illegible]646

h'(Fagaras) [illegible]178

g(Fagaras) [illegible]g(Sibiu)+cost(Sibiu, Fagaras) [illegible]140+99[illegible]239

f'(Fagaras) [illegible]g(Fagaras)+ h'(Fagaras)

[illegible]239+178[illegible]417

h'(Oradea) [illegible]380

g(Oradea) = g(Sibiu)+cost(Sibiu, Oradea)

= 140+151 = 291

f’(Oradea) = g(Oradea)+ h’(Oradea)

= 291+380 = 671

h’(R.Vilcea) = 193

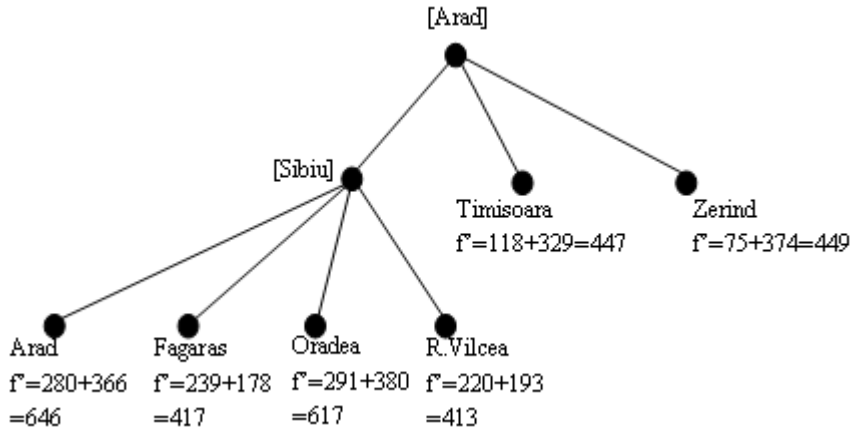
g(R.Vilcea) = g(Sibiu)+cost(Sibiu, R.Vilcea)

= 140+80 = 220

f’(R.Vilcea) = g(R.Vilcea)+ h’(R.Vilcea)

= 220+193 = 413

Nút Arad đã có trong CLOSE. Tuy nhiên, do g(Arad) mới được tạo ra (có giá trị 280) lớn hơn g(Arad) lưu trong CLOSE (có giá trị 0) nên ta sẽ không cập nhật lại giá trị g và f’ của Arad lưu trong CLOSE. 3 nút còn lại : Fagaras, Oradea, Rimnicu đều không có trong cả OPEN và CLOSE nên ta sẽ đưa 3 nút này vào OPEN, đặt cha của chúng là Sibiu. Như vậy, đến bước này OPEN đã chứa tổng cộng 5 thành phố.



OPEN = {(Timisoara,g=118,h’=329,f’=**447**,Cha=Arad)

(Zerind,g=75,h’=374,f’=**449**,Cha=Arad)

(Fagaras,g=239,h’=178,f’=**417**,Cha=Sibiu)

(Oradea,g=291,h’=380,f’=**617**,Cha=Sibiu)

(R.Vilcea,g=220,h’=193,f’=**413**,Cha=Sibiu)}

CLOSE = {(Arad,g=0,h’=0,f’=0)

(Sibiu,g[redacted]40,h'[redacted]253,f'[redacted]393,Cha[redacted]Arad)}

Trong tập OPEN, nút R.Vilcea là nút có giá trị f' nhỏ nhất. Ta chọn Tmax [redacted]R.Vilcea. Chuyển R.Vilcea từ OPEN sang CLOSE. Từ R.Vilcea có thể đi đến được 3 thành phố là Craiova, Pitesti và Sibiu. Ta lần lượt tính giá trị f', g và h' của 3 thành phố này.

h'(Sibiu) [redacted]253

g(Sibiu) [redacted]g(R.Vilcea)+ cost(R.Vilcea,Sibiu)

[redacted]220+80[redacted]300

f'(Sibiu) [redacted]g(Sibiu)+h'(Sibiu)

[redacted]300+253 [redacted]553

h'(Craiova) [redacted]160

g(Craiova) [redacted]g(R.Vilcea)+ cost(R.Vilcea, Craiova)

[redacted]220+146[redacted]366

f'(Craiova) [redacted]g(Fagaras)+h'(Fagaras)

[redacted]366+160[redacted]526

h'(Pitesti) [redacted]98

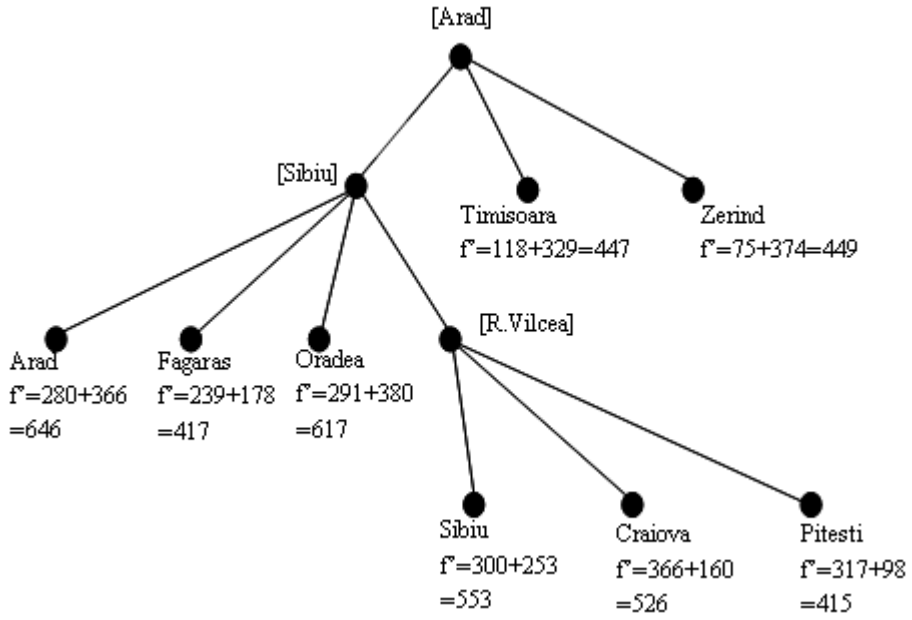
g(Pitesti) [redacted]g(R.Vilcea)+ cost(R.Vilcea, Pitesti)

[redacted]220+97 [redacted]317

f'(Pitesti) [redacted]g(Oradea)+h'(Oradea)

[redacted]317+98 [redacted]415

Sibiu đã có trong tập CLOSE. Tuy nhiên, do g'(Sibiu) mới (có giá trị là 553) lớn hơn g'(Sibiu) (có giá trị là 393) nên ta sẽ không cập nhật lại các giá trị của Sibiu được lưu trong CLOSE. Còn lại 2 thành phố là Pitesti và Craiova đều không có trong cả OPEN và CLOSE nên ta sẽ đưa nó vào OPEN và đặt cha của chúng là R.Vilcea.



OPEN [illegible](Timisoara,g[illegible]18,h'[illegible]329,f'[illegible]**447**,Cha[illegible]Arad)

(Zerind,g[illegible]75,h'[illegible]374,f'[illegible]**449**,Cha[illegible]Arad) (Fagaras,g[illegible]239,h'[illegible] 178,f'[illegible]**417**,Cha[illegible]Sibiu)

(Oradea,g[illegible]291,h'[illegible]380,f'[illegible]**617**,Cha[illegible]Sibiu) (Craiova,g[illegible]366,h'[illegible] 160,f'[illegible]**526**,Cha[illegible]R.Vilcea)

(Pitesti,g[illegible]317,h'[illegible]98,f'[illegible]**415**,Cha[illegible]R.Vilcea) }

CLOSE [illegible](Arad,g[illegible],h'[illegible],f'[illegible])

(Sibiu,g[illegible]140,h'[illegible]253,f'[illegible]**393**,Cha[illegible]Arad)

(R.Vilcea,g[illegible]220,h'[illegible]193,f'[illegible]**413**,Cha[illegible]Sibiu) }

Đến đây, trong tập OPEN, nút tốt nhất là Pitesti, từ Pitesti ta có thể đi đến được R.Vilcea, Bucharest và Craiova. Lấy Pitesti ra khỏi OPEN và đặt nó vào CLOSE. Thực hiện tiếp theo tương tự như trên, ta sẽ không cập nhật giá trị f', g của R.Vilcea và Craiova lưu trong CLOSE. Sau khi tính toán f', g của Bucharest, ta sẽ đưa Bucharest vào tập OPEN, đặt Cha(Bucharest) [illegible]Pitesti.

h'(Bucharest) [illegible]

g(Bucharest) [illegible]g(Pitesti)+cost(Pitesti, Bucharest)

[illegible]317+100[illegible]418

f'(Bucharest) [illegible]g(Fagaras)+h'(Fagaras)

[illegible]417+0[illegible]417

Ở bước kế tiếp, ta sẽ chọn được Tmax [redacted]Bucharest. Và như vậy thuật toán kết thúc (thực ra thì tại bước này, có hai ứng cử viên là Bucharest và Fagaras vì đều cùng có f'[redacted]417 , nhưng vì Bucharest là đích nên ta sẽ ưu tiên chọn hơn).

Để xây dựng lại con đường đi từ Arad đến Bucharest ta lần theo giá trị Cha được lưu trữ kèm với f', g và h' cho đến lúc đến Arad.

Cha(Bucharest) [redacted]Pitesti

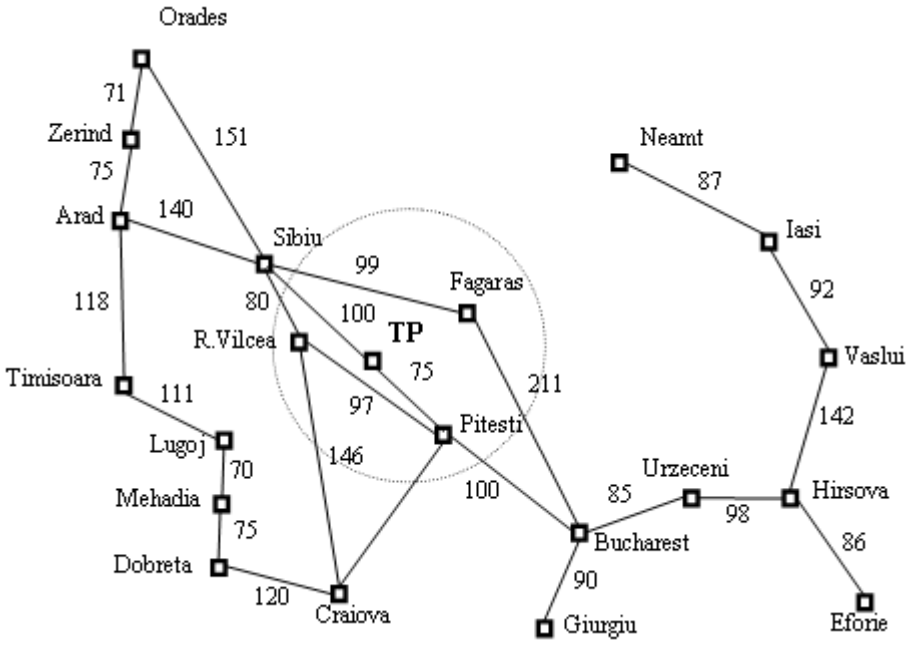
Cha(R.Vilcea) [redacted]Sibiu

Cha(Sibiu) [redacted]Arad

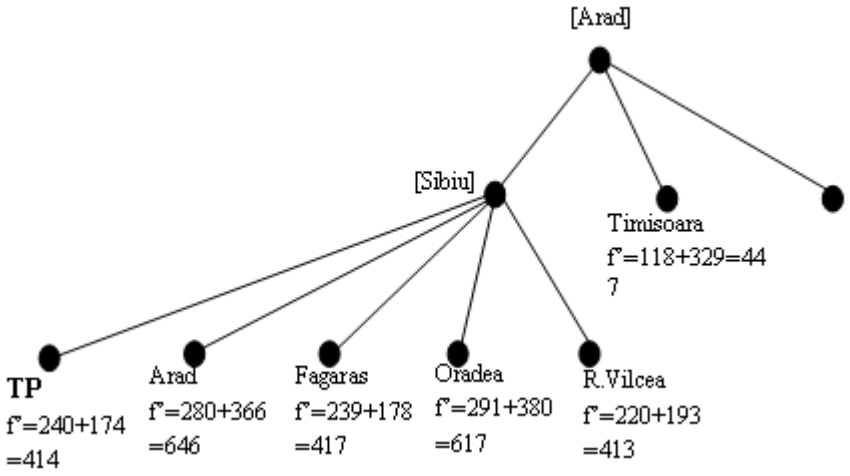
Vậy con đường đi ngắn nhất từ Arad đến Bucharest là Arad, Sibiu, R.Vilcea, Pitesti, Bucharest.

Trong ví dụ minh họa này, hàm **h'** có chất lượng khá tốt và cấu trúc đồ thị khá đơn giản nên ta gần như đi thẳng đến đích mà ít phải khảo sát các con đường khác. Đây là một trường hợp đơn giản, trong trường hợp này, thuật giải có dáng dấp của tìm kiếm chiều sâu.

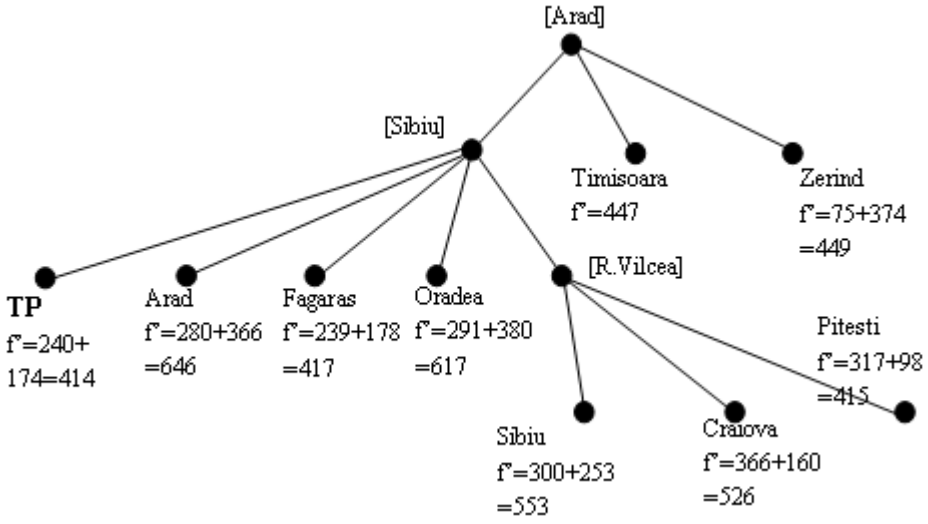
Đến đây, để minh họa một trường hợp phức tạp hơn của thuật giải. Ta thử sửa đổi lại cấu trúc đồ thị và quan sát hoạt động của thuật giải. Giả sử ta có thêm một thành phố tạm gọi là **TP** và con đường giữa **Sibiu** và **TP** có chiều dài **100**, con đường giữa **TP** và **Pitesti** có chiều dài 6**0**. Và khoảng cách đường chim bay từ TP đến Bucharest là **174**. Như vậy rõ ràng, con đường tối ưu đến Bucharest không còn là Arad, Sibiu, R.Vilcea, Pitesti, Bucharest nữa mà là Arad, Sibiu, TP, Pitesti, Bucharest.



Trong trường hợp này, chúng ta vẫn tiến hành bước 1 như ở trên. Sau khi thực hiện hiện bước 2 (mở rộng Sibiu), chúng ta có cây tìm kiếm như hình sau. Lưu ý là có thêm nhánh TP.



R.Vilcea vẫn có giá trị f' thấp nhất. Nên ta mở rộng R.Vilcea như trường hợp đầu tiên.



Bước kế tiếp của trường hợp đơn giản là mở rộng Pitesti để có được kết quả. Tuy nhiên, trong trường hợp này, TP có giá trị f' thấp hơn. Do đó, ta chọn mở rộng TP. Từ TP ta chỉ có 2 hướng đi, một quay lại Sibiu và một đến Pitesti. Để nhanh chóng, ta sẽ không tính toán giá trị của Sibiu vì biết chắc nó sẽ lớn hơn giá trị được lưu trữ trong CLOSE (vì đi ngược lại).

h'(Pitesti) [illegible]98

g(Pitesti) [illegible](TP)+cost(TP, Pitesti)

[illegible]240+75[illegible]315

f'(Pitesti) [illegible](TP)+h'(Pitesti) [illegible]315+98[illegible]413

Pistestti đã xuất hiện trong tập OPEN và g'(Pitesti) mới (có giá trị là 315) thấp hơn g'(Pitesti) cũ (có giá trị 317) nên ta phải cập nhật lại giá trị của f',g, Cha của Pitesti lưu trong OPEN. Sau khi cập nhật xong, tập OPEN và CLOSE sẽ như sau :

OPEN [illegible](Timisoara,g[illegible]18,h'[illegible]29,f'[illegible]**47**,Cha[illegible]Arad)

(Zerind,g[redacted]75,h'[redacted]374,f'[redacted]**449**,Cha[redacted]Arad)

(Fagaras,g[redacted]239,h'[redacted]178,f'[redacted]**417**,Cha[redacted]Sibiu)

(Oradea,g[redacted]291,h'[redacted]380,f'[redacted]**517**,Cha[redacted]Sibiu)

(Craiova,g[redacted]366,h'[redacted]160,f'[redacted]**526**,Cha[redacted]R.Vilcea)

*(Pitesti,g[redacted]415,h'[redacted]98,f'[redacted]**413**,Cha[redacted]**TP**)* }

CLOSE [redacted](Arad,g[redacted]0,h'[redacted]0,f'[redacted]0)

(Sibiu,g[redacted]140,h'[redacted]253,f'[redacted]**393**,Cha[redacted]Arad)

(R.Vilcea,g[redacted]220,h'[redacted]193,f'[redacted]**413**,Cha[redacted]Sibiu)

}

Đến đây ta thấy rằng, ban đầu thuật giải chọn đường đi đến Pitesti qua R.Vilcea. Tuy nhiên, sau đó, thuật giải phát hiện ra con đường đến Pitesti qua TP là tốt hơn nên nó sẽ sử dụng con đường này. Đây chính là trường hợp 2.b.iii.2 trong thuật giải.

Bước sau, chúng ta sẽ chọn mở rộng Pitesti như bình thường. Khi lần ngược theo thuộc tính Cha, ta sẽ có con đường tối ưu là Arad, Sibiu, TP, Pitesti, Bucharest.

III.9. Bàn luận về A*

Đến đây, có lẽ bạn đã hiểu được thuật giải này. Ta có một vài nhận xét khá thú vị về A*. Đầu tiên là vai trò của **g** trong việc giúp chúng ta lựa chọn đường đi. Nó cho chúng ta khả năng lựa chọn trạng thái nào để mở rộng tiếp theo, không chỉ dựa trên việc trạng thái đó tốt như thế nào (thể hiện bởi giá trị **h'**) mà còn trên cơ sở con đường từ trạng thái khởi đầu đến trạng thái hiện tại đó tốt ra sao. Điều này sẽ rất hữu ích nếu ta không chỉ quan tâm việc tìm ra lời giải hay không mà còn quan tâm đến hiệu quả của con đường dẫn đến lời giải. Chẳng hạn như trong bài toán tìm đường đi ngắn nhất giữa hai điểm. Bên cạnh việc tìm ra đường đi giữa hai điểm, ta còn phải tìm ra một con đường ngắn nhất. Tuy nhiên, nếu ta chỉ quan tâm đến việc **tìm được lời giải** (mà không quan tâm đến hiệu quả của con đường đến lời giải), chúng ta có thể đặt g=0 ở mọi trạng thái. Điều này sẽ giúp ta luôn chọn đi theo trạng thái có vẻ gần nhất với trạng thái kết thúc (vì lúc này **f'** chỉ phụ thuộc vào **h'** là hàm ước lượng "khoảng cách" gần nhất để tới đích). Lúc này thuật giải có dáng dấp của tìm kiếm chiều sâu theo nguyên lý hướng đích kết hợp với lần ngược.

Ngược lại, nếu ta muốn tìm ra kết quả với **số bước ít nhất** (đạt được trạng thái đích với số trạng thái trung gian ít nhất), thì ta đặt giá trị để đi từ một trạng thái đến các trạng thái con kế tiếp của nó luôn là hằng số, thường là 1 Nghĩa đặt cost(T_{i-1}, Ti) = 1 (và vẫn dùng một hàm ước lượng **h'** như bình thường). Còn ngược lại, nếu muốn tìm chi phí rẻ nhất thì ta phải đặt giá trị hàm cost chính xác (phản ánh đúng ghi phí thực sự).

Đến đây, chắc bạn đọc đã có thể bắt đầu cảm nhận được rằng thuật giải A* không hoàn toàn là một thuật giải tối ưu tuyệt đối. Nói đúng hơn, A* chỉ là một thuật giải linh động và cho chúng ta khá nhiều tùy chọn. Tùy theo bài toán mà ta sẽ có một bộ thông số thích hợp cho A* để thuật giải hoạt động hiệu quả nhất.

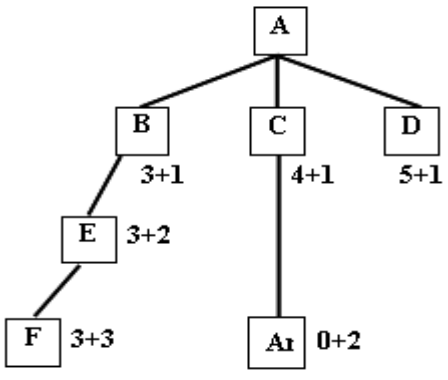
Điểm quan tâm thứ hai là về giá trị **h'** – sự ước lượng khoảng cách (chi phí) từ một trạng thái đến trạng thái đích. Nếu **h'** chính là **h** (đánh giá tuyệt đối chính xác) thì A* sẽ đi một mạch từ trạng thái đầu đến trạng thái kết thúc mà không cần phải thực hiện bất kỳ một thao tác đổi hướng nào!. Dĩ nhiên, trên thực tế, hầu như chẳng bao giờ ta tìm thấy một đánh giá tuyệt đối chính xác. Tuy nhiên, điều đáng quan tâm ở đây là **h'** được ước lượng càng gần với **h**, quá trình tìm kiếm càng ít bị sai sót, ít bị rẽ vào những nhánh cụt hơn. Hay nói ngắn gọn là càng nhanh chóng tìm thấy lời giải hơn.

Nếu **h'** luôn bằng 0 ở mọi trạng thái (trở về thuật giải AT) thì quá trình tìm kiếm sẽ được điều khiển hoàn toàn bởi giá trị **g**. Nghĩa là thuật giải sẽ chọn đi theo những hướng mà sẽ tốn ít chi phí/bước đi nhất (chi phí tính từ trạng thái đầu tiên đến trạng thái hiện đang xét) bất chấp việc đi theo hướng đó có khả năng dẫn đến lời giải hay không. Đây chính là hình ảnh của nguyên lý tham lam (Greedy).

Nếu chi phí từ trạng thái sang trạng thái khác luôn là hằng số (dĩ nhiên lúc này h' luôn bằng 0) thì thuật giải A* trở thành thuật giải tìm kiếm theo chiều rộng! Lý do là vì tất cả những trạng thái cách trạng thái khởi đầu n bước đều có cùng giá trị **g** và vì thế đều có cùng **f'** và giá trị này sẽ nhỏ hơn tất cả các trạng thái cách trạng thái khởi đầu n+1 bước. Và nếu **g** luôn bằng 0 và h' cũng luôn bằng 0, mọi trạng thái đang xét đều tương đương nhau. Ta chỉ có thể chọn bằng trạng thái kế tiếp bằng ngẫu nhiên !

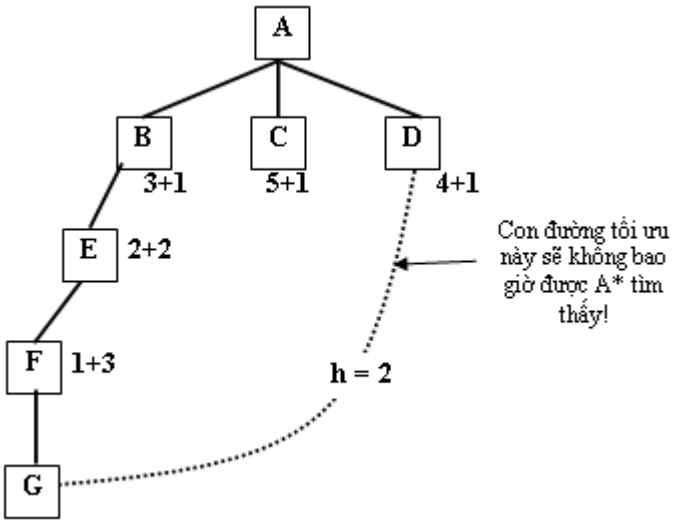
Còn nếu như **h'** không thể tuyệt đối chính xác (nghĩa là không bằng đúng h) và cũng không luôn bằng 0 thì sao? Có điều gì thú vị về cách xử lý của quá trình tìm kiếm hay không? Câu trả lời là có. Nếu như bằng một cách nào đó, ta có thể chắc chắn rằng, ước lượng **h'** luôn nhỏ hơn **h** (đối với mọi trạng thái) thì thuật giải A* sẽ thường tìm ra con đường tối ưu (xác định bởi g) để đi đến đích, nếu đường dẫn đó tồn tại và quá trình tìm kiếm sẽ ít khi bị sa lầy vào những con đường quá dở. Còn nếu vì một lý do nào đó, ước lượng h' lại lớn hơn h thì thuật giải sẽ dễ dàng bị vướng vào những hướng tìm kiếm vô ích. Thậm chí nó lại có khuynh hướng tìm kiếm ở những hướng đi vô ích trước! Điều này có thể thấy một cách dễ dàng từ vài ví dụ.

Xét trường hợp được trình bày trong hình sau. Giả sử rằng tất cả các cung đều có giá trị 1. **G** là trạng thái đích. Khởi đầu, **OPEN** chỉ chứa A, sau đó A được mở rộng nên B, C, D sẽ được đưa vào OPEN (hình vẽ mô tả trạng thái 2 bước sau đó, khi B và E đã được mở rộng). Đối với mỗi nút, con số đầu tiên là giá trị **h'**, con số kế tiếp là **g**. Trong ví dụ này, nút B có f' thấp nhất là 4 = **h'**+**g** = 3 + 1 , vì thế nó được mở rộng trước tiên. Giả sử nó chỉ có một nút con tiếp theo là E và h'(E) = 3, do E các A hai cung nên g(E) = 2 suy ra f'(E) = 5, giống như f'(C). Ta chọn mở rộng E kế tiếp. Giả sử nó cũng chỉ có duy nhất một con kế tiếp là F và h'(F) cũng bằng 3. Rõ ràng là chúng ta đang di chuyển xuống và không phát triển rộng. Nhưng f'(F) = 6 lớn hơn f'(D). Do đó, chúng ta sẽ mở rộng C tiếp theo và đạt đến trạng thái đích. Như vậy, ta thấy rằng do đánh giá thấp h(B) nên ta đã lãng phí một số bước (E,F), nhưng cuối cùng ta cùng phát hiện ra B khác xa với điều ta mong đợi và quay lại để thử một đường dẫn khác.



Hình : h' đánh giá thấp h

Bây giờ hãy xét trường hợp ở hình tiếp theo. Chúng ta cũng mở rộng B ở bước đầu tiên và E ở bước thứ hai. Kế tiếp là F và cuối cùng G, cho đường dẫn kết thúc có độ dài là 4. Nhưng giả sử có đường dẫn trực tiếp từ D đến một lời giải có độ dài **h** *thực sự* là 2 thì chúng ta sẽ *không bao giờ* tìm được đường dẫn này (tuy rằng ta có thể tìm thấy lời giải). Bởi vì việc đánh giá quá cao h'(D), chúng ta sẽ làm cho D trông dở đến nỗi mà ta phải tìm một đường đi khác – đến một lời giải tệ hơn - mà không bao giờ nghĩ đến việc mở rộng D. Nói chung, nếu **h'** đánh giá cao **h** thì A* sẽ có thể không thể tìm ra đường dẫn tối ưu đến lời giải (nếu như có nhiều đường dẫn đến lời giải). Một câu hỏi thú vị là "*Liệu có một nguyên tắc chung nào giúp chúng ta đưa ra một cách ước lượng h' không bao giờ đánh giá cao h hay không?*". Câu trả lời là "hầu như không", bởi vì đối với hầu hết các vấn đề thực ta đều không biết h. Tuy nhiên, cách duy nhất để bảo đảm **h'** không bao giờ đánh giá cao **h** là đặt **h'** bằng 0 !



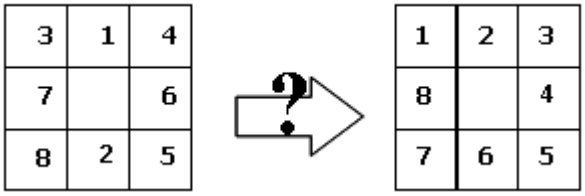
Hình : h' đánh giá cao h

Đến đây chúng ta đã kết thúc việc bàn luận về thuật giải A*, một thuật giải linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Chính vì thế mà người ta thường nói, A* chính là thuật giải tiêu biểu cho Heuristic.

A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản – giống như chiến lược tìm kiếm chiều rộng – đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua – nếu chúng ta muốn nó chắc chắn tìm thấy lời giải tối ưu. Với những không gian tìm kiếm lớn nhỏ thì đây không phải là một điểm đáng quan tâm. Tuy nhiên, với những không gian tìm kiếm khổng lồ (chẳng hạn tìm đường đi trên một ma trận kích thước cỡ 10^6 x 10^6) thì không gian lưu trữ là cả một vấn đề hóc búa. Các nhà nghiên cứu đã đưa ra khá nhiều các hướng tiếp cận lai để giải quyết vấn đề này. Chúng ta sẽ tìm hiểu một số phương án nhưng quan trọng nhất, ta cần phải nắm rõ vị trí của A* so với những thuật giải khác.

III.10. Ứng dụng A* để giải bài toán Ta-canh

Bài toán Ta-canh đã từng là một trò chơi khá phổ biến, đôi lúc người ta còn gọi đây là bài toán 9-puzzle. Trò chơi bao gồm một hình vuông kích thướoc 3x3 ô. Có 8 ô có số, mỗi ô có một số từ 1 đến 8. Một ô còn trống. Mỗi lần di chuyển chỉ được di chuyển một ô nằm cạnh ô trống về phía ô trống. Vấn đề là từ một trạng thái ban đầu bất kỳ, làm sao đưa được về trạng thái cuối là trạng thái mà các ô được sắp lần lượt từ 1 đến 8 theo thứ tự từ trái sang phải, từ trên xuống dưới, ô cuối dùng là ô trống.



Cho đến nay, ngoại trừ 2 giải pháp vét cạn và tìm kiếm Heuristic, người ta vẫn chưa tìm được một thuật toán chính xác, tối ưu để giải bài toán này. Tuy nhiên, cách giải theo thuật giải A^* lại khá đơn giản và thường tìm được lời giải (nhưng không phải lúc nào cũng tìm được lời giải). Nhận xét rằng: Tại mỗi thời điểm ta chỉ có tối đa 4 ô có thể di chuyển. Vấn đề là tại thời điểm đó, ta sẽ chọn lựa di chuyển ô nào? Chẳng hạn ở hình trên, ta nên di chuyển (1), (2), (6), hay (7) ? Bài toán này hoàn toàn có cấu trúc thích hợp để có thể giải bằng A* (tổng số trạng thái có thể có của bàn cờ là $n^2!$ với n là kích thước bàn cờ vì mỗi trạng thái là một hoán vị của tập n^2 con số).

Tại một trạng thái đang xét Tk, đặt d(i,j)là **số ô** cần di chuyển để đưa con số ở ô (i,j) về đúng vị trí của nó ở trạng thái đích.

Hàm ước lượng h' tại trạng thái Tk bất kỳ bằng tổng của các d(i,j) sao cho vị trí (i,j) không phải là ô trống.

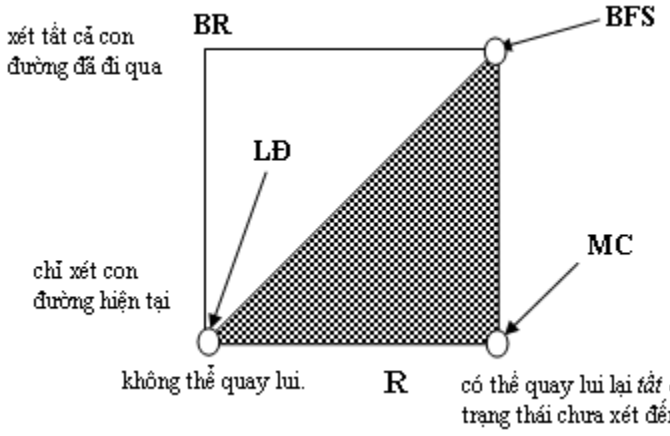
Như vậy đối với trạng thái ở hình ban đầu, hàm f(Tk) sẽ có giá trị là

$$Fk=2+1+3+1+0+1+2+2=12$$

III.11. Các chiến lược tìm kiếm lai

Chúng ta đã biết qua 4 kiểu tìm kiếm : leo đèo (LĐ), tìm theo chiều sâu (MC), tìm theo chiều rộng (BR) và tìm kiếm BFS. Bốn kiểu tìm kiếm này có thể được xem như 4 thái cực của không gian liên tục bao gồm các chiến lược tìm kiếm khác nhau. Để giải thích điều này rõ hơn, sẽ tiện hơn cho chúng ta nếu nhìn một chiến lược tìm kiếm lời giải dưới hai chiều sau :

> ***Chiều khả năng quay lui*** **(R)**: là khả năng cho phép quay lại để xem xét những trạng thái xét đến trước đó nếu gặp một trạng thái không thể đi tiếp.
>
> ***Chiều phạm vi của sự đánh giá*** **(S)**: số các trạng thái xét đến trong mỗi quyết định.

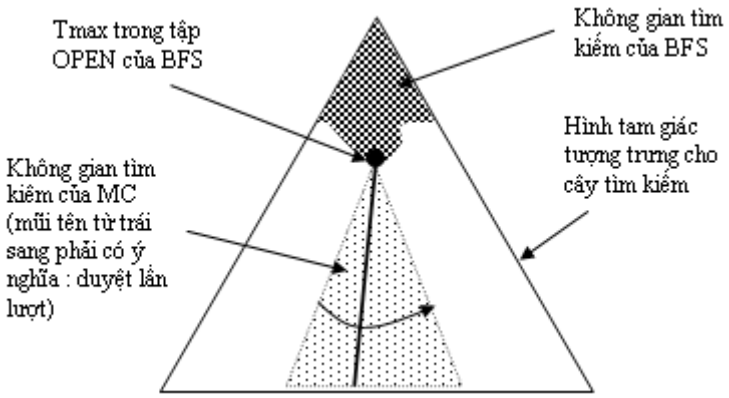


Hình : Tương quan giữa các chiến lược leo đèo, quay lui và tốt nhất

Theo hướng R, chúng ta thấy leo đèo nằm ở một thái cực (nó không cho phép quay lại những trạng thái chưa được xét đến), trong khi đó tìm kiếm quay lui và BFS ở một thái cực khác (cho phép quay lại tất cả các hướng đi chưa xét đến). Theo hướng S chúng ta thấy leo đèo và lần ngược nằm ở một thái cực (chỉ tập trung vào một phạm vi hẹp trên tập các trạng thái mới tạo ra từ trạng thái hiện tại) và BFS nằm ở một thái cực khác (trong khi BF xem xét toàn bộ tập các con đường đã có, bao gồm cả những con đường mới được tạo ra cũng như tất cả những con đường không được xét tới trước đây trước mỗi một quyết định).

Những thái cực này được trực quan hóa bằng hình ở trên. Vùng in đậm biểu diễn một mặt phẳng liên tục các chiến lược tìm kiếm mà nó kết hợp một số đặc điểm của một trong ba thái cực (leo đèo, chiều sâu, BFS) để có được một hòa hợp các đặc tính tính toán của chúng.

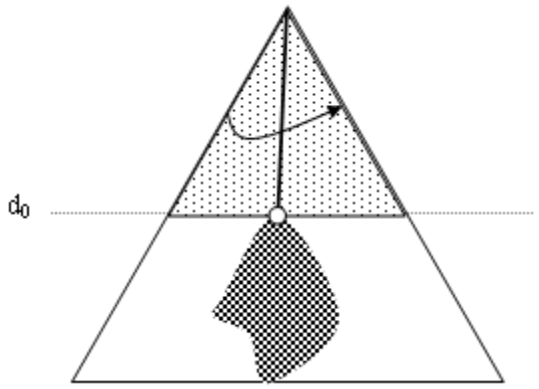
Nếu chúng ta không đủ bộ nhớ cần thiết để áp dụng thuật toán BFS thuần túy. Ta có thể kết hợp BFS với tìm theo chiều sâu để giảm bớt yêu cầu bộ nhớ. Dĩ nhiên, cái giá mà ta phải trả là số lượng các trạng thái có thể xét đến tại một bước sẽ nhỏ đi. Một loại kết hợp như thế được chỉ ra trong hình dưới. Trong hình này, thuật giải BFS được áp dụng tại đỉnh của đồ thị tìm kiếm (biểu diễn bằng vùng tô tậm) và tìm kiếm theo chiều sâu được áp dụng tại đáy (biểu diễn bởi tam giác tô nhạt). Đầu tiên ta áp dụng BFS vào trạng thái ban đầu T_0 một cách bình thường. BFS sẽ thi hành cho đến một lúc nào đó, số lượng trạng thái được lưu trữ chiếm dụng một không gian bộ nhớ vượt quá một mức cho phép nào đó. Đến lúc này, ta sẽ áp dụng tìm kiếm chiều sâu xuất phát từ trạng thái tốt nhất Tmax trong OPEN cho tới khi toàn bộ không gian con phía "dưới" trạng thái đó được duyệt hết. Nếu không tìm thấy kết quả, trạng thái Tmax này được ghi nhận là không dẫn đến kết quả và ta lại chọn ra trạng thái tốt thứ hai trong OPEN và lại áp dụng tìm kiếm chiều sâu cho cho phần không gian phía "dưới" trạng thái này....



Hình : Chiến lược lai BFS-MC trong đó, BFS áp dụng tại đỉnh và MC tại đáy.

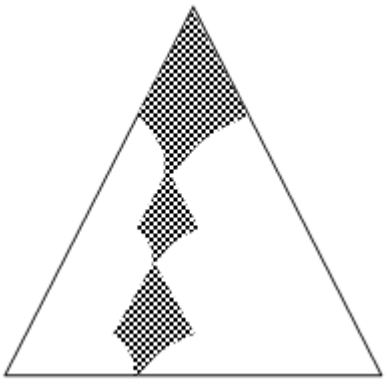
Một cách kết hợp khác là dùng tìm kiếm chiều sâu tại đỉnh không gian tìm kiếm và BFS được dùng tại đáy. Chúng ta áp dụng tìm kiếm chiều sâu cho tới khi gặp một trạng thái Tk mà độ sâu (số trạng thái trung gian) của nó vượt quá một ngưỡng d_0 nào đó. Tại điểm này, thay vì lần ngược trở lại, ta áp dụng kiểu tìm kiếm BFS cho phần không gian phía "dưới" bắt đầu từ Tk cho tới khi nó trả về một giải pháp hoặc không tìm thấy. Nếu nó không tìm thấy kết quả, chúng ta lần ngược trở lại và lại dùng BFS khi đạt độ sâu d_0. Tham số d_0 sẽ được chọn sao cho bộ nhớ dùng cho tìm kiếm BFS trên không gian "dưới" mức d_0 sẽ không vượt quá một hằng số cho trước. Rõ ràng ta ta không dễ gì xác định được d_0 (vì nói chung, ta khó đánh giá được không gian bài toán rộng đến mức nào). Tuy nhiên, kiểu kết hợp này lại có một thuận lợi. Phần đáy không gian tìm kiếm thường chứa nhiều thông tin "bổ ích" hơn là phần đỉnh. (Chẳng hạn, tìm đường đi đến khu trung tâm của thành phố, khi càng đến gần khu trung tâm – đáy đồ thị – bạn càng dễ dàng tiến đến trung tâm hơn vì có nhiều "dấu hiệu" của trung tâm xuất hiện xung quanh bạn!). Nghĩa là, càng tiến về phía đáy của không

gian tìm kiếm, ước lượng **h'** thường càng trở nên chính xác hơn và do đó, càng dễ dẫn ta đến kết quả hơn.



Hình : Chiến lược lai BFS-MC trong đó, MC áp dụng tại đỉnh và BFS tại đáy.

Còn một kiểu kết hợp phức tạp hơn nữa. Trong đó, BFS được thực hiện cục bộ và chiều sâu được thực hiện toàn cục. Ta bắt đầu tìm kiếm theo BFS cho tới khi một sự lượng bộ nhớ xác định M_0 được dùng hết. Tại điểm này, chúng ta xem tất cả những trạng thái trong OPEN như những trạng thái con trực tiếp của trạng thái ban đầu và chuyển giao chúng cho tìm kiếm chiều sâu. Tìm kiếm chiều sâu sẽ chọn trạng thái tốt nhất trong những trạng thái con này và "bành trướng" nó dùng BFS, nghĩa là nó chuyển trạng thái đã chọn cho tìm kiếm BFS cục bộ cho đến khi một lượng bộ nhớ M_0 lại được dùng hết và trạng thái con mới trong OPEN lại tiếp tục được xem như nút con của nút "bành trướng"...Nếu việc "bành trướng" bằng BFS thất bại thì ta quay lui lại và chọn nút con tốt thứ hai của tập OPEN trước đó, rồi lại tiếp tục bành trướng bằng BFS...



Hình : Chiến lược lai BFS-MC trong đó, BFS được áp dụng cục bộ và chiều sâu được áp dụng toàn cục.

Có một cách phối hợp nổi tiếng khác được gọi là tìm kiếm theo giai đoạn được thực hiện như sau. Thay vì lưu trữ trong bộ nhớ toàn bộ cây tìm kiếm được sinh ra bởi BFS, ta chỉ giữ lại cây con có triển vọng nhất. Khi một lượng bộ nhớ M_0 được dùng hết, ta sẽ đánh dấu một tập con các trạng thái trong **OPEN** (những trạng thái có giá trị hàm f thấp nhất) để giữ lại; những đường đi tốt nhất qua những trạng thái này cũng sẽ được ghi nhớ và tất cả phần còn lại của cây bị loại bỏ. Quá trình tìm kiếm sau đó sẽ tiếp tục theo BFS cho tới khi một lượng bộ nhớ M_0 lại được dùng hết và cứ thế. Chiến lược này có thể được xem như là một sự lai ghép giữa BF và leo đèo. Trong đó, leo đèo thuần

túy loại bỏ tất cả nhưng chỉ giữ lại phương án tốt nhất còn tìm kiếm theo giai đoạn loại bỏ tất cả nhưng chỉ giữ lại *tập* các phương án tốt nhất.

CHƯƠNG 2 : BIỂU DIỄN TRI THỨC

I. MỞ ĐẦU

II. THÔNG TIN, DỮ LIỆU VÀ TRI THỨC

III. THUẬT TOÁN – MỘT PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC?

IV. LÀM QUEN VỚI CÁCH GIẢI QUYẾT VẤN ĐỀ BẰNG CÁCH CHUYỂN GIAO TRI THỨC CHO MÁY TÍNH

V. LOGIC MỆNH ĐỀ

VI. LOGIC VỊ TỪ

VII. MỘT SỐ THUẬT GIẢI LIÊN QUAN ĐẾN LOGIC MỆNH ĐỀ

VII.1. Thuật giải Vương Hạo

VII.2 Thuật giải Robinson

VIII. BIỂU DIỄN TRI THỨC SỬ DỤNG LUẬT DẪN XUẤT (LUẬT SINH)

VIII.1. Khái niệm

VIII.2. Cơ chế suy luận trên các luật sinh

VIII.3. Vấn đề tối ưu luật

VIII.4. Ưu điểm và nhược điểm của biểu diễn tri thức bằng luật

X. BIỂU DIỄN TRI THỨC SỬ DỤNG MẠNG NGỮ NGHĨA

X.1. Khái niệm

X.2. Ưu điểm và nhược điểm của mạng ngữ nghĩa

X.3. Một ví dụ tiêu biểu

XI. BIỂU DIỄN TRI THỨC BẰNG FRAME

XI.1. Khái niệm

XI.2. Cấu trúc của frame

XI.3. Tính kế thừa

XII. BIỂU DIỄN TRI THỨC BẰNG SCRIPT

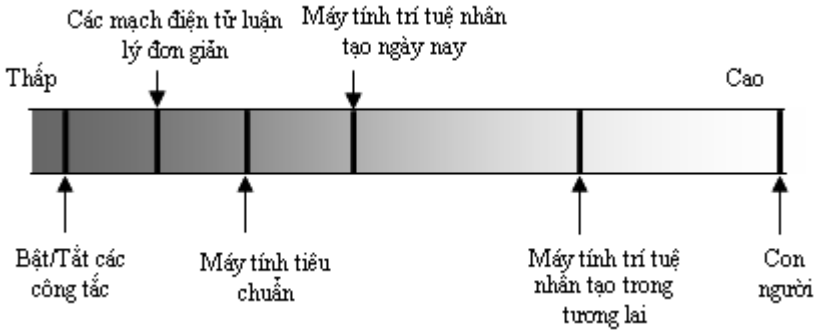
XIII. PHỐI HỢP NHIỀU CÁCH BIỂU DIỄN TRI THỨC

A. TỔNG QUAN TRÍ TUỆ NHÂN TẠO

I. MỞ ĐẦU

Chế tạo được những cỗ máy thông minh như con người (thậm chí thông minh hơn con người) là một ước mơ cháy bỏng của loài người từ hàng ngàn năm nay. Hẳn bạn đọc còn nhớ đến nhà khoa học Alan Turing cùng những đóng góp to lớn của ông trong lĩnh vực trí tuệ nhân tạo. Năng lực máy tính ngày càng mạnh mẽ là một điều kiện hết sức thuận lợi cho trí tuệ nhân tạo. Điều này cho phép những chương trình máy tính áp dụng các thuật giải trí tuệ nhân tạo có khả năng phản ứng nhanh và hiệu quả hơn trước. Sự kiện máy tính Deep Blue đánh bại kiện tướng cờ vua thế giới *Casparov* là một minh chứng hùng hồn cho một bước tiến dài trong công cuộc nghiên cứu về trí tuệ nhân tạo. Tuycó thể đánh bại được Casparov nhưng Deep Blue là một cỗ máy *chỉ* biết đánh cờ ! Nó thậm chí không có được trí thông minh sơ đẳng của một đứa bé biết lên ba như nhận diện được những người thân, khả năng quan sát nhận biết thế giới, tình cảm thương, ghét, ... Ngành trí tuệ nhân tạo đã có những bước tiến đáng kể, nhưng một trí tuệ nhân tạo thực sự vẫn chỉ có trong những bộ phim khoa học giả tưởng của Hollywood. Vậy thì tại sao chúng ta vẫn nghiên cứu về trí tuệ nhân tạo? Điều này cũng tương tự như ước mơ chế tạo vàng của các nhà giả kim thuật thời Trung Cổ, tuy chưa thành công nhưng chính quá trình nghiên cứu đã làm sáng tỏ nhiều vấn đề.

Mặc dù mục tiêu tối thượng của ngành TTNT là xây dựng một chiếc máy có năng lực tư duy tương tự như con người nhưng khả năng hiện tại của tất cả các sản phẩm TTNT vẫn còn rất khiêm tốn so với mục tiêu đã đề ra. Tuy vậy, ngành khoa học mới mẻ này vẫn đang tiến bộ mỗi ngày và đang tỏ ra ngày càng hữu dụng trong một số công việc đòi hỏi trí thông minh của con người. Hình ảnh sau sẽ giúp bạn hình dung được tình hình của ngành trí tuệ nhân tạo.



Trước khi bước vào tìm hiểu về trí tuệ nhân tạo, chúng ta hãy nhắc lại một định nghĩa được nhiều nhà khoa học chấp nhận.

Mục tiêu của ngành khoa học trí tuệ nhân tạo ?

Tạo ra những chiếc máy tính có khả năng nhận thức, suy luận và phản ứng.

Nhận thức được hiểu là khả năng quan sát, học hỏi, hiểu biết cũng như những kinh nghiệm về thế giới xung quanh. Quá trình nhận thức giúp con người có tri thức. Suy luận là khả năng vận dụng những tri thức sẵn có để phản ứng với những tình huống hay những vấn đề - bài toán gặp phải trong cuộc sống. Nhận thức và suy luận để từ đó đưa ra những phản ứng thích hợp là ba hành vi có thể nói là đặc trưng cho trí tuệ của con người. (Dĩ nhiên còn một yếu tố nữa là tình cảm. Nhưng chúng

ta sẽ không đề cập đến ở đây!). Do đó, cũng không có gì ngạc nhiên khi muốn tạo ra một chiếc máy tính thông minh, ta cần phải trang bị cho nó những khả năng này. Cả ba khả năng này đều cần đến một yếu tố cơ bản là tri thức.

Dưới góc nhìn của tập sách này, xây dựng trí tuệ nhân tạo là tìm cách **biểu diễn tri thức, tìm cách vận dụng tri thức** để giải quyết vấn đề và **tìm cách bổ sung tri thức** bằng cách "phát hiện" tri thức từ các thông tin sẵn có **(máy học).**

II. THÔNG TIN, DỮ LIỆU VÀ TRI THỨC

Tri thức là một khái niệm rất trừu tượng. Do đó, chúng ta sẽ không cố gắng đưa ra một định nghĩa hình thức chính xác ở đây. Thay vào đó, chúng ta hãy cùng nhau cảm nhận khái niệm "tri thức" bằng cách so sánh nó với hai khái niệm khác là thông tin và dữ liệu.

Nhà bác học nổi tiếng Karan Sing đã từng nói rằng *"Chúng ta đang ngập chìm trong **biển thông tin** nhưng lại đang **khát tri thức**"*. Câu nói này làm nổi bật sự khác biệt về lượng lẫn về chất giữa hai khái niệm thông tin và tri thức.

Trong ngữ cảnh của ngành khoa học máy tính, người ta quan niệm rằng **dữ liệu** là các con số, chữ cái, hình ảnh, âm thanh... mà máy tính có thể tiếp nhận và xử lý. Bản thân dữ liệu thường không có ý nghĩa đối với con người. Còn thông tin là tất cả những gì mà con người có thể cảm nhận được một cách trực tiếp thông qua các giác quan của mình (khứu giác, vị giác, thính giác, xúc giác, thị giác và giác quan thứ 6) hoặc gián tiếp thông qua các phương tiện kỹ thuật như tivi, radio, cassette,... Thông tin đối với con người luôn có một ý nghĩa nhất định nào đó. Với phương tiện máy tính (mà cụ thể là các thiết bị đầu ra), con người sẽ tiếp thu được *một phần* dữ liệu có ý nghĩa đối với mình. Nếu so về lượng, dữ liệu thường nhiều hơn thông tin.

Cũng có thể quan niệm thông tin là quan hệ giữa các dữ liệu. Các dữ liệu được sắp xếp theo một thứ tự hoặc được tập hợp lại theo một quan hệ nào đó sẽ chứa đựng thông tin. Nếu những quan hệ này được chỉ ra một cách rõ ràng thì đó là các tri thức. Chẳng hạn :

Trong toán học :

Bản thân từng con số riêng lẻ như 1, 1, 3, 5, 2, 7, 11, ... là các dữ liệu. Tuy nhiên, khi đặt chúng lại với nhau theo trật tự như dưới đây thì giữa chúng đã bắt đầu có một mối liên hệ

Dữ liệu : 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Mối liên hệ này có thể được biểu diễn bằng công thức sau : $Un = Un_{-1} + Un_{-2}$.

Công thức nêu trên chính là tri thức.

Trong vật lý :

Bản sau đây cho chúng ta biết số đo về điện trở (R), điện thế (U) và cường độ dòng điện (I) trong một mạch điện.

I	U	R
5	10	2
2.5	20	8
4	12	3
7.3	14.6	2

Bản thân những con số trong các cột của bản trên không có mấy ý nghĩa nếu ta tách rời chúng ta. Nhưng khi đặt kế nhau, chúng đã cho thấy có một sự liên hệ nào đó. Và mối liên hệ này có thể được diễn tả bằng công thức đơn giản sau :

$$I = U/R$$

Công thức này là tri thức.

Trong cuộc sống hàng ngày :

Hằng ngày, người nông dân vẫn quan sát thấy các hiện tượng nắng, mưa, râm và chuồn chuồn bay. Rất nhiều lần quan sát, họ đã có nhận xét như sau :

Chuồn chuồn bay thấp thì mưa, bay cao thì nắng, bay vừa thì râm.

Lời nhận xét trên là tri thức.

> Có quan điểm trên cho rằng chỉ những mối liên hệ *tường minh* (có thể chứng minh được) giữa các dữ liệu mới được xem là tri thức. Còn những mối quan hệ *không tường minh* thì không được công nhận. Ở đây, ta cũng có thể quan niệm rằng, *mọi mối liên hệ* giữa các dữ liệu đều có thể được xem là tri thức, bởi vì, những mối liên hệ này thực sự tồn tại. Điểm khác biệt là chúng ta chưa phát hiện ra nó mà thôi. Rõ ràng rằng "dù sao thì trái đất cũng vẫn xoay quanh mặt trời" dù tri thức này có được Galilê phát hiện ra hay không!

Như vậy, so với dữ liệu thì tri thức có số lượng *ít* hơn rất nhiều. Thuật ngữ *ít* ở đây không chỉ đơn giản là một dấu nhỏ hơn bình thường mà là *sự kết tinh* hoặc *cô đọng* lại. Bạn hãy hình dung dữ liệu như là những điểm trên mặt phẳng còn tri thức chính là *phương trình* của đường cong nối tất cả những điểm này lại. Chỉ cần *một* phương trình đường cong ta có thể biểu diễn được *vô số* điểm!. Cũng vậy, chúng ta cần có những kinh nghiệm, nhận xét từ hàng đống số liệu thống kê, nếu không, chúng ta sẽ *ngập chìm* trong *biển* thông tin như nhà bác học Karan Sing đã cảnh báo!.

Người ta thường phân loại tri thức ra làm các dạng như sau :

Tri thức sự kiện : là các khẳng định về một sự kiện, khái niệm nào đó (trong một phạm vi xác định). Các định luật vật lý, toán học, ... thường được xếp vào loại này. (Chẳng hạn : mặt trời mọc ở đằng đông, tam giác đều có 3 góc 60^0, ...)

Tri thức thủ tục : thường dùng để diễn tả phương pháp, các bước cần tiến hành, trình từ hay ngắn gọn là cách giải quyết một vấn đề. Thuật toán, thuật giải là một dạng của tri thức thủ tục.

Tri thức mô tả : cho biết một đối tượng, sự kiện, vấn đề, khái niệm, ... được thấy, cảm nhận, cấu tạo như thế nào (một cái bàn thường có 4 chân, con người có 2 tay, 2 mắt,...)

Tri thức Heuristic : là một dạng tri thức cảm tính. Các tri thức thuộc loại này thường có dạng ước lượng, phỏng đoán, và thường được hình thành thông qua kinh nghiệm.

Trên thực tế, rất hiếm có một trí tuệ mà không cần đến tri thức (liệu có thể có một đại kiện tướng cờ vua mà không biết đánh cờ hoặc không biết các thế cờ quan trọng không?). Tuy tri thức không quyết định sự thông minh (người biết nhiều định lý toán hơn chưa chắc đã giải toán giỏi hơn!) nhưng nó là một yếu tố cơ bản cấu thành trí thông minh. Chính vì vậy, muốn xây dựng một trí thông minh nhân tạo, ta cần phải có yếu tố cơ bản này. Từ đây đặt ra vấn đề đầu tiên là … Các phương pháp đưa tri thức vào máy tính được gọi là biểu diễn tri thức.

III. THUẬT TOÁN – MỘT PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC?

Trước khi trả lời câu hỏi trên, bạn hãy thử nghĩ xem, liệu một chương trình giải phương trình bậc 2 có thể được xem là một chương trình có *tri thức* hay không? ... Có chứ ! Vậy thì tri thức nằm ở đâu? Tri thức về giải phương trình bậc hai thực chất đã được mã hóa dưới dạng các câu lệnh *if..then..else* trong chương trình. Một cách tổng quát, có thể khẳng định là tất cả các chương trình máy tính ít nhiều đều đã có tri thức. Đó chính là tri thức của lập trình viên được chuyển thành các câu lệnh của chương trình. Bạn sẽ thắc mắc *"như vậy tại sao đưa tri thức vào máy tính lại là một vấn đề ? (vì từ trước tới giờ chúng ta đã, đang và sẽ tiếp tục làm như thế mà?)"*. Đúng như thế thật, nhưng vấn đề nằm ở chỗ, các tri thức trong những chương trình truyền thống là những tri thức "cứng", nghĩa là nó không thể được *thêm vào hay điều chỉnh một khi chương trình đã được biên dịch*. Muốn điều chỉnh thì chúng ta phải tiến hành sửa lại mã nguồn của chương trình (rồi sau đó biên dịch lại). Mà thao tác sửa chương trình thì chỉ có những lập trình viên mới có thể làm được. Điều này sẽ làm giảm khả năng ứng dụng chương trình (vì đa số người dùng bình thường đều không biết lập trình).

Bạn thử nghĩ xem, với một chương trình hỗ trợ ra quyết định (như đầu tư cổ phiếu, đầu tư bất động sản chẳng hạn), liệu người dùng có cảm thấy thoải mái không khi muốn đưa vào chương trình những kiến thức của mình thì anh ta phải chọn một trong hai cách là (1) *tự sửa lại mã chương trình!?* (2) *tìm tác giả của chương trình để nhờ người này sửa lại!?*. Cả hai thao tác trên đều không thể chấp nhận được đối với bất kỳ người dùng bình thường nào. Họ cần có một cách nào đó để chính họ có thể đưa tri thức vào máy tính một cách dễ dàng, thuận tiện giống như họ đang đối thoại với một con người.

Để làm được điều này, chúng ta cần phải "mềm" hóa các tri thức được biểu diễn trong máy tính. Xét cho cùng, mọi chương trình máy tính đều gồm hai thành phần là các mã lệnh và dữ liệu. Mã lệnh được ví như là phần cứng của chương trình còn dữ liệu được xem là phần mềm (vì nó có thể được thay đổi bởi người dùng). Do đó, "mềm" hóa tri thức cũng đồng nghĩa với việc tìm các phương pháp để có thể *biểu diễn các loại tri thức của con người* bằng *các cấu trúc dữ liệu* mà máy tính có thể xử lý được. Đây cũng chính là ý nghĩa của thuật ngữ "biểu diễn tri thức".

Bạn cần phải biết rằng, ít ra là cho đến thời điểm bạn đang đọc cuốn sách này, con người vẫn chưa thể tìm ra một kiểu biểu diễn tổng quát cho mọi loại tri thức!

Để làm vấn đề mà chúng ta đang bàn luận trở nên sáng tỏ hơn. Chúng ta hãy xem xét một số bài toán trong phần tiếp theo.

IV. LÀM QUEN VỚI CÁCH GIẢI QUYẾT VẤN ĐỀ BẰNG CÁCH CHUYỂN GIAO TRI THỨC CHO MÁY TÍNH

Bài toán 1 : Cho hai bình rỗng X và Y có thể tích lần lượt là VX và VY, hãy dùng hai bình này để đong ra z lít nước (z <= min(VX,VY)).

Bài toán 2 : Cho biết một số yếu tố của tam giác (như chiều dài cạnh và góc, ...). Hãy tính các yếu tố còn lại.

Bài toán 3 : Tính diện tích phần giao của các hình hình học cơ bản.

Hai bài toán đầu là hai bài toán khá tiêu biểu, thường được dùng để minh họa cho nét đẹp của phương pháp giải quyết vấn đề bài toán bằng cách chuyển giao tri thức cho máy tính. Nếu sử dụng thuật toán thông thường, chúng ta thường chỉ giải được một số trường hợp cụ thể của các bài toán này. Thậm chí, nhiều người khi mới tiếp cận với 2 bài toán này còn không tin là nó có thể hoàn toàn được giải một cách tổng quát bởi máy tính!. Bài toán số 3 là một minh họa đẹp mắt cho kỹ thuật giải quyết vấn đề "vĩ mô", nghĩa là ta chỉ cần mô tả các bước giải quyết ở mức tổng quát cho máy tính mà không cần đi vào cài đặt cụ thể.

Bài toán 1 sẽ được giải quyết bằng cách sử dụng các luật dẫn xuất (luật sinh). Bài toán 2 sẽ được giải quyết bằng mạng ngữ nghĩa và bài toán 3 sẽ giải quyết bằng công cụ frame. Ở đây chúng ta cùng nhau tìm hiểu cách giải bài toán đầu tiên. Hai bài toán kế tiếp sẽ được giải quyết lần lượt ở các mục sau.

Với một trường hợp cụ thể của bài toán 1, như VX = 5 và VY = 7 và z = 4. Sau một thời gian tính toán, bạn có thể sẽ đưa ra một quy trình đổ nước đại loại như :

Múc đầy bình 7

Trút hết qua bình 5 cho đến khi 5 đầy.

Đổ hết nước trong bình 5

Đổ hết nước còn lại từ bình 7 sang bình 5

Múc đầy bình 7

Trút hết qua bình 5 cho đến khi bình 5 đầy.

Phần còn lại chính là số nước cần đong.

Tuy nhiên, với những số liệu khác, bạn phải "mày mò" lại từ đầu để tìm ra quy trình đổ nước. Cứ thế, mỗi một trường hợp sẽ có một cách đổ nước hoàn toàn khác nhau. Như vậy, nếu có một ai đó yêu cầu bạn đưa ra một cách làm tổng quát thì chính bạn cũng sẽ lúng túng (dĩ nhiên, ngoại trừ trường hợp bạn đã biết trước cách giải theo tri thức mà chúng ta sắp sửa tìm hiểu ở đây!).

Đến đây, bạn hãy bình tâm kiểm lại cách thức bạn tìm kiếm lời giải cho một trường hợp cụ thể. Vì chưa tìm ra một quy tắc cụ thể nào, bạn sẽ thực hiện một loạt các thao tác "cảm tính" như đong đầy một bình, trút một bình này sang bình kia, đổ hết nước trong một bình ra... vừa làm vừa nhẩm tính xem cách làm này có thể đi đến kết quả hay không. Sau nhiều lần thí nghiệm, rất có thể bạn sẽ rút ra được một số kinh nghiệm như *"khi bình 7 đầy nước mà bình 5 chưa đầy thì hãy đổ nó sang bình 5 cho đến khi bình 5 đầy"*... Vậy thì tại sao bạn lại không thử "truyền" những kinh nghiệm này cho máy tính và để cho máy tính "mày mò" tìm các thao tác cho chúng ta? Điều này hoàn toàn có lợi, vì máy tính có khả năng "mày mò" hơn hẳn chúng ta! Nếu những "kinh nghiệm" mà chúng ta cung cấp cho máy tính không giúp chúng ta tìm được lời giải, chúng ta sẽ thay thế nó bằng những kinh nghiệm khác và lại tiếp tục để máy tính tìm kiếm lời giải!

Chúng ta hãy phát biểu lại bài toán một cách hình thức hơn.

Không làm mất tính tổng quát, ta luôn có thể giả sử rằng VX<VY.

Gọi lượng nước chứa trong bình X là x (0<=x<=VX)

Gọi lượng nước chứa trong bình Y là y (0<=y<=VY)

Như vậy, điều kiện kết thúc của bài toán sẽ là :

x = z hoặc y = z

Điều kiện đầu của bài toán là : x = 0 và y=0

Quá trình giải được thực hiện bằng cách xét lần lượt các luật sau, luật nào thỏa mãn thì sẽ được áp dụng. Lúc này, các luật chính là các "kinh nghiệm" hay tri thức mà ta đã chuyển giao cho máy tính. Sau khi áp dụng luật, trạng thái của bài toán sẽ thay đổi, ta lại tiếp tục xét các luật kế tiếp, nếu hết luật, quay trở lại luật đầu tiên. Quá trình tiếp diễn cho đến khi đạt được điều kiện kết thúc của bài toán.

Ba luật này được mô tả như sau :

(L1) Nếu bình X đầy thì đổ hết nước trong bình X đi.

(L2) Nếu bình Y rỗng thì đổ đầy nước vào bình Y.

(L3) Nếu bình X không đầy và bình Y không rỗng thì hãy trút nước t? bình Y sang bình X (cho đến khi bình X đầy hoặc bình Y hết nước).

> Trên thực tế, lúc đầu để giải trường hợp tổng quát của bài toán này, người ta đã dùng đến hơn 15 luật (kinh nghiệm) khác nhau. Tuy nhiên, sau này, người ta đã rút gọn lại chỉ còn 3 luật như trên.

Bạn có thể dễ dàng chuyển đổi cách giải này thành chương trình như sau :

```
...

x := 0; y := 0;

WHILE ( (x <> z) AND (y<>z) ) DO BEGIN

IF (x = Vx) THEN x := 0;

IF (y = 0) THEN (y:= Vy);

IF (y > 0) THEN BEGIN

k:= min(Vx - x, y);

x := x + k;

y := y - k;

END;
```

```
END;

...
```

Thử "chạy" chương trình trên với số liệu cụ thể là :

Vx = 3, Vy = 4 và z = 2

Ban đầu : x = 0, y = 0

Luật (L2) -> x = 0, y = 4

Luật (L3) -> x = 3, y = 1

Luật (L1) -> x = 0, y = 1

Luật (L3) -> x = 1, y = 0

Luật (L2) -> x = 1, y = 4

Luật (L3) -> x = 3, y = **2**

3 luật mà chúng ta đã cài đặt trong chương trình ở trên được gọi là **cơ sở tri thức.** Còn cách thức tìm kiếm lời giải bằng cách duyệt tuần tự từng luật và áp dụng nó được gọi là **động cơ suy diễn.** Chúng ta sẽ định nghĩa chính xác hai thuật ngữ này ở cuối mục.

Người ta đã chứng minh được rằng, bài toán đong nước chỉ có lời giải khi số nước cần đong là một bội số của ước số chung lớn nhất của thể tích hai bình. z = n ▉USCLN(VX, VY) (với n nguyên dương)

Cách giải quyết vấn đề theo kiểu này khác so với cách giải bằng thuật toán thông thường là chúng ta *không đưa ra một trình tự giải quyết vấn đề cụ thể* mà chỉ đưa ra các quy tắc chung chung (dưới dạng các luật), máy tính sẽ dựa vào đó (áp dụng các luật) để *tự xây dựng* một quy trình giải quyết vấn đề. Điều này cũng giống như việc chúng ta giải toán bằng cách đưa ra các định lý, quy tắc liên quan đến bài toán mà không cần phải chỉ ra cách giải cụ thể.

Vậy thì điểm thú vị nằm ở điểm nào? Bạn sẽ có thể cảm thấy rằng chúng ta vẫn đang dùng tri thức "cứng" ! (vì các tri thức vẫn là các câu lệnh IF được cài sẵn trong chương trình). Thực ra thì chương trình của chúng ta đã "mềm" hơn một tí rồi đấy. Nếu không tin, các bạn hãy quan sát phiên bản kế tiếp của chương trình này.

```
FUNCTION DK(L INTEGER):BOOLEAN;

BEGIN

CASE L OF

1 : DK := (x = Vx);

2 : DK := (y = 0);

3 : DK := (y>0);
```

```
END;

END;

PROCEDURE ThiHanh(L INTEGER):BOOLEAN;

BEGIN

CASE L OF

1 : x := 0;

2: y := Vy;

3 : BEGIN

k := min(Vx-x,y);

x := x+k;

y := y-k;

END;

END;

END;

CONST SO_LUAT = 3;

BEGIN

WHILE (x<>z) AND (y<>z) DO BEGIN

FOR i:=1 TO SO_LUAT DO

IF DK(L) THEN ThiHanh(L);

END;

END.
```

Đoạn chương trình chính cũng thi hành bằng cách lần lượt xét qua 3 lệnh IF như chương trình đầu tiên. Tuy nhiên, ở đây, biểu thức điều kiện được thay thế bằng hàm DK và các hành động ứng với điều kiện đã được thay thế bằng thủ tục ThiHanh. Tính chất "mềm" hơn của chương trình này thể hiện ở chỗ, nếu muốn bổ sung "tri thức", ta chỉ phải điều chỉnh lại các hàm DK và ThiHanh mà không cần phải **sửa lại chương trình chính**.

Bây giờ hãy giả sử rằng ta đã có hàm và thủ tục đặc biệt sau :

FUNCTION GiaTriBool(DK : String) : BOOLEAN;

PROCEDURE ThucHien(ThaoTac : String) ;

hàm GiaTriBool nhận vào một **chuỗi** điều kiện, nó sẽ phân tích chuỗi, tính toán rồi trả ra giá trị BOOLEAN của biểu thức này.

Ví dụ : GiaTriBoolean(‘6<7’) sẽ trả ra FALSE

Thủ tục ThucHien cũng nhận vào một chuỗi, nó cũng sẽ phân tích chuỗi rồi tiến hành thực hiện những hành động được miêu tả trong chuỗi này.

Với hàm và thủ tục này, chương trình của chúng ta sẽ như sau :

```
CONST SO_LUAT = 3;

TYPE

Luat RECORD

DK : String;

ThiHanh : String;

END;

DSLuat ARRAY [1..SO_LUAT] OF Luat; 9;

VAR

CacLuat DSLuat;

PROCEDURE KhoiDong;

BEGIN

CacLuat[1].DK := ‘x = Vx’;

CacLuat[2].DK := ‘y = 0’;

CacLuat[3].DK := ‘y>0’; 9;

CacLuat[1].ThaoTac := ‘x:=0’;

CacLuat[2].ThaoTac:= ‘y:=Vy’;

CacLuat[3].ThaoTac:= ‘k:=min(Vx-x,y), x:=x+k, y:=y-k’;

END;

BEGIN

WHILE (x<>z) AND (y<>z) DO BEGIN

FOR i:=1 TO SO_LUAT DO

IF GiaTriBoolean(CacLuat[i].DK)

THEN ThucHien(CacLuat[i].ThaoTac);

END;

END.
```

Chúng ta tạm cho rằng trong quá trình chương trình thi hành, ta có thể dễ dàng thay đổi số phần tử mảng CacLuat (các ngôn ngữ lập trình sau này như Visual C++, Delphi đều cho phép điều này). Với chương trình này, khi muốn sửa đổi "tri thức", bạn chỉ cần thay đổi **giá trị mảng Luat** là xong.

Tuy nhiên, người dùng vẫn gặp khó khăn khi muốn bổ sung hoặc hiệu chỉnh tri thức. Họ cần phải nhập các chuỗi đại loại như ‘x=0’ hoặc ‘k:=min(Vx-x,y)’ ...Các chuỗi này, tuy có ý nghĩa đối với chương trình nhưng vẫn còn khá xa lạ đối với người dùng bình thường. Chúng ta cần giảm bớt "khoảng cách" này lại bằng cách đưa ra những chuỗi điều kiện hoặc thao tác có **ý nghĩa trực tiếp** đối với người dùng. Chương trình sẽ có **chuyển đổi** lại các điều kiện và thao tác này sang dạng phù hợp với chương trình.

Để làm được điều trên. Chúng ta cần phải liệt kê được các trạng thái và thao tác cơ bản của bài toán này. Sau đây là một số trạng thái và thao tác cơ bản.

Trạng thái cơ bản :

Bình X đầy, Bình X rỗng, Bình X không rỗng, Bình X có n lít nước.

Thao tác

Đổ hết nước trong bình, Đổ đầy nước trong bình, Đổ nước từ bình A sang bình B cho đến khi B đầy hoặc A rỗng.

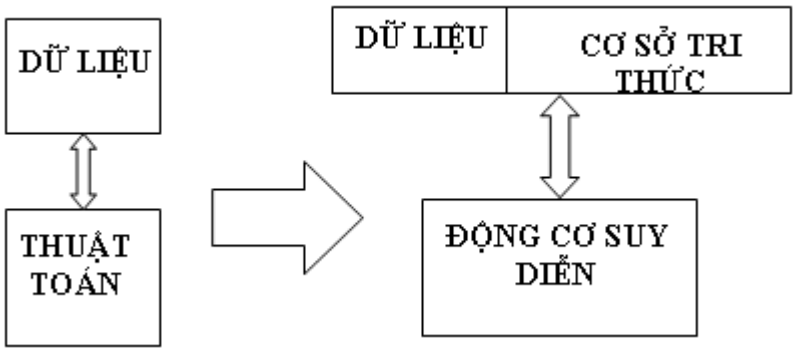
> Lưu ý rằng ta không thể có thao tác "Đổ n lít nước từ A sang B" vì bài toán đã giả định rằng các bình đều không có vạch chia, hơn nữa nếu ta biết cách đổ n lít nước từ A sang B thì lời giải bài toán trở thành quá đơn giản.
>
> "Múc đầy X"
>
> "Đổ z lít nước từ X sang Y"
>
> Vì đây là một bài toán đơn giản nên bạn có thể dễ nhận thấy rằng, các trạng thái cơ bản và thao tác chẳng có gì khác so với các điều kiện mà chúng ta đã đưa ra.

Kế tiếp, ta sẽ viết các đoạn chương trình cho phép người dùng nhập vào các luật (dạng nếu ... thì ...) được hình thành từ các trạng thái và điều kiện cơ bản này, đồng thời tiến hành chuyển sang dạng máy tính có thể xử lý được như ở ví dụ trên. Chúng ta sẽ không bàn đến việc cài đặt các đoạn chương trình giao tiếp với người dùng ở đây.

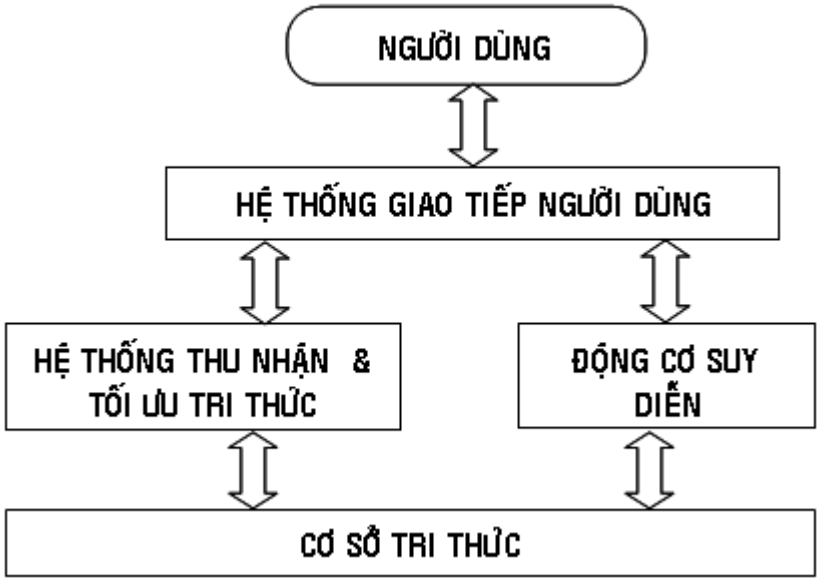
Như vậy, so với chương trình truyền thống (được cấu tạo từ hai "chất liệu" cơ bản là ***dữ liệu*** và ***thuật toán***), chương trình trí tuệ nhân tạo được cấu tạo từ hai thành phần là ***cơ sở tri thức*** (knowledge base) và ***động cơ suy diễn*** *(inference engine).*

Cơ sở tri thức : là tập hợp các tri thức liên quan đến vấn đề mà chương trình quan tâm giải quyết.

Động cơ suy diễn : là phương pháp vận dụng tri thức trong cơ sở tri thức để giải quyết vấn đề.



Nếu xét theo quan niệm biểu diễn tri thức mà ta vừa bàn luận ở trên thì cơ sở tri thức chỉ là một dạng dữ liệu đặc biệt và động cơ suy diễn cũng chỉ là một dạng của thuật toán đặc biệt mà thôi. Tuy vậy, có thể nói rằng, cơ sở tri thức và động cơ suy diễn là một bước tiến hóa mới của dữ liệu và thuật toán của chương trình! Bạn có thể hình dung *động cơ suy diễn* giống như một loại động cơ *tổng quát, được chuẩn hóa* có thể dùng để vận hành nhiều loại xe máy khác nhau và *cơ sở tri thức* chính là loại nhiên liệu đặc biệt để vận hành loại động cơ này !



Cơ sở tri thức cũng gặp phải những vấn đề tương tự như những cơ sở dữ liệu khác như sự trùng lắp, thừa, mâu thuẫn. Khi xây dựng cơ sở tri thức, ta cũng phải chú ý đến những yếu tố này. Như vậy, bên cạnh vấn đề biểu diễn tri thức, ta còn phải đề ra các phương pháp để loại bỏ những tri thức trùng lắp, thừa hoặc mâu thuẫn. Những thao tác này sẽ được thực hiện trong quá trình ghi nhận tri thức vào hệ thống. Chúng ta sẽ đề cập đến những phương pháp này trong phần tìm hiểu về các luật dẫn.

Hình ảnh trên tóm tắt cho chúng ta thấy cấu trúc chung nhất của một chương trình trí tuệ nhân tạo.

B. CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC TRÊN MÁY TÍNH

V. LOGIC MỆNH ĐỀ

Đây có lẽ là kiểu biểu diễn tri thức đơn giản nhất và gần gũi nhất đối với chúng ta. Mệnh đề là một khẳng định, một phát biểu mà giá trị của nó chỉ có thể hoặc là đúng hoặc là sai.

Ví dụ :

phát biểu "1+1=2" có giá trị đúng.

phát biểu "Mọi loại cá có thể sống trên bờ" có giá trị sai.

Giá trị của mệnh đề không chỉ phụ thuộc vào bản thân mệnh đề đó. Có những mệnh đề mà giá trị của nó luôn đúng hoặc sai bất chấp thời gian nhưng cũng có những mệnh đề mà giá trị của nó lại phụ thuộc vào thời gian, không gian và nhiều yếu tố khác quan khác. Chẳng hạn như mệnh đề : "Con người không thể nhảy cao hơn 5m với chân trần" là đúng khi ở trái đất , còn ở những hành tinh có lực hấp dẫn yếu thì có thể sai.

Ta ký hiệu mệnh đề bằng những chữ cái la tinh như **a, b, c, ...**

Có 3 phép nối cơ bản để tạo ra những mệnh đề mới từ những mệnh đề cơ sở là phép hội ([redacted], giao([redacted]) và phủ định ([redacted]

Bạn đọc chắn hẳn đã từng sử dụng logic mệnh đề trong chương trình rất nhiều lần (như trong cấu trúc lệnh IF ... THEN ... ELSE) để biểu diễn các tri thức "cứng" trong máy tính !

Bên cạnh các thao tác tính ra giá trị các mệnh đề phức từ giá trị những mệnh đề con, chúng ta có được một cơ chế suy diễn như sau :

Modus Ponens : Nếu mệnh đề A là đúng và mệnh đề A[redacted] là đúng thì giá trị của B sẽ là đúng.

Modus Tollens : Nếu mệnh đề A[redacted] là đúng và mệnh đề B là sai thì giá trị của A sẽ là sai.

Các phép toán và suy luận trên mệnh đề đã được đề cập nhiều đến trong các tài liệu về toán nên chúng ta sẽ không đi vào chi tiết ở đây.

VI. LOGIC VỊ TỪ

Biểu diễn tri thức bằng mệnh đề gặp phải một trở ngại cơ bản là ta không thể can thiệp vào cấu trúc của một mệnh đề. Hay nói một cách khác là mệnh đề *không có cấu trúc* . Điều này làm hạn chế rất nhiều thao tác suy luận . Do đó, người ta đã đưa vào khái niệm vị từ và lượng từ ([redacted] với mọi, [redacted] tồn tại) để tăng cường tính cấu trúc của một mệnh đề.

Trong logic vị từ, một mệnh đề *được cấu tạo bởi* hai thành phần là các *đối tượng tri thức* và *mối liên hệ giữa chúng (gọi là vị từ).* Các mệnh đề sẽ được biểu diễn dưới dạng :

Vị từ (<đối tượng 1>, <đối tượng 2>, …, <đối tượng n>)

Như vậy để biểu diễn vị của các trái cây, các mệnh đề sẽ được viết lại thành :

Cam có vị Ngọt [redacted]ị (Cam, Ngọt)

Cam có màu Xanh [redacted]àu (Cam, Xanh)

...

Kiểu biểu diễn này có hình thức tương tự như hàm trong các ngôn ngữ lập trình, các đối tượng tri thức chính là các tham số của hàm, giá trị mệnh đề chính là kết quả của hàm (thuộc kiểu BOOLEAN).

Với vị từ, ta có thể biểu diễn các tri thức dưới dạng các mệnh đề tổng quát, là những mệnh đề mà giá trị của nó được xác định thông qua các đối tượng tri thức cấu tạo nên nó.

Chẳng hạn tri thức : *"A là bố của B nếu B là anh hoặc em của một người con của A"* có thể được biểu diễn dưới dạng vị từ như sau :

Bố (A, B) = Tồn tại Z sao cho : Bố (A, Z) và (Anh(Z, B) hoặc Anh(B,Z))

Trong trường hợp này, mệnh đề Bố(A,B) là một mệnh đề tổng quát

Như vậy nếu ta có các mệnh đề cơ sở là :

a) Bố ("An", "Bình") có giá trị đúng (Anh là bố của Bình)

b) Anh("Tú", "Bình") có giá trị đúng (Tú là anh của Bình)

thì mệnh đề **c)** Bố ("An", "Tú") sẽ có giá trị là đúng. (An là bố của Tú).

Rõ ràng là nếu chỉ sử dụng logic mệnh đề thông thường thì ta sẽ không thể tìm được một mối liên hệ nào giữa c và a,b bằng các phép nối mệnh đề [redacted]Từ đó, ta cũng không thể tính ra được giá trị của mệnh đề **c.** Sở dĩ như vậy vì ta không thể thể hiện tường minh tri thức *"(A là bố của B) nếu có Z sao cho (A là bố của Z) và (Z anh hoặc em C)"* dưới dạng các mệnh đề thông thường. Chính đặc trưng của vị từ đã cho phép chúng ta thể hiện được các tri thức dạng tổng quát như trên.

Thêm một số ví dụ nữa để các bạn thấy rõ hơn khả năng của vị từ :

Câu cách ngôn "Không có vật gì là lớn nhất và không có vật gì là bé nhất!" có thể được biểu diễn dưới dạng vị từ như sau :

LớnHơn(x,y) = x>y

NhỏHơn(x,y) = x<y

[redacted] : LớnHơn(y,x) và [redacted] : NhỏHơn(y,x)

Câu châm ngôn "Gần mực thì đen, gần đèn thì sáng" được hiểu là "chơi với bạn xấu nào thì ta cũng sẽ thành người xấu" có thể được biểu diễn bằng vị từ như sau :

NgườiXấu (x) = [redacted] : Bạn(x,y) và NgườiXấu(y)

Công cụ vị từ đã được nghiên cứu và phát triển thành một ngôn ngữ lập trình đặc trưng cho trí tuệ nhân tạo. Đó là ngôn ngữ PROLOG. Phần đọc thêm của chương sẽ giới thiệu tổng quan với các bạn về ngôn ngữ này.

VII. MỘT SỐ THUẬT GIẢI LIÊN QUAN ĐẾN LOGIC MỆNH ĐỀ

Một trong những vấn đề khá quan trọng của logic mệnh đề là chứng minh tính đúng đắn của phép suy diễn (a [redacted]. Đây cũng chính là bài toán chứng minh thường gặp trong toán học.

Rõ ràng rằng với hai phép suy luận cơ bản của logic mệnh đề (Modus Ponens, Modus Tollens) cộng với các phép biến đổi hình thức, ta cũng có thể chứng minh được phép suy diễn. Tuy nhiên, thao tác biến đối hình thức là rất khó cài đặt được trên máy tính. Thậm chí điều này còn khó khăn với cả con người!

Với công cụ máy tính, bạn có thể cho rằng ta sẽ dễ dàng chứng minh được mọi bài toán bằng một phương pháp "thô bạo" là lập bảng chân trị . Tuy về lý thuyết, phương pháp lập bảng chân trị luôn cho được kết quả cuối cùng nhưng độ phức tạp của phương pháp này là quá lớn, $O(2^n)$ với n là số biến mệnh đề. Sau đây chúng ta sẽ nghiên cứu hai phương pháp chứng minh mệnh đề với độ phức tạp chỉ có O(n).

VII.1. Thuật giải Vương Hạo

B1 : Phát biểu lại giả thiết và kết luận của vấn đề theo dạng chuẩn sau :

$GT_1, GT_2, ..., GTn$ [redacted] $L_1, KL_2, ..., KLm$

Trong đó các GTi và KLi là các mệnh đề được xây dựng từ các biến mệnh đề và 3 phép nối cơ bản : [redacted]

B2 : Chuyển vế các GTi và KLi có dạng phủ định.

Ví dụ :

p [redacted], [redacted] r [redacted]), [redacted], p [redacted] [redacted]

[redacted], p [redacted], p [redacted] [redacted]), g, s

B3 : Nếu GTi có phép [redacted] thì thay thế phép [redacted] bằng dấu ","

Nếu KLi có phép [redacted] thì thay thế phép [redacted] bằng dấu ","

Ví dụ :

p [redacted], r [redacted] [redacted]) [redacted], [redacted]

[redacted] q, r, [redacted] [redacted] [redacted], [redacted]

B4 : Nếu GTi có phép [redacted] thì tách thành hai dòng con.

Nếu ở KLi có phép [redacted] thì tách thành hai dòng con.

Ví dụ :

p, [redacted] [redacted] [redacted]

p, [redacted] [redacted] p, q [redacted]

B5 : Một dòng được chứng minh nếu tồn tại chung một mệnh đề ở ở

cả hai phía.

Ví dụ :

p, **q** [redacted] được chứng minh

p, [redacted] q

B6 :

a) Nếu một dòng không còn phép nối [redacted] hoặc [redacted] cả hai vế và ở 2 vế không có chung một biến mệnh đề thì dòng đó không được chứng minh.

b) Một vấn đề được chứng minh nếu tất cả dòng dẫn xuất từ dạng chuẩn ban đầu đều được chứng minh.

VII.2 Thuật giải Robinson

Thuật giải này hoạt động dựa trên phương pháp chứng minh phản chứng.

Phương pháp chứng minh phản chứng

Chứng minh phép suy luận (a [redacted] là đúng (với a là giả thiết, b là kết luận).

Phản chứng : giả sử b sai suy ra [redacted] là đúng.

Bài toán được chứng minh nếu a đúng và [redacted] đúng sinh ra một mâu thuẫn.

B1 : Phát biểu lại giả thiết và kết luận của vấn đề dưới dạng chuẩn như sau :

GT_1, GT_2, ...,GTn [redacted] L_1, KL_2, .., KLm

Trong đó : GTi và KLj được xây dựng từ các biến mệnh đề và các phép toán : [redacted]

B2 : Nếu GTi có phép [redacted] thì thay bằng dấu ","

Nếu KLi có phép [redacted] thì thay bằng dấu ","

B3 : Biến đổi dòng chuẩn ở B1 về thành danh sách mệnh đề như sau :

{ GT_1, GT_2, ..., GTn , [redacted] L_1, [redacted] L_2, ..., [redacted] Lm }

B4 : Nếu trong danh sách mệnh đề ở bước 2 có 2 mệnh đề đối ngẫu nhau thì bài toán được chứng minh. Ngược lại thì chuyển sang B4. (a và [redacted] gọi là hai mệnh đề đối ngẫu nhau)

B5 : Xây dựng một mệnh đề mới bằng cách tuyển một cặp mệnh đề trong danh sách mệnh đề ở bước 2. Nếu mệnh đề mới có các biến

mệnh đề đối ngẫu nhau thì các biến đó được loại bỏ.

Ví dụ : &#p [redacted]

Hai mệnh đề [redacted]**, q** là đối ngẫu nên sẽ được loại bỏ

[redacted]

B6 : Thay thế hai mệnh đề vừa tuyển trong danh sách mệnh đề bằng mệnh đề mới.

Ví dụ :

{ p [redacted] , [redacted] , w [redacted], s [redacted] }

[redacted] , w [redacted], s [redacted] }

B7 : Nếu không xây dựng được thêm một mệnh đề mới nào và trong danh sách mệnh đề không có 2 mệnh đề nào đối ngẫu nhau thì vấn đề không được chứng minh.

Ví dụ : Chứng minh rằng

[redacted]

B3: { [redacted] **p, u** }

B4 : Có tất cả 6 mệnh đề nhưng chưa có mệnh đề nào đối ngẫu nhau.

B5 : [redacted]yển một cặp mệnh đề (chọn hai mệnh đề có biến đối ngẫu). Chọn hai mệnh đề đầu :

[redacted]

Danh sách mệnh đề thành :

{[redacted], p, u }

Vẫn chưa có mệnh đề đối ngẫu.

Tuyển hai cặp mệnh đề đầu tiên

[redacted]

Danh sách mệnh đề thành {[redacted], p, u }

Vẫn chưa có hai mệnh đề đối ngẫu

Tuyển hai cặp mệnh đề đầu tiên

[redacted]

Danh sách mệnh đề thành : {[redacted], p, u }

Vẫn chưa có hai mệnh đề đối ngẫu

Tuyển hai cặp mệnh đề :

[redacted]

Danh sách mệnh đề trở thành : {[redacted], p }

Có hai mệnh đề đối ngẫu nên biểu thức ban đầu đã được chứng minh.

VIII. BIỂU DIỄN TRI THỨC SỬ DỤNG LUẬT DẪN XUẤT (LUẬT SINH)

VIII.1. Khái niệm

Phương pháp biểu diễn tri thức bằng luật sinh được phát minh bởi Newell và Simon trong lúc hai ông đang cố gắng xây dựng một hệ giải bài toán tổng quát. Đây là một kiểu biểu diễn tri thức có cấu trúc. Ý tưởng cơ bản là tri thức có thể được cấu trúc bằng một cặp **điều kiện – hành động** : "NẾU *điều kiện* xảy ra THÌ *hành động* sẽ được thi hành". Chẳng hạn : NẾU đèn giao thông là đỏ THÌ bạn không được đi thẳng, NẾU máy tính đã mở mà không khởi động được THÌ kiểm tra nguồn điện, …

Ngày nay, các luật sinh đã trở nên phổ biến và được áp dụng rộng rãi trong nhiều hệ thống trí tuệ nhân tạo khác nhau. Luật sinh có thể là một công cụ mô tả để giải quyết các vấn đề thực tế thay cho các kiểu phân tích vấn đề truyền thống. Trong trường hợp này, các luật được dùng như là những chỉ dẫn (tuy có thể không hoàn chỉnh) nhưng rất hữu ích để trợ giúp cho các quyết định trong quá trình tìm kiếm, từ đó làm giảm không gian tìm kiếm. Một ví dụ khác là luật sinh có thể được dùng để bắt chước hành vi của những chuyên gia. Theo cách này, luật sinh không chỉ đơn thuần là một kiểu biểu diễn tri thức trong máy tính mà là một kiểu biểu diễn các hành vi của con người.

Một cách tổng quát luật sinh có dạng như sau :

P_1 [redacted] P_2 [redacted]. [redacted]Pn [redacted]

Tùy vào các vấn đề đang quan tâm mà luật sinh có những ngữ nghĩa hay cấu tạo khác nhau :

Trong logic vị từ : P_1, P_2, ..., Pn, Q là những biểu thức logic.

Trong ngôn ngữ lập trình, mỗi một luật sinh là một câu lệnh.

IF (P_1 AND P_2 AND .. AND Pn) THEN Q.

Trong lý thuyết hiểu ngôn ngữ tự nhiên, mỗi luật sinh là một phép dịch :

ONE [redacted]ột.

TWO [redacted]i.

JANUARY [redacted]áng một

Để biểu diễn một tập luật sinh, người ta thường phải chỉ rõ hai thành phần chính sau :

(1) Tập các sự kiện F(Facts)

F = { f_1, f_2, ... fn }

(2) Tập các quy tắc R (Rules) áp dụng trên các sự kiện dạng như sau :

f_1 ^ f_2 ^ ... ^ fi [redacted]

Trong đó, các fi , q đều thuộc F

Ví dụ : Cho 1 cơ sở tri thức được xác định như sau :

Các sự kiện : A, B, C, D, E, F, G, H, K

Tập các quy tắc hay luật sinh (rule)

R1 : A [redacted]

R2 : B [redacted]

R3 : H [redacted]

R4 : E [redacted]

R5 : E [redacted]

R6 : D [redacted]

R7 : G [redacted]

VIII.2. Cơ chế suy luận trên các luật sinh

Suy diễn tiến : là quá trình suy luận xuất phát từ một số sự kiện ban đầu, xác định các sự kiện có thể được "sinh" ra từ sự kiện này.

Sự kiện ban đầu : H, K

R3 : H [redacted] {A, H. K }

R1 : A [redacted] { A, E, H, H }

R5 : E [redacted] { A, B, E, H, K }

R2 : B [redacted] { A, B, D, E, H, K }

R6 : D [redacted] { A, B, C, D, E, H, K }

Suy diễn lùi : là quá trình suy luận ngược xuất phát từ một số sự kiện ban đầu, ta tìm kiếm các sự kiện đã "sinh" ra sự kiện này. Một ví dụ thường gặp trong thực tế là xuất phát từ các tình trạng của máy tính, chẩn đoán xem máy tính đã bị hỏng hóc ở đâu.

Ví dụ :

Tập các sự kiện :

- Ổ cứng là "hỏng" hay "hoạt động bình thường"
- Hỏng màn hình.
- Lỏng cáp màn hình.
- Tình trạng đèn ổ cứng là "tắt" hoặc "sáng"
- Có âm thanh đọc ổ cứng.
- Tình trạng đèn màn hình "xanh" hoặc "chớp đỏ"
- Không sử dụng được máy tính.
- Điện vào máy tính "có" hay "không"

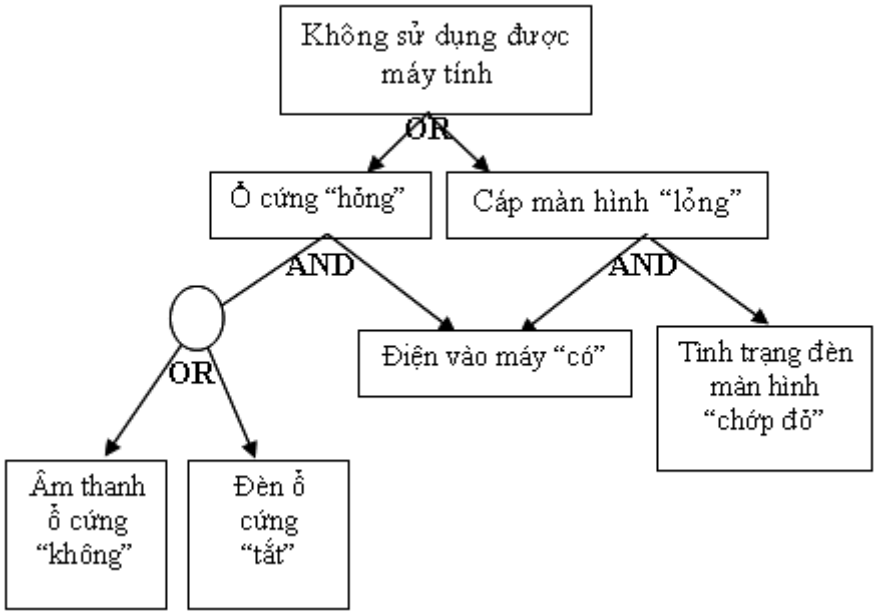
Tập các luật :

R1. Nếu ((ổ cứng "hỏng") hoặc (cáp màn hình "lỏng")) thì không sử dụng được máy tính.

R2. Nếu (điện vào máy là "có") và ((âm thanh đọc ổ cứng là "không") hoặc tình trạng đèn ổ cứng là "tắt")) thì (ổ cứng "hỏng").

R3. Nếu (điện vào máy là "có") và (tình trạng đèn màn hình là "chớp đỏ") thì (cáp màn hình "lỏng").

Để xác định được các nguyên nhân gây ra sự kiện "không sử dụng được máy tính", ta phải xây dựng một cấu trúc đồ thị gọi là đồ thị AND/OR như sau :



Như vậy là để xác định được nguyên nhân gây ra hỏng hóc là do ổ cứng hỏng hay cáp màn hình lỏng, hệ thống phải lần lượt đi vào các nhánh để kiểm tra các điều kiện như điện vào máy "có", âm thanh ổ cứng "không"…Tại một bước, nếu giá trị cần xác định không thể được suy ra từ bất kỳ một luật nào, hệ thống sẽ yêu cầu người dùng trực tiếp nhập vào. Chẳng hạn như để biết máy tính có điện không, hệ thống sẽ hiện ra màn hình câu hỏi *"Bạn kiểm tra xem có điện vào máy tính không (kiểm tra đèn nguồn)? (C/K)"*. Để thực hiện được cơ chế suy luận lùi, người ta thường sử dụng ngăn xếp (để ghi nhận lại những nhánh chưa kiểm tra).

VIII.3. Vấn đề tối ưu luật

Tập các luật trong một cơ sở tri thức rất có khả năng thừa, trùng lắp hoặc mâu thuẫn. Dĩ nhiên là hệ thống có thể *đổ lỗi* cho người dùng về việc đưa vào hệ thống những tri thức như vậy. Tuy việc tối ưu một cơ sở tri thức về mặt tổng quát là một thao tác khó (vì giữa các tri thức thường có quan hệ không tường minh), nhưng trong giới hạn cơ sở tri thức dưới dạng luật, ta vẫn có một số thuật toán đơn giản để loại bỏ các vấn đề này.

VIII.3.1. Rút gọn bên phải

Luật sau hiển nhiên đúng :

A [redacted] (1)

Do đó luật

A [redacted]

Là hoàn toàn tương đương với

A [redacted]

Quy tắc rút gọn : Có thể loại bỏ những sự kiện bên vế phải nếu những sự kiện đó đã xuất hiện bên vế trái. Nếu sau khi rút gọn mà vế phải trở thành rỗng thì luật đó là luật hiển nhiên. Ta có thể loại bỏ các luật hiển nhiên ra khỏi tri thức.

VIII.3.2. Rút gọn bên trái

Xét các luật :

(L1) A, B [redacted] (L2) A [redacted] (L3) X [redacted]

Rõ ràng là luật A, B [redacted] có thể được thay thế bằng luật A [redacted] mà không làm ảnh hưởng đến các kết luận trong mọi trường hợp. Ta nói rằng sự kiện B trong luật (1) là dư thừa và có thể được loại bỏ khỏi luật dẫn trên.

VIII.3.3. Phân rã và kết hợp luật

Luật A [redacted]

Tương đương với hai luật

A [redacted]

B [redacted]

Với quy tắc này, ta có thể loại bỏ hoàn toàn các luật có phép nối HOẶC. Các luật có phép nối này thường làm cho thao tác xử lý trở nên phức tạp.

VIII.3.4. Luật thừa

Một luật dẫn A [redacted] được gọi là thừa nếu có thể suy ra luật này từ những luật còn lại.

Ví dụ : trong tập các luật gồm {A [redacted], B [redacted], A [redacted]} thì luật thứ 3 là luật thừa vì nó có thể được suy ra từ 2 luật còn lại.

VIII.3.5. Thuật toán tối ưu tập luật dẫn

Thuật toán này sẽ tối ưu hóa tập luật đã cho bằng cách loại đi các luật có phép nối HOẶC, các luật hiển nhiên hoặc các luật thừa.

Thuật toán bao gồm các bước chính

B1 : Rút gọn vế phải

Với mỗi luật r trong R

Với mỗi sự kiện A [redacted] VếPhải(r)

Nếu A [redacted] VếTrái(r) thì Loại A ra khỏi vế phải của R.

Nếu VếPhải(r) rỗng thì loại bỏ r ra khỏi hệ luật dẫn : R = R – {r}

B2 : Phân rã các luật

Với mỗi luật r : X_1 [redacted] X_2 [redacted] .. [redacted] Xn [redacted] trong R

Với mỗi i từ 1 đến n R := R + { Xi [redacted] }

R := R – {r}

B3 : Loại bỏ luật thừa

Với mỗi luật r thuộc R

Nếu VếPhải(r) [redacted] BaoĐóng(VếTrái(r), R-{r}) thì R := R – {r}

B4 : Rút gọn vế trái

Với mỗi luật dẫn r : X : A_1 [redacted] A_2, …, An [redacted] thuộc R

Với mỗi sự kiện Ai thuộc r

Gọi luật r_1 : X – Ai [redacted]

S = (R – {r}) [redacted] r_1}

Nếu BaoĐóng(X – Ai , S) [redacted] BaoĐóng(X, R) thì loại sự kiện A ra khỏi X

VIII.4. Ưu điểm và nhược điểm của biểu diễn tri thức bằng luật

Ưu điểm

Biểu diễn tri thức bằng luật đặc biệt hữu hiệu trong những tình huống hệ thống cần đưa ra những hành động dựa vào những sự kiện có thể quan sát được. Nó có những ưu điểm chính yếu sau đây :

Các luật rất dễ hiểu nên có thể dễ dàng dùng để trao đổi với người dùng (vì nó là một trong những dạng tự nhiên của ngôn ngữ).

Có thể dễ dàng xây dựng được cơ chế suy luận và giải thích từ các luật.

Việc hiệu chỉnh và bảo trì hệ thống là tương đối dễ dàng.

Có thể cải tiến dễ dàng để tích hợp các luật mờ.

Các luật thường ít phụ thuộc vào nhau.

Nhược điểm

Các tri thức phức tạp đôi lúc đòi hỏi quá nhiều (hàng ngàn) luật sinh. Điều này sẽ làm nảy sinh nhiều vấn đề liên quan đến tốc độ lẫn quản trị hệ thống.

Thống kê cho thấy, người xây dựng hệ thống trí tuệ nhân tạo thích sử dụng luật sinh hơn tất cả phương pháp khác (dễ hiểu, dễ cài đặt) nên họ thường tìm mọi cách để biểu diễn tri thức bằng luật sinh cho dù có phương pháp khác thích hợp hơn! Đây là nhược điểm mang tính chủ quan của con người.

Cơ sở tri thức luật sinh lớn sẽ làm giới hạn khả năng tìm kiếm của chương trình điều khiển. Nhiều hệ thống gặp khó khăn trong việc đánh giá các hệ dựa trên luật sinh cũng như gặp khó khăn khi suy luận trên luật sinh.

X. BIỂU DIỄN TRI THỨC SỬ DỤNG MẠNG NGỮ NGHĨA

X.1. Khái niệm

Mạng ngữ nghĩa là một phương pháp biểu diễn tri thức đầu tiên và cũng là phương pháp dễ hiểu nhất đối với chúng ta. Phương pháp này sẽ biểu diễn tri thức dưới dạng một đồ thị, trong đó đỉnh là các đối tượng (khái niệm) còn các cung cho biết mối quan hệ giữa các đối tượng (khái niệm) này.

Chẳng hạn : giữa các khái niệm *chích chòe, chim, hót, cánh, tổ* có một số mối quan hệ như sau :

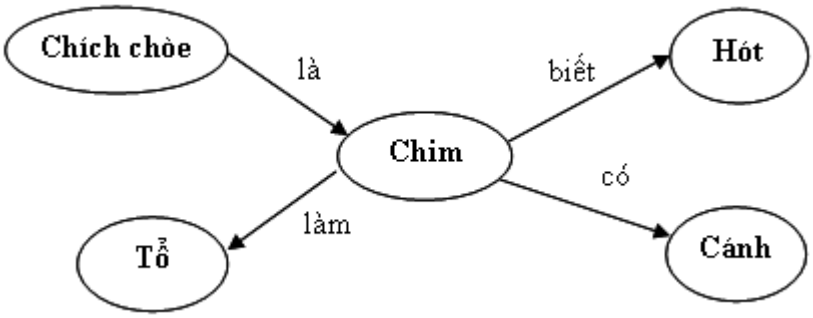
Chích chòe là một loài chim.

Chim biết hót

Chim có cánh

Chim sống trong tổ

Các mối quan hệ này sẽ được biểu diễn trực quan bằng một đồ thị như sau :



Do mạng ngữ nghĩa là một loại đồ thị cho nên nó thừa hưởng được tất cả những mặt mạnh của công cụ này. Nghĩa là ta có thể dùng những thuật toán của đồ thị trên mạng ngữ nghĩa như thuật toán tìm liên thông, tìm đường đi ngắn nhất,… để thực hiện các cơ chế suy luận. Điểm đặc biệt của mạng ngữ nghĩa so với đồ thị thông thường chính là việc gán một ý nghĩa (*có, làm, là, biết, ...*) cho các cung. Trong đồ thị tiêu chuẩn, việc có một cung nối giữa hai đỉnh chỉ cho biết có sự *liên hệ* giữa hai đỉnh đó và tất cả các cung trong đồ thị đều biểu diễn cho cùng một loại liên hệ. Trong mạng ngữ nghĩa, cung nối giữa hai đỉnh còn cho biết giữa hai khái niệm tương ứng có sự liên hệ *như thế nào*. Việc gán ngữ nghĩa vào các cung của đồ thị đã giúp giảm bớt được số lượng đồ thị cần phải dùng để biểu diễn các mối liên hệ giữa các khái niệm. Chẳng hạn như trong ví dụ trên, nếu sử dụng đồ thị thông thường, ta phải dùng đến 4 loại đồ thị cho 4 mối liên hệ : một đồ thị để biểu diễn mối liên hệ "*là"*, một đồ thị cho mối liên hệ "*làm"*, một cho "*biết"* và một cho "*có"*.

Một điểm khá thú vị của mạng ngữ nghĩa là tính kế thừa. Bởi vì ngay từ trong khái niệm, mạng ngữ nghĩa đã hàm ý sự phân cấp (như các mối liên hệ *"là"*) nên có nhiều đỉnh trong mạng mặc nhiên sẽ có những thuộc tính của những đỉnh khác. Chẳng hạn theo mạng ngữ nghĩa ở trên, ta có thể dễ dàng trả lời "có" cho câu hỏi : "Chích chòe có làm tổ không?". Ta có thể khẳng định được điều này vì đỉnh "chích chòe" có liên kết "là" với đỉnh "chim" và đỉnh "chim" lại liên kết "biết" với đỉnh "làm tổ" nên suy ra đỉnh "chích chòe" cũng có liên kết loại "biết" với đỉnh "làm tổ". (Nếu để ý, bạn sẽ nhận ra được kiểu *"suy luận"* mà ta vừa thực hiện bắt nguồn từ thuật toán "loang" hay "tìm liên thông" trên đồ thị!). Chính đặc tính kế thừa của mạng ngữ nghĩa đã cho phép ta có thể thực hiện được rất nhiều phép suy diễn từ những thông tin sẵn có trên mạng.

Tuy mạng ngữ nghĩa là một kiểu biểu diễn trực quan đối với con người nhưng khi đưa vào máy tính, các đối tượng và mối liên hệ giữa chúng thường được biểu diễn dưới dạng những phát biểu động từ (như vị từ). Hơn nữa, các thao tác tìm kiếm trên mạng ngữ nghĩa thường khó khăn (đặc biệt đối với những mạng có kích thước lớn). Do đó, mô hình mạng ngữ nghĩa được dùng chủ yếu để phân tích vấn đề. Sau đó, nó sẽ được chuyển đổi sang dạng luật hoặc frame để thi hành hoặc mạng ngữ nghĩa sẽ được dùng kết hợp với một số phương pháp biểu diễn khác.

X.2. Ưu điểm và nhược điểm của mạng ngữ nghĩa

Ưu điểm

Mạng ngữ nghĩa rất linh động, ta có thể dễ dàng thêm vào mạng các đỉnh hoặc cung mới để bổ sung các tri thức cần thiết.

Mạng ngữ nghĩa có tính trực quan cao nên rất dễ hiểu.

Mạng ngữ nghĩa cho phép các đỉnh có thể thừa kế các tính chất từ các đỉnh khác thông qua các cung loại "là", từ đó, có thể tạo ra các liên kết "ngầm" giữa những đỉnh không có liên kết trực tiếp với nhau.

Mạng ngữ nghĩa hoạt động khá tự nhiên theo cách thức con người ghi nhận thông tin.

Nhược điểm

Cho đến nay, vẫn chưa có một chuẩn nào quy định các giới hạn cho các đỉnh và cung của mạng. Nghĩa là bạn có thể gán ghép bất kỳ khái niệm nào cho đỉnh hoặc cung!

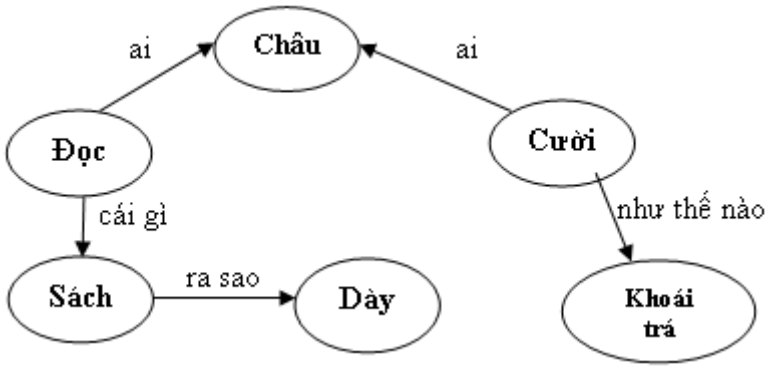
Tính thừa kế (vốn là một ưu điểm) trên mạng sẽ có thể dẫn đến nguy cơ mâu thuẫn trong tri thức. Chẳng hạn, nếu bổ sung thêm nút "Gà" vào mạng như hình sau thì ta có thể kết luận rằng "Gà" biết "bay"!. Sở dĩ có điều này là vì có sự không rõ ràng trong ngữ nghĩa gán cho một nút của mạng. Bạn đọc có thể phản đối quan điểm vì cho rằng, việc sinh ra mâu thuẫn là do ta thiết kế mạng dở chứ không phải do khuyết điểm của mạng!. Tuy nhiên, xin lưu ý rằng, tính thừa kế sinh ra *rất nhiều* mối liên "ngầm" nên khả năng nảy sinh ra một mối liên hệ không hợp lệ là rất lớn!

Hầu như không thể biển diễn các tri thức dạng thủ tục bằng mạng ngữ nghĩa vì các khái niệm về thời gian và trình tự không được thể hiện tường minh trên mạng ngữ nghĩa.

X.3. Một ví dụ tiêu biểu

Dù là một phương pháp tương đối cũ và có những yếu điểm nhưng mạng ngữ nghĩavẫn có những ứng dụng vô cùng độc đáo. Hai loại ứng dụng tiêu biểu của mạng ngữ nghĩa là ứng dụng xử lý ngôn ngữ tự nhiên và ứng dụng giải bài toán tự động.

Ví dụ 1 : Trong ứng dụng xử lý ngôn ngữ tự nhiên, mạng ngữ nghĩa có thể giúp máy tính phân tích được cấu trúc của câu để từ đó có thể phần nào "hiểu" được ý nghĩa của câu. Chẳng hạn, câu *"Châu đang đọc một cuốn sách dày và cười khoái trá"* có thể được biểu diễn bằng một mạng ngữ nghĩa như sau :



Ví dụ 2 : Giải bài toán tam giác tổng quát

Chúng ta sẽ không đi sâu vào ví dụ 1 vì đây là một vấn đề quá phức tạp để có thể trình bày trong cuốn sách này. Trong ví dụ này, chúng ta sẽ khảo sát một vấn đề đơn giản hơn nhưng cũng không kém phần độc đáo. Khi mới học lập trình, bạn thường được giáo viên cho những bài tập nhập môn đại loại như *"Cho 3 cạnh của tam giác, tính chiều dài các đường cao", "Cho góc a, b và cạnh AC. Tính chiều dài trung tuyến", ...* Với mỗi bài tập này, việc bạn cần làm là lấy giấy bút ra tìm cách tính, sau khi đã xác định các bước tính toán, bạn chuyển nó thành chương trình. Nếu có 10 bài, bạn

phải làm lại việc tính toán rồi lập trình 10 lần. Nếu có 100 bài, bạn phải làm 100 lần. Và tin buồn cho bạn là số lượng bài toán thuộc loại này là rất nhiều! Bởi vì một tam giác có tất cả **22** yếu tố khác nhau!. Không lẽ mỗi lần gặp một bài toán mới, bạn đều phải lập trình lại? Liệu có một chương trình tổng quát có thể tự động giải được *tất cả* (***vài ngàn!***) những bài toán tam giác thuộc loại này không? Câu trả lời là CÓ ! Và ngạc nhiên hơn nữa, chương trình này lại khá đơn giản. Bài toán này sẽ được giải bằng mạng ngữ nghĩa.

Có 22 yếu tố liên quan đến cạnh và góc của tam giác. Để xác định một tam giác hay để xây dựng một 1 tam giác ta cần có 3 yếu tố trong đó phải có yếu tố cạnh. Như vậy có khoảng C^3_{22}-1 (**khoảng vài ngàn**) cách để xây dựng hay xác định một tam giác. Theo thống kê, có khoảng 200 công thức liên quan đến cạnh và góc 1 tam giác.

Để giải bài toán này bằng công cụ mạng ngữ nghĩa, ta phải sử dụng khoảng 200 đỉnh để chứa công thức và khoảng 22 đỉnh để chứa các yếu tố của tam giác. Mạng ngữ nghĩa cho bài toán này có cấu trúc như sau :

Đỉnh của đồ thị bao gồm hai loại :

- Đỉnh chứa công thức (ký hiệu bằng hình chữ nhật)
- Đỉnh chứa yếu tố của tam giác (ký hiệu bằng hình tròn)

Cung : chỉ nối từ đỉnh hình tròn đến đỉnh hình chữ nhật cho biết yếu tố tam giác xuất hiện trong công thức nào *(không có trường hợp cung nối giữa hai đỉnh hình tròn hoặc cung nối giữa hai đỉnh hình chữ nhật).*

> * Lưu ý : trong một công thức liên hệ giữa n yếu tố của tam giác, ta giả định rằng nếu đã biết giá trị của n-1 yếu tố thì sẽ tính được giá trị của yếu tố còn lại. Chẳng hạn như trong công thức tổng 3 góc của tam giác bằng 180^0 thì khi biết được hai góc, ta sẽ tính được góc còn lại.

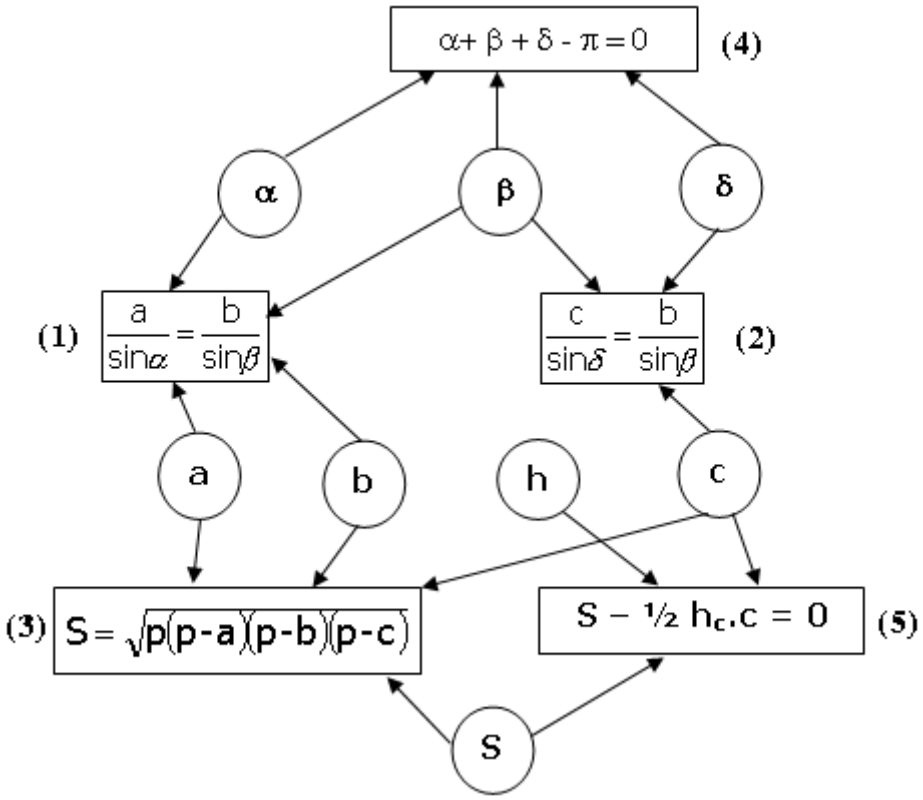
Cơ chế suy diễn thực hiện theo thuật toán "loang" đơn giản sau :

B1 : Kích hoạt những **đỉnh hình tròn** đã cho ban đầu (những yếu tố đã có giá trị)

B2 : Lặp lại bước sau cho đến khi kích hoạt được tất cả những đỉnh ứng với những yếu tố cần tính hoặc không thể kích hoạt được bất kỳ đỉnh nào nữa.

Nếu một đỉnh hình chữ nhật có cung nối với **n** đỉnh hình tròn mà **n-1** đỉnh hình tròn đã được kích hoạt **thì** kích hoạt đỉnh hình tròn còn lại (và tính giá trị đỉnh còn lại này thông qua công thức ở đỉnh hình chữ nhật).

Giả sử ta có mạng ngữ nghĩa để giải bài toán tam giác như hình sau



Ví dụ : *"Cho hai góc [redacted] và chiều dài cạnh a của tam giác. Tính chiều dài đường cao hC".* Với mạng ngữ nghĩa đã cho trong hình trên. Các bước thi hành của thuật toán như sau :

Bắt đầu : đỉnh [redacted] của đồ thị được kích hoạt.

Công thức (1) được kích hoạt (vì [redacted] được kích hoạt). Từ công thức (1) tính được cạnh b. Đỉnh b được kích hoạt.

Công thức (4) được kích hoạt (vì [redacted]. Từ công thức (4) tính được góc [redacted]

Công thức (2) được kích hoạt (vì 3 đỉnh [redacted] được kích hoạt). Từ công thức (2) tính được cạnh **c**. Đỉnh **c** được kích hoạt.

Công thức (3) được kích hoạt (vì 3 đỉnh a, b, c được kích hoạt) . Từ công thức (3) tính được diện tích **S.** Đỉnh S được kích hoạt.

Công thức (5) được kích hoạt (vì 2 đỉnh S, c được kích hoạt). Từ công thức (5) tính được hC. Đỉnh hC được kích hoạt.

Giá trị hC đã được tính. Thuật toán kết thúc.

Về mặt chương trình, ta có thể cài đặt mạng ngữ nghĩa giải bài toán tam giác bằng một mảng hai chiều A trong đó :

Cột : ứng với công thức. Mỗi cột ứng với một công thức tam giác khác nhau (đỉnh hình chữ nhật).

Dòng : ứng với yếu tố tam giác. Mỗi dòng ứng với một yếu tố tam giác khác nhau (đỉnh hình tròn).

Phần tử A[i, j] = **-1** nghĩa là trong công thức ứng với cột **j** có yếu tố tam giác ứng với cột **i**. Ngược lại A[i,j] = **0.**

Để thực hiện thao tác "kích hoạt" một đỉnh hình tròn, ta đặt giá trị của toàn dòng ứng với yếu tố tam giác bằng 1.

Để kiểm tra xem một công thức đã có đủ n-1 yếu tố hay chưa (nghĩa là kiểm tra điều kiện "*đỉnh hình chữ nhật có cung nối với n đỉnh hình tròn mà n-1 đỉnh hình tròn đã được kích hoạt*"), ta chỉ việc lấy **hiệu** giữa **tổng** số ô có giá trị bằng 1 và **tổng** số ô có giá trị **-1** trên cột ứng với công thức cần kiểm tra. Nếu kết quả bằng **n**, thì công thức đã có đủ n-1 yếu tố.

Trở lại mạng ngữ nghĩa đã cho. Quá trình thi hành kích hoạt được diễn ra như sau :

Mảng biểu diễn mạng ngữ nghĩa ban đầu

[redacted]	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**
[redacted]	**-1**	0	0	**-1**	0
[redacted]	**-1**	**-1**	0	**-1**	0
[redacted]	0	**-1**	0	**-1**	0
a	**-1**	0	**-1**	0	0
b	**-1**	**-1**	**-1**	0	0
c	0	**-1**	**-1**	0	**-1**
S	0	0	**-1**	0	**-1**
hC	0	0	0	0	**-1**

Khởi đầu : đỉnh [redacted] **a** của đồ thị được kích hoạt.

[redacted]	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**
[redacted]	**1**	0	0	**1**	0
[redacted]	**1**	**1**	0	**1**	0
[redacted]	0	**-1**	0	**-1**	0
a	**1**	**0**	**1**	**1**	0
b	**-1**	**-1**	**-1**	0	0
c	0	**-1**	**-1**	0	**-1**

S	0	0	**-1**	0	**-1**
hC	0	0	0	0	**-1**

Trên cột **(1)**, hiệu (1+1+1 – (-1)) = 4 nên dòng **b** sẽ được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
■	**1**	0	0	**1**	0
■	**1**	**1**	0	**1**	0
■	0	**-1**	0	**-1**	0
a	**1**	**0**	**1**	**1**	0
b	**1**	**1**	**1**	0	0
c	0	**-1**	**-1**	0	**-1**
S	0	0	**-1**	0	**-1**
hC	0	0	0	0	**-1**

Trên cột (4), hiệu (1+1+1 – (-1)) = 4 nên dòng ■ sẽ được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
■	**1**	0	0	**1**	0
■	**1**	**1**	0	**1**	0
■	0	**1**	0	**1**	0
a	**1**	**0**	**1**	**1**	0
b	**1**	**1**	**1**	0	0
c	0	**-1**	**-1**	0	**-1**
S	0	0	**-1**	0	**-1**
hC	0	0	0	0	**-1**

Trên cột (2), hiệu (1+1+1 – (1)) = 4 nên dòng **c** được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
■	**1**	0	0	**1**	0

[redacted]	**1**	**1**	0	**1**	0
[redacted]	0	**1**	0	**1**	0
A	**1**	**0**	**1**	**1**	0
B	**1**	**1**	**1**	0	0
C	0	**1**	**1**	0	**1**
S	0	0	**-1**	0	**-1**
hC	0	0	0	0	**-1**

Trên cột (3), hiệu (1+1+1 – (-1)) = 4 nên dòng **S** được kích hoạt.

	(1)	**(2)**	**(3)**	**(4)**	**(5)**
[redacted]	**1**	0	0	**1**	0
[redacted]	**1**	**1**	0	**1**	0
[redacted]	0	**1**	0	**1**	0
a	**1**	**0**	**1**	**1**	0
b	**1**	**1**	**1**	0	0
c	0	**1**	**1**	0	**1**
S	0	0	**1**	0	**1**
hC	0	0	0	0	**-1**

Trên cột (5), hiệu (1+1 – (1)) = 3 nên dòng hC được kích hoạt.

Khả năng của hệ thống này không chỉ dừng lại ở việc tính ra giá trị các yếu tố cần thiết, với một chút sửa đổi, chương trình này còn có thể đưa ra cách giải hình thức của bài toán và thậm chí còn có thể chọn được cách giải hình thức tối ưu (tối ưu hiểu theo nghĩa là cách giải sử dụng những công thức đơn giản nhất). Sở dĩ có thể nói như vậy vì cách suy luận của ta trong bài toán này là *tìm kiếm theo chiều rộng*. Do đó, khi đạt đến kết quả, ta có thể có rất nhiều cách khác nhau. Để có thể chọn được giải pháp tối ưu, bạn cần phải định nghĩa được độ "phức tạp" của một công thức. Một trong những tiêu chuẩn thường được dùng là số lượng phép nhân, chia, cộng, trừ, rút căn, tính sin, cos, ... được áp dụng trong công thức. Các phép tính sin, cos và rút căn có độ phức tạp cao nhất, kế đến là nhân chia và cuối cùng là cộng trừ. Cuối cùng bạn có thể cải tiến lại phương pháp suy luận bằng cách vận dụng thuật toán A[redacted]với ước lượng **h=0** để có thể chọn ra được "đường đi" tối ưu. Ta chọn ước lượng h=0 vì hai lý do sau (1) không gian bài toán nhỏ nên ta không cần phải giới hạn độ rộng tìm kiếm (2) xây dựng một ước lượng như vậy là tương đối khó khăn, đặc biệt là làm sao để hệ thống không đánh giá quá cao h.

XI. BIỂU DIỄN TRI THỨC BẰNG FRAME

XI.1. Khái niệm

Frame là một cấu trúc dữ liệu chứa đựng tất cả những tri thức liên quan đến một đối tượng cụ thể nào đó. Frames có liên hệ chặt chẽ đến khái niệm hướng đối tượng (thực ra frame là nguồn gốc của lập trình hướng đối tượng). Ngược lại với các phương pháp biểu diễn tri thức đã được đề cập đến, frame "đóng gói" toàn bộ một đối tượng, tình huống hoặc cả một vấn đề phức tạp thành một thực thể duy nhất có cấu trúc. Một frame bao hàm trong nó một khối lượng tương đối lớn tri thức về một đối tượng, sự kiện, vị trí, tình huống hoặc những yếu tố khác. Do đó, frame có thể giúp ta mô tả khá chi tiết một đối tượng.

> Dưới một khía cạnh nào đó, người ta có thể xem phương pháp biểu diễn tri thức bằng frame chính là nguồn gốc của ngôn ngữ lập trình hướng đối tượng. Ý tưởng của phương pháp này là "*thay vì bắt người dùng sử dụng các công cụ phụ như dao mở để đồ hộp, ngày nay các hãng sản xuất đồ hộp thường gắn kèm các nắp mở đồ hộp ngay bên trên vỏ lon. Như vậy, người dùng sẽ không bao giờ phải lo lắng đến việc tìm một thiết bị để mở đồ hộp nữa!*". Cũng vậy, ý tưởng chính của frame (hay của phương pháp lập trình hướng đối tượng) là khi biểu diễn một tri thức, ta sẽ "gắn kèm" những thao tác thường gặp trên tri thức này. Chẳng hạn như khi mô tả khái niệm về hình chữ nhật, ta sẽ gắn kèm *cách tính* chu vi, diện tích.

Frame thường được dùng để biểu diễn những tri thức "chuẩn" hoặc những tri thức được xây dựng dựa trên những kinh nghiệm hoặc các đặc điểm đã được hiểu biết cặn kẽ. Bộ não của con người chúng ta vẫn luôn "lưu trữ" rất nhiều các tri thức chung mà khi cần, chúng ta có thể "lấy ra" để vận dụng nó trong những vấn đề cần phải giải quyết. Frame là một công cụ thích hợp để biểu diễn những kiểu tri thức này.

XI.2. Cấu trúc của frame

Mỗi một frame mô tả một *đối tượng (object)*. Một frame bao gồm 2 thành phần cơ bản là **slot** và **facet.** Một **slot** là một thuộc tính đặc tả đối tượng được biểu diễn bởi frame. Ví dụ : trong frame mô tả xe hơi, có hai slot là *trọng lượng* và *loại máy*.

Mỗi slot có thể chứa một hoặc nhiều **facet.** Các facet (đôi lúc được gọi là slot "con") đặc tả một số thông tin hoặc thủ tục liên quan đến thuộc tính được mô tả bởi slot. Facet có nhiều loại khác nhau, sau đây là một số facet thường gặp.

Value *(giá trị)* : cho biết *giá trị* của thuộc tính đó (như xanh, đỏ, tím vàng nếu slot là màu xe).

Default *(giá trị mặc định)* : hệ thống sẽ tự động sử dụng giá trị trong facet này nếu slot là rỗng (nghĩa là chẳng có đặc tả nào!). Chẳng hạn trong frame về xe, xét slot về *số lượng bánh.* Slot này sẽ có giá trị 4. Nghĩa là, mặc định một chiếc xe hơi sẽ có 4 bánh!

Range *(miền giá trị)* : (tương tự như kiểu biến), cho biết giá trị slot có thể nhận những loại giá trị gì (như số nguyên, số thực, chữ cái, ...)

If added : mô tả một hành động sẽ được thi hành khi một giá trị trong slot được thêm vào (hoặc được hiệu chỉnh). Thủ tục thường được viết dưới dạng một script.

If needed **:** được sử dụng khi slot không có giá trị nào. Facet mô tả một hàm để tính ra giá trị của slot.

Frame : **XE HƠI**

Thuộc lớp : phương tiện vận chuyển.

Tên nhà sản xuất : Audi

Quốc gia của nhà sản xuất : Đức

Model *:* 5000 Turbo

Loại xe : Sedan

Trọng lượng : 3300lb

Số lượng cửa : 4 (default)

Hộp số : 3 số tự động

Số lượng bánh : 4 (default)

Máy (tham chiếu đến frame Máy)

Kiểu : In-line, overhead cam

Số xy-lanh : 5

Khả năng tăng tốc

0-60 : 10.4 giây

¼ dặm : 17.1 giây, 85 mph.

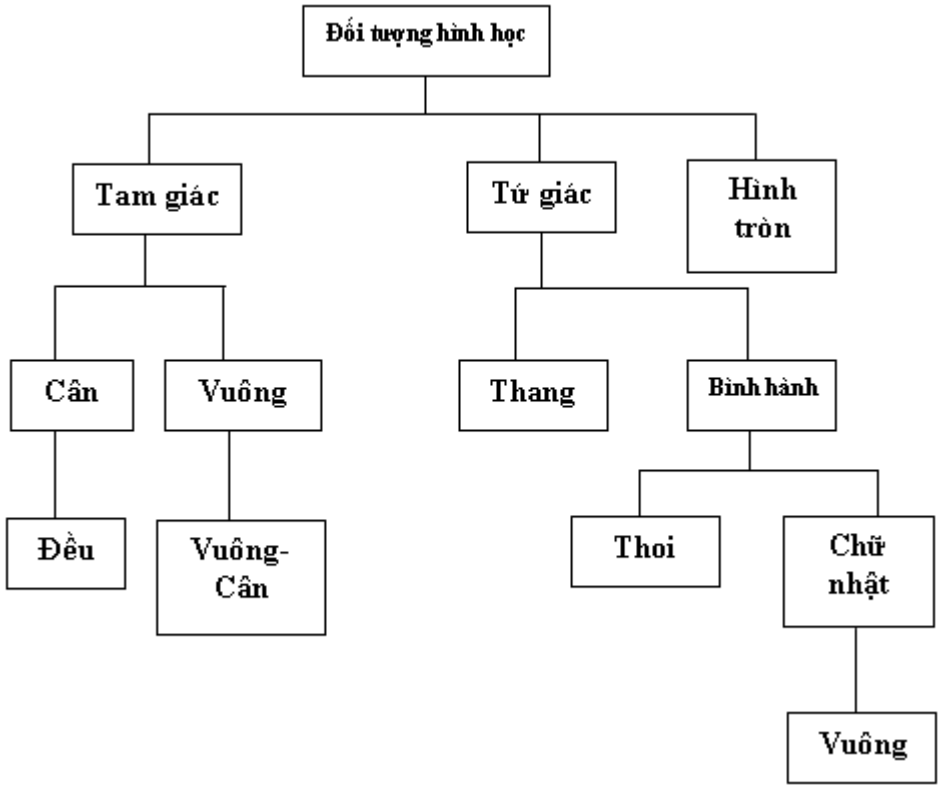
Frame **MÁY**
Xy-lanh : 3.19 inch
Tỷ lệ nén : 3.4 inche
Xăng : TurboCharger
Mã lực : 140 hp

XI.3. Tính kế thừa

Trong thực tế, một hệ thống trí tuệ nhân tạo thường sử dụng nhiều frame được liên kết với nhau theo một cách nào đó. Một trong những điểm thú vị của frame là tính phân cấp. Đặc tính này cho phép kế thừa các tính chất giữa các frame.

Hình sau đây cho thấy cấu trúc phân cấp của các loại hình hình học cơ bản. Gốc của cây ở trên cùng tương ứng với mức độ trừu tượng cao nhất. Các frame nằm ở dưới cùng (không có frame con nào) gọi là lá. Những frame nằm ở mức thấp hơn có thể thừa kế tất cả những tính chất của những frame cao hơn.

Các frame cha sẽ cung cấp những mô tả tổng quát về thực thể. Frame có cấp càng cao thì mức độ tổng quát càng cao. Thông thường, frame cha sẽ bao gồm các *định nghĩa* của các thuộc tính. Còn các frame con sẽ chứa đựng giá trị thực sự của các thuộc tính này.



Một ví dụ biểu diễn các đối tượng hình học bằng frame

Các kiểu dữ liệu cơ bản :

Area : numeric; // diện tích

Height : numeric; //chiều cao

Perimeter : numberic; //chu vi

Side : numeric; //cạnh

Diagonal : numeric; //đường chéo

Radius : numeric; //bán kính

Angle : numeric; //góc

Diameter : numeric; //đường kính

pi : (val:numeric = 3.14159)

Frame : **CIRCLE** (hình tròn)

r : radius;

s : area;

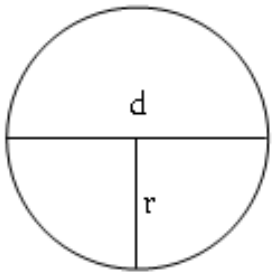
p : perimeter;

d : diameter;

d = 2 [redacted];

s = pi [redacted]^2;

p = 2 [redacted]pi [redacted];



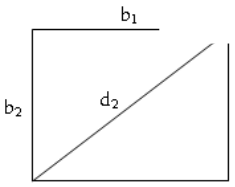
Frame **RECTANGLE** (hình chữ nhật)

b_1 : side;

b_2 : side;

s : area;

p : perimeter;



s = b_1 [redacted]b_2;

p = 2 [redacted]b_1+b_2);

$d^2 = b_1^2 + b_2^2$;

Frame **SQUARE** (hình vuông)

Là : **RECTANGLE**

$b_1 = b_2$;

Frame **RHOMBUS** (hình thoi)

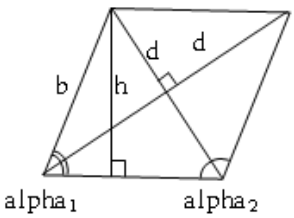
b : side;

d_1 : diagonal;

d_2 : diagonal;

s : area;

p : perimeter;



$alpha_1$: angle;

$alpha_2$: angle;

h : height;

cos ($alpha_2$/2) [redacted]$_1$ = h;

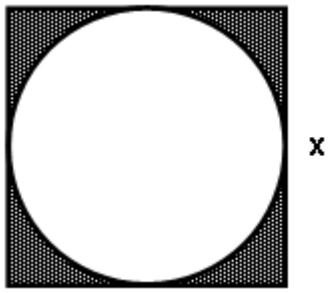
s = d_1 [redacted]$_2$ / 2;

p = 4 [redacted];

s = b [redacted];

cos (alpha$_2$/2)/(2[redacted]) = d_2;

Chúng ta có thể dễ dàng khai báo các đối tượng hình học khác theo cách này. Sau khi đã biểu diễn các tri thức về các hình hình học cơ bản xong, ta có thể vận dụng nó để giải các bài toán hình học, chẳng hạn bài toán tính diện tích. Ví dụ, cho hình vuông **k** và vòng tròn nội tiếp **c**, biết cạnh hình vuông có chiều dài là **x**, hãy viết chương trình để tính diện tích phần tô đen



Dễ thấy rằng, diện tích phần tô đen chính là hiệu giữa diện tích hình vuông và diện tích hình tròn nội tiếp. Dĩ nhiên là bạn cũng có thể viết một chương trình bình thường để tính toán, nhưng khi đã "tích hợp" các tri thức về tính diện tích bên trong biểu diễn, chương trình của chúng ta trở nên rất gọn nhẹ. Bạn hãy lưu ý 3 lệnh được in đậm trong ví dụ dưới. Lệnh đầu tiên sẽ "đặc tả" lại giả thiết *"hình vuông có cạnh với chiều dài x"*, lệnh kế tiếp đặc tả giả thiết *"hình tròn nội tiếp"*, còn lệnh thứ 3 mô tả việc tính diện tích bằng cách lấy diện tích hình vuông trừ cho diện tích hình tròn.

VAR x, s : numeric; k : square; c : circle;

BEGIN

<Nhập x>;

k.b_1 := x;

c.d := x;

s := k.s – c.s;

END.

Như vậy, chương trình máy tính của chúng ta đã hoạt động khá giống như việc "mô tả" các giải bài toán bằng ngôn ngữ tự nhiên. Hãy nghĩ xa hơn một tí. Các bài toán hình học thường được mô tả bằng các ngôn từ khá chính xác (chẳng hạn như : *cho* một *tam giác* với *chiều cao* xuất phát từ *đỉnh A* là 5, chiều dài cạnh *đáy* là 6, ...). Do đó, về mặt nguyên tác, chúng ta vẫn có thể xây dựng một chương trình để "hiểu" những đề bài này (theo như cách mà chúng ta vừa làm). Sau đó, người dùng có thể hoàn toàn nhờ máy tính giải giúp bài toán cho mình bằng cách *mô tả lời giải* cho máy tính (chứ không cần phải lập trình). Bạn có cảm giác điều này thật thú vị không? Đây chính là bước đi

đầu tiên trong việc tạo ra một chương trình *trợ giúp* cho việc giải các bài toán hình học trên máy tính với giao tiếp bằng ngôn ngữ tự nhiên!

Để tăng thêm sức mạnh cho hệ thống này, người ta thường cài đặt một mạng ngữ nghĩa ngay bên trong mỗi frame. Chẳng hạn, ta có thể có một frame **TRIANGLE**, trong đó cài đặt một mạng ngữ nghĩa (giống như ở ví dụ trong phần mạng ngữ nghĩa) để đặc tả mối liên hệ giữa các yếu tố tam giác (thay vì sử dụng các công thức liên hệ đơn giản như ví dụ trên).

XII. BIỂU DIỄN TRI THỨC BẰNG SCRIPT

Script là một cách biểu diễn tri thức tương tự như frame nhưng thay vì đặc tả một đối tượng, nó mô tả *một chuỗi các sự kiện*. Để mô tả chuỗi sự kiện, script sử dụng một dãy các **slot** chứa thông tin về các con người, đối tượng và hành động liên quan đến sự kiện đó.

Tuy cấu trúc của các script là rất khác nhau tùy theo bài toán, nhưng nhìn chung một script thường bao gồm các thành phần sau :

Điều kiện vào (entry condition): mô tả những tình huống hoặc điều kiện cần được thỏa mãn trước khi các sự kiện trong script có thể diễn ra.

Role (diễn viên): là những con người có liên quan trong script.

Prop (tác tố): là tất cả những đối tượng được sử dụng trong các chuỗi sự kiện sẽ diễn ra.

Scene(Tình huống) : là chuỗi sự kiện thực sự diễn ra.

Result (Kết quả) : trạng thái của các *Role* sau khi script đã thi hành xong.

Track (phiên bản) : mô tả một biến thể (hoặc trường hợp đặc biệt) có thể xảy ra trong đoạn script.

Sau đây là một ví dụ tiêu biểu cho script. Ví dụ này là một biến thể của ví dụ nổi tiếng về nhà hàng bán thức ăn nhanh (các nhà hàng bán gà rán mà ta thường gặp trong các siêu thị!) thường được sử dụng để minh họa cách biểu diễn tri thức bằng script trong cách sách nói về trí tuệ nhân tạo. Đi ăn trong một nhà hàng là một tình huống thường gặp trong cuộc sống với những *điều kiện vào, diễn viên, tác tố, hoàn cảnh, kết quả* khá "chuẩn". Và qua script ở ví dụ, bạn sẽ thấy phương pháp này có thể được dùng để mô tả chính xác những tình huống diễn ra hàng ngày của những nhà hàng bán thức ăn nhanh. Các *tình huống* là những đoạn script con trong đoạn script chính để mô tả những tình huống nhỏ trong toàn bộ quá trình. Lưu ý rằng trong đoạn script này có tình huống tùy chọn trong đó mô tả việc khách hàng mua thức ăn về thay vì vào nhà hàng ăn.

Script "nhà hàng"

Phiên bản : Nhà hàng bán thức ăn nhanh.

Diễn viên : Khách hàng

Người phục vụ.

Tác tố : Bàn phục vụ.

Chỗ ngồi.

Khay đựng thức ăn

Thức ăn

Tiền

Các loại gia vị như muối, tương, ớt, tiêu, ...

Điều kiện vào :

Khách hàng đói

Khách hàng có đủ tiền để trả.

Tình huống 1 : Vào nhà hàng

Khách hàng đậu xe vào bãi đậu xe.

Khách hàng bước vào nhà hàng.

Khách hàng xếp hàng trước bàn phục vụ.

Khách hàng đọc thực đơn trên tường và quyết định sẽ kêu món ăn gì.

Tình huống 2: Kêu món ăn.

Khách hàng kêu món ăn với người phục vụ (đang đứng ở quầy phục vụ)

Người phục vụ đặt thức ăn lên khay và đưa hóa đơn tính tiền cho khách.

Khách hàng trả tiền cho người phục vụ.

Tình huống 3: Khách hàng dùng món ăn

Khách hàng lấy thêm các gia vị

Khách hàng cầm khay đến một bàn còn trống.

Khách hàng ăn thức ăn.

Tình huống 3A (tùy chọn) : Khách hàng mua thức ăn đem về

Khách hàng mang thức ăn về nhà.

Tình huống 4 : Ra về

Khách hàng thu dọn bàn

Khách hàng bỏ rác (thức ăn thừa, xương, mảng vụn, ...) vào thùng rác.

Khách hàng ra khỏi nhà hàng.

Khách hàng lái xe đi.

Kết quả :

Khách hàng không còn đói.

Khách hàng còn ít tiền hơn ban đầu.

Khách hàng vui vẻ *

Khách hàng bực mình *

Khách hàng quá no.

* Tùy chọn.

Script rất hữu dụng trong việc dự đoán điều gì sẽ xảy đến trong những tình huống xác định. Thậm chí trong những tình huống chưa diễn ra, script còn cho phép máy tính *dự đoán* được việc gì sẽ xảy ra và xảy ra đối với ai và vào thời điểm nào. Nếu máy tính kích hoạt một script, người dùng có thể đặt câu hỏi và hệ thống có thể suy ra được những câu trả lời chính xác mà không cần người dùng cung cấp thêm nhiều thông tin (trong một số trường hợp có thể không cần thêm thông tin). Do đó, cũng giống như frame, script là một dạng biểu diễn tri thức tương đối hữu dụng vì nó cho phép ta mô tả chính xác những tình huống "chuẩn" mà con người vẫn thực hiện mỗi ngày hoặc đã nắm bắt chính xác.

Để cài đặt script trong máy tính, bạn phải tìm cách lưu trữ các tri thức dưới dạng hình thức. LISP là ngôn ngữ lập trình phù hợp nhất để làm điều này. Sau khi đã cài đặt xong script, bạn (người dùng) có thể đặt câu hỏi về những con người hoặc điều kiện có liên quan trong script. Hệ thống sau đó sẽ tiến hành thao tác tìm kiếm hoặc thao tác so mẫu để tìm câu trả lời. Chẳng hạn bạn có thể đặt câu hỏi "Khách hàng làm gì trước tiên?". Hệ thống sẽ tìm thấy câu trả lời trong scene 1 và đưa ra đáp án "Đậu xe và bước vào nhà hàng".

XIII. PHỐI HỢP NHIỀU CÁCH BIỂU DIỄN TRI THỨC

Mục tiêu chính biểu diễn tri thức trong máy tính là phục vụ cho việc thu nhận tri thức vào máy tính, truy xuất tri thức và thực hiện các phép suy luận dựa trên những tri thức đã lưu trữ. Do đó, để thỏa mãn được 3 mục tiêu trên, khi chọn phương pháp biểu diễn tri thức, chúng ta phải cân nhắc một số yếu tố cơ bản sau đây :

Tính tự nhiên, đồng bộ và dễ hiểu của biểu diễn tri thức.

Mức độ trừu tượng của tri thức : tri thức được khai báo cụ thể hay nhúng vào hệ thống dưới dạng các mã thủ tục?

Tính đơn thể và linh động của cơ sở tri thức (có cho phép dễ dàng bổ sung tri thức, mức độ phụ thuộc giữa các tri thức, ...)

Tính hiệu quả trong việc truy xuất tri thức và sức mạnh của các phép suy luận (theo kiểu heuristic) .

Bảng sau cho chúng ta một số ưu và khuyết điểm của các phương pháp biểu diễn tri thức đã được trình bày.

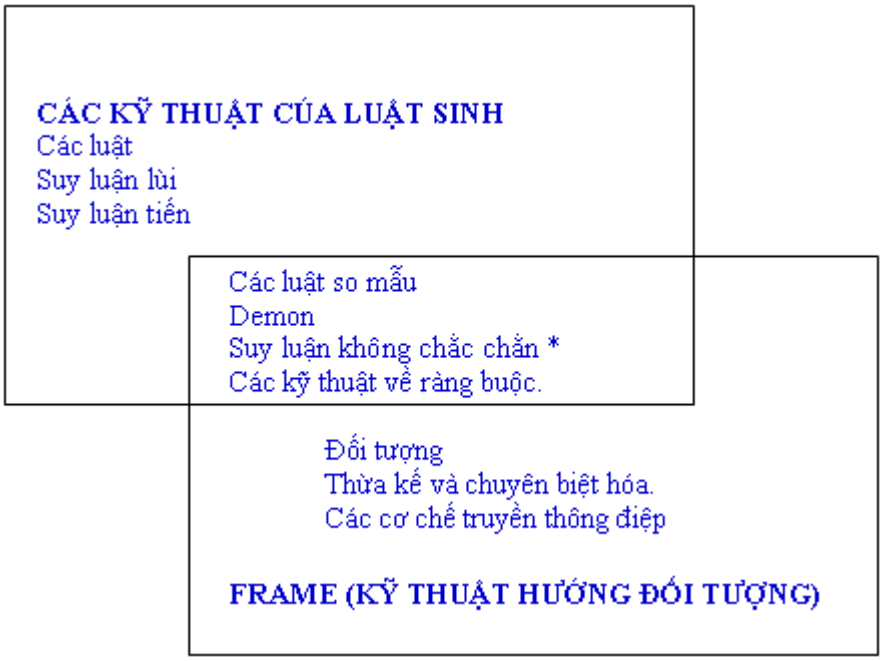
P.Pháp	Ưu điểm	Nhược điểm
Luật sinh	Cú pháp đơn giản, dễ hiểu, diễn dịch đơn giản, tính đơn thể cao, linh động (dễ điều chỉnh).	Rất khó theo dõi sự phân cấp, không hiệu quả trong những hệ thống lớn, không thể biểu diễn được mọi loại tri thức, rất yếu trong việc biểu diễn các tri thức dạng mô tả, có cấu trúc.
Mạng ngữ nghĩa	Dễ theo dõi sự phân cấp, sẽ dò theo các mối liên hệ, linh động	Ngữ nghĩa gắn liền với mỗi đỉnh có thể nhập nhằng, khó xử lý các ngoại lệ, khó lập trình.
Frame	Có sức mạnh diễn đạt tốt, dễ cài đặt các thuộc tính cho các slot cũng như các mối liên hệ, dễ dàng tạo ra các thủ tục chuyên biệt hóa, dễ đưa vào các thông tin mặc định và dễ thực hiện các thao tác phát hiện các giá trị bị thiếu sót.	Khó lập trình, khó suy diễn, thiếu phần mềm hỗ trợ.
Logic hình thức	Cơ chế suy luận chính xác (được chứng minh bởi toán học).	Tách rời việc biểu diễn và xử lý, không hiệu quả với lượng dữ liệu lớn, quá chậm khi cơ sở dữ liệu lớn.

Tuy vậy, như chúng ta đã biết, hiện nay vẫn chưa có một kiểu biểu diễn tri thức nào phù hợp với mọi tình huống. Do đó, khi phải làm việc với nhiều nguồn tri thức khác nhau (khác loại, khác tính chất), chúng ta nhiều lúc phải hy sinh tính đồng bộ bằng cách sử dụng cùng lúc nhiều kiểu biểu diễn tri thức, mỗi kiểu biểu diễn ứng với một nhiệm vụ con. Nhưng như vậy, chúng ta lại nảy sinh ra vấn đề "dịch" một tri thức từ kiểu biểu diễn này sang kiểu biểu diễn khác. Tuy thế nhưng một số hệ chương trình trí tuệ gần đây vẫn dùng cùng lúc nhiều kiểu biểu diễn dữ liệu khác nhau.

> Một trong những ví dụ kết hợp nhiều kiểu biểu diễn tri thức mà chúng ta đã từng làm quen là kiểu kết hợp giữa frame và mạng ngữ nghĩa trong việc trợ giúp giải bài toán hình học.

Một trong những sự phối hợp tương đối thành công là sự kết hợp giữa luật sinh và frame. Luật sinh không đủ hiệu quả trong nhiều ứng dụng, đặc biệt là trong các tác vụ định nghĩa, mô tả các đối tượng hoặc những mối liên kết tĩnh giữa các đối tượng. Nhưng những yếu điểm này lại chính là ưu điểm của frame. Ngày nay, đã có rất nhiều hệ thống đã tạo ra một kiểu biểu diễn lai giữa luật sinh và frame có được ưu điểm của hai cách biểu diễn. Sự thành công của các hệ thống nổi tiếng như KEE, Level5 Object và Nexpert Object đã minh chứng cho điều này. Frame cung cấp một ngôn ngữ cấu trúc hiệu quả để đặc tả những đối tượng xuất hiện trong các luật. Frame còn đóng vai trò như một lớp hỗ trợ cho thao tác suy diễn cơ bản trên những đối tượng không cần phải tương tác một cách tường minh trong các luật. Khả năng phân lớp của frame còn có thể được dùng để phân hoạch, tạo chỉ mục và sắp xếp các luật sinh trong hệ thống. Khả năng này rất thích hợp cho người dùng trong việc xây dựng và hiểu các luật, cũng như cũng có thể theo dõi được các luật được sử dụng khi nào và cho mục gì.

Hình sau cho thấy một kiểu kết hợp giữa luật sinh và frame. Sự kết hợp này đã cho phép tạo ra các luật so mẫu nhằm tăng tốc độ tìm kiếm của hệ thống. Kết quả của sự kết hợp này cho phép tạo ra các biểu diễn phức tạp hơn rất nhiều so với việc chỉ dùng frame, thậm chí phức tạp hơn cả việc lập trình trực tiếp bằng ngôn ngữ C++ !!.



** Suy luận không chắc chắn (Hypothetical reasoning) :* là kỹ thuật suy luận dựa trên các điều kiện có thể có mâu thuẫn hoặc không chắc chắn.

Ví dụ kết hợp biểu diễn tri thức bằng luật sinh và frame trong bài toán điều chế chất hóa học

Vấn đề : Cho trước một số chất hóa học. Hãy xây dựng chuỗi các phản ứng hóa học để điều chế một số chất hóa học khác.

Đầu tiên, đây là một ứng dụng hết sức tự nhiên của tri thức biểu diễn dưới dạng luật. Lý do là vì bản thân các phản ứng hóa học tiêu chuẩn đều được thể hiện dưới dạng luật. Chẳng hạn ta có các phương trình phản ứng sau :

$Na + Cl_2 \rightarrow NaCl$

$Fe + Cl_2 \rightarrow FeCl_2$

$Cu + Cl_2 \rightarrow CuCl_2$

$Cl_2 + H_2O \rightarrow HCl + HClO$

$MnO_2 + 4HCl \rightarrow MnCl_2 + Cl_2 + H_2O$

$HCl + KMnO_4 \rightarrow Cl + MnCl_2 + H_2O + Cl_2$

$NaCl + H_2O \rightarrow Cl_2 + H_2 + NaOH$

...

Như vậy, nếu xem một chất hóa học là một sự kiện và một phương trình phản ứng như là một luật dẫn thì bài toán điều chế chất hóa học, một cách rất tự nhiên, trở thành bài toán suy luận tiến trong cơ sở tri thức dạng luật dẫn.

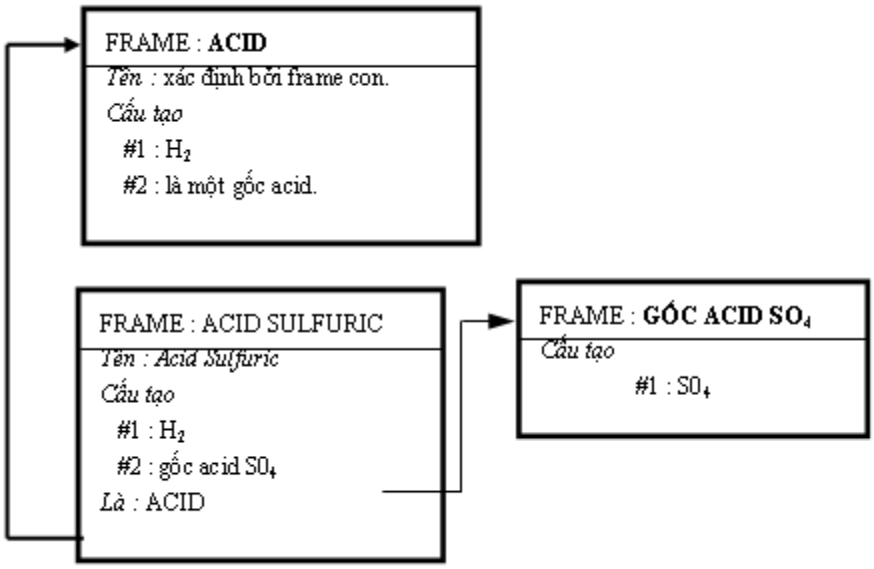
Tuy nhiên, số lượng các phản ứng là rất lớn, nên ta không thể sử dụng các luật dựa trên các phản ứng cụ thể như vậy mà phải sử dụng các phản ứng tổng quát hơn như :

Axit + Bazơ [redacted]uối + Nước

Kiềm + Nước [redacted]út + H_2

(trong hóa học cũng có nhiều phản ứng rất đặc biệt không thể tổng quát được, trong trường hợp này, ta sẽ xem phản ứng đó như là một luật riêng!).

Để mô tả được các phản ứng tổng quát như trên, ta sẽ sử dụng các frame. Chẳng hạn để đặc tả Acid Sulfuric H_2SO_4 ta sử dụng các frame tổng quát sau.



Dĩ nhiên là trong các frame ở trên còn rất nhiều thuộc tính hóa học khác. Ở đây chúng tôi chỉ trình bày sơ lược về mặt ý tưởng để bạn đọc có cơ sở bắt đầu. Ý tưởng này đã được một số sinh viên năm 4 của khoa Công Nghệ Thông Tin Đại Học Khoa Học Tự Nhiên TP. Hồ Chí Minh cài đặt thành công. Chương trình chạy tốt trong phạm vi các phản ứng trong sách giáo khoa lớp 10, 11 và 12.

Chương 3 MỞ ĐẦU VỀ QUAN MÁY HỌC

I. THẾ NÀO LÀ MÁY HỌC ?

II. HỌC BẰNG CÁCH XÂY DỰNG CÂY ĐỊNH DANH

II.1. Đâm chồi

II.2. Phương án chọn thuộc tính phân hoạch

II.2.1. Quinlan

II.2.2. Độ đo hỗn loạn

II.3. Phát sinh tập luật

II.4. Tối ưu tập luật

II.4.1. Loại bỏ mệnh đề thừa

II.4.2. Xây dựng mệnh đề mặc định

I. THẾ NÀO LÀ MÁY HỌC ?

Thuật ngữ "học" theo nghĩa thông thường là **tiếp thu tri thức** để biết cách vận dụng. Ở ngoài đời, quá trì học diễn ra dưới nhiều hình thức khác nhau như học thuộc lòng (học vẹt), học theo kinh nghiệm (học dựa theo trường hợp), học theo kiểu nghe nhìn,... Trên máy tính cũng có nhiều thuật toán học khác nhau. Tuy nhiên, trong phạm vi của giáo trình này, chúng ta chỉ khảo sát phương pháp học dựa theo trường hợp. Theo phương pháp này, hệ thống sẽ được cung cấp một số các trường hợp "mẫu", dựa trên tập mẫu này, hệ thống sẽ tiến hành phân tích và rút ra các quy luật (biểu diễn bằng luật sinh). Sau đó, hệ thống sẽ dựa trên các luật này để "đánh giá" các trường hợp khác (thường không giống như các trường hợp "mẫu"). Ngay cả chỉ với kiểu học này, chúng ta cũng đã có nhiều thuật toán học khác nhau. Một lần nữa, với mục đích giới thiệu, chúng ta chỉ khảo sát một trường hợp đơn giản.

Có thể khái quát quá trình *học theo trường hợp* dưới dạng hình thức như sau :

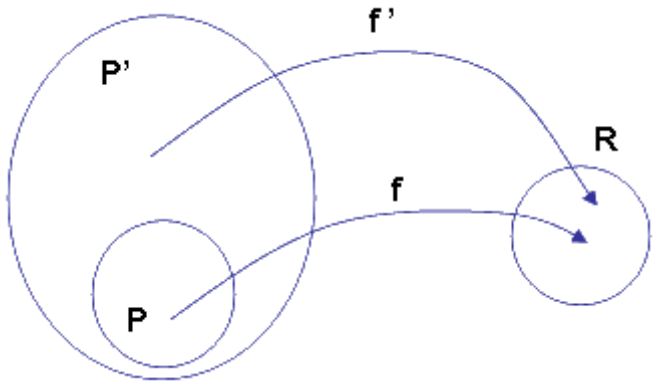
Dữ liệu cung cấp cho hệ thống là một ánh xạ f trong đó ứng một trường hợp p trong tập hợp P với một "lớp" r trong tập R.

f : P

p

Tuy nhiên, tập **P** thường nhỏ (và hữu hạn) so với tập tất cả các trường hợp cần quan tâm **P' (P P')**. Mục tiêu của chúng ta là xây dựng ánh xạ **f '** sao cho có thể ứng mọi trường hợp **p'** trong tập **P'** với một "lớp" **r** trong tập **R**. Hơn nữa, **f '** phải bảo toàn **f**, nghĩa là :

Với mọi **p thì f(p) '(p)**



Hình 3.1 : Học theo trường hợp là tìm cách xây dựng ánh xạ f' dựa theo ánh xạ f. f được gọi là **tập mẫu.**

Phương pháp học theo trường hợp là một phương pháp phổ biến trong cả nghiên cứu khoa học và mê tín dị đoan. Cả hai đều dựa trên các dữ liệu quan sát, thống kê để từ đó rút ra các quy luật. Tuy nhiên, khác với khoa học, mê tín dị đoan thường dựa trên tập mẫu không đặc trưng, cục bộ, thiếu cơ sở khoa học.

II. HỌC BẰNG CÁCH XÂY DỰNG CÂY ĐỊNH DANH

Phát biểu hình thức có thể khó hình dung. Để cụ thể hơn, ta hãy cùng nhau quan sát một ví dụ cụ. Nhiệm vụ của chúng ta trong ví dụ này là xây dựng các quy luật để có thể kết luận một người *như thế nào* khi đi tắm biển thì bị cháy nắng. Ta gọi tính chất cháy nắng hay không cháy nắng là thuộc tính quan tâm *(thuộc tính mục tiêu)*. Như vậy, trong trường hợp này, tập **R** của chúng ta chỉ gồm có hai phần tử **{"cháy nắng", "bình thường"}**. Còn tập **P** là tất cả những người được liệt kê trong bảng dưới (8 người) Chúng ta quan sát hiện tượng cháy nắng dựa trên 4 thuộc tính sau : *chiều cao (cao, trung bình, thấp), màu tóc (vàng, nâu, đỏ) cân nặng (nhẹ, TB, nặng),* dùng kem (*có, không*),. Ta gọi các thuộc tính này gọi là *thuộc tính dẫn xuất*.

Dĩ nhiên là trong thực tế để có thể đưa ra được một kết luận như vậy, chúng ta cần nhiều dữ liệu hơn và đồng thời cũng cần nhiều thuộc tính dẫn xuất trên. Ví dụ đơn giản này chỉ nhằm để minh họa ý tưởng của thuật toán máy học mà chúng ta sắp trình bày.

Tên	Tóc	Ch.Cao	Cân Nặng	Dùng kem?	Kết quả
Sarah	Vàng	T.Bình	Nhẹ	Không	Cháy
Dana	Vàng	Cao	T.Bình	Có	Không
Alex	Nâu	Thấp	T.Bình	Có	Không
Annie	Vàng	Thấp	T.Bình	Không	Cháy
Emilie	Đỏ	T.Bình	Nặng	Không	Cháy
Peter	Nâu	Cao	Nặng	Không	Không
John	Nâu	T.Bình	Nặng	Không	Không
Kartie	Vàng	Thấp	Nhẹ	Có	Không

Ý tưởng đầu tiên của phương pháp này là tìm cách *phân hoạch* tập P ban đầu thành các tập Pi sao cho tất cả các phần tử trong tất cả các tập Pi đều có chung thuộc tính mục tiêu.

P = P_1 [redacted]$_2$ [redacted]. [redacted]n và **[redacted]i,j) i[redacted] :** thì **(Pi [redacted]j =** [redacted] và

[redacted] [redacted]i,l : pk [redacted]i và **pl [redacted]j** thì **f(pk) = f(pl)**

Sau khi đã phân hoạch xong tập P thành tập các phân hoạch **Pi** được đặc trưng bởi thuộc tính đích **ri (ri [redacted]R)**, bước tiếp theo là ứng với *mỗi* phân hoạch **Pi** ta xây dựng luật **Li : GTi** [redacted] trong đó các GT_i là mệnh đề được hình thành bằng cách kết hợp các thuộc tính dẫn xuất.

Một lần nữa, vấn đề hình thức có thể làm bạn cảm thấy khó khăn. Chúng ta hãy thử ý tưởng trên với bảng số liệu mà ta đã có.

Có hai cách phân hoạch hiển nhiên nhất mà ai cũng có thể nghĩ ra. Cách đầu tiên là cho *mỗi người* vào một phân hoạch riêng (P_1 = {Sarah}, P_2 = {Dana}, … tổng cộng sẽ có 8 phân hoạch cho 8 người). Cách thứ hai là phân hoạch thành hai tập, một tập gồm tất cả những người *cháy nắng* và tập còn lại bao gồm tất cả những người *không cháy nắng*. Tuy đơn giản nhưng phân hoạch theo kiểu này thì chúng ta chẳng giải quyết được gì !!

II.1. Đâm chồi

Chúng ta hãy thử một phương pháp khác. Bây giờ bạn hãy quan sát thuộc tính đầu tiên – màu tóc. Nếu dựa theo màu tóc để phân chia ta sẽ có được 3 phân hoạch khác nhau ứng với mỗi giá trị của thuộc tính màu tóc. Cụ thể là :

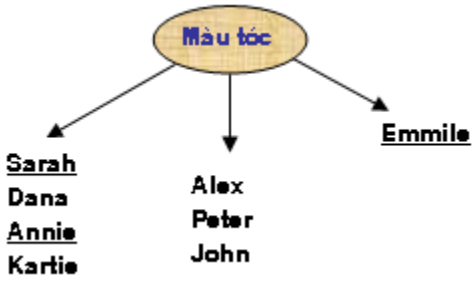
Pvàng = { **Sarah**, Dana, **Annie**, Kartie }

Pnâu = { Alex, Peter, John }

Pđỏ = { **Emmile** }

* Các người bị cháy nắng được gạch dưới và in đậm.

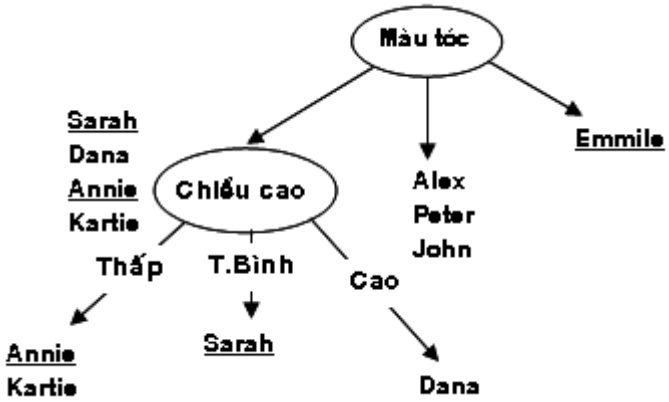
Thay vì liệt kê ra như trên, ta dùng sơ đồ cây để tiện mô tả cho các bước phân hoạch sau :



Quan sát hình trên ta thấy rằng phân hoạch Pnâu và Pđỏ thỏa mãn được điều kiện *"có chung thuộc tính mục tiêu"* (Pnâu chứa toàn người không cháy nắng, Pđỏ chứa toàn người cháy nắng).

Còn lại tập Pvàng là còn lẫn lộn người cháy năng và không cháy nắng. Ta sẽ tiếp tục phân hoạch tập này thành các tập con. Bây giờ ta hãy quan sát thuộc tính chiều cao. Thuộc tính này giúp phân hoạch tập Pvàng thành 3 tập con : $PVàng_{Thấp}$ = {**Annie**, Kartie}, $PVàng_{T.Bình}$= {**Sarah**} và $PVàng_{Cao}$= { Dana }

Nếu nối tiếp vào cây ở hình trước ta sẽ có hình ảnh cây phân hoạch như sau :



Quá trình này cứ thế tiếp tục cho đến khi tất cả các nút lá của cây không còn lẫn lộn giữa cháy nắng và không cháy nắng nữa. Bạn cũng thấy rằng, qua mỗi bước phân hoạch cây phân hoạch ngày càng "phình" ra. Chính vì vậy mà quá trình này được gọi là quá trình "đâm chồi". Cây mà chúng ta đang xây dựng được gọi là cây định danh.

Đến đây, chúng ta lại gặp một vấn đề mới. Nếu như ban đầu ta không chọn thuộc tính màu tóc để phân hoạch mà chọn thuộc tính khác như chiều cao chẳng hạn để phân hoạch thì sao? Cuối cùng thì cách phân hoạch nào sẽ tốt hơn?

II.2. Phương án chọn thuộc tính phân hoạch

Vấn đề mà chúng ta gặp phải cũng tương tự như bài toán tìm kiếm : "Đứng trước một ngã rẽ, ta cần phải đi vào hướng nào?". Hai phương pháp đánh giá dưới đây sẽ giúp ta chọn được thuộc tính phân hoạch tại mỗi bước xây dựng cây định danh.

II.2.1. Quinlan

Quinlan quyết định thuộc tính phân hoạch bằng cách xây dựng các *vector đặc trưng* cho mỗi giá trị của từng thuộc tính dẫn xuất và thuộc tính mục tiêu. Cách tính cụ thể như sau :

Với mỗi thuộc tính dẫn xuất **A** *còn có thể sử dụng* để phân hoạch, tính :

$$VA(j) = (T(j , r_1), T(j , r_2) , \ldots, T(j , rn))$$

T(j, ri) = (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j **và** có giá trị thuộc tính mục tiêu là ri) **/** (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j)

** trong đó $r_1, r_2, \ldots , rn$ là các giá trị của thuộc tính mục tiêu*

$$* \sum_i T(j, r_i) = 1$$

Như vậy nếu một thuộc tính A có thể nhận một trong 5 giá trị khác nhau thì nó sẽ có 5 vector đặc trưng.

Một vector V(Aj) được gọi là vector đơn vị nếu nó chỉ có duy nhất một thành phần có giá trị 1 và những thành phần khác có giá trị 0.

Thuộc tính được chọn để phân hoạch là thuộc tính có nhiều vector đơn vị nhất.

Trở lại ví dụ của chúng ta, ở trạng thái ban đầu (chưa phân hoạch) chúng ta sẽ tính vector đặc trưng cho từng thuộc tính dẫn xuất để tìm ra thuộc tính dùng để phân hoạch. Đầu tiên là thuộc tính màu tóc. Thuộc tính màu tóc có 3 giá trị khác nhau *(vàng, đỏ, nâu)* nên sẽ có 3 vector đặc trưng tương ứng là :

VTóc (vàng) = (T(vàng, cháy nắng), T(vàng, không cháy nắng))

Số người tóc vàng là : **4**

Số người tóc vàng và cháy nắng là : **2**

Số người tóc vàng và không cháy nắng là : **2**

Do đó

VTóc(vàng) = (2/4 , 2/4) = (0.5, 0.5)

Tương tự

VTóc(nâu) = (0/3, 3/3) = (0,1) (vector đơn vị)

Số người tóc nâu là : **3**

Số người tóc nâu và cháy nắng là : **0**

Số người tóc nâu và không cháy nắng là : **3**

VTóc(đỏ) = (1/1, 0/1) = (1,0) (vector đơn vị)

Tổng số vector đơn vị của thuộc tính tóc vàng là **2**

Các thuộc tính khác được tính tương tự, kết quả như sau :

$VC_{.Cao}$(Cao) = (0/2,2/2) = **(0,1)**

$VC_{.Cao}$(T.B) = (2/3,1/3)

$VC_{.Cao}$(Thấp) = (1/3,2/3)

$VC_{.Nặng}$ (Nhẹ) = (1/2,1/2)

$VC_{.Nặng}$ (T.B) = (1/3,2/3)

$VC_{.Nặng}$ (Nặng) = (1/3,2/3)

VKem (Có) = (3/3,0/3) = **(1,0)**

VKem (Không) = (3/5,2/5)

Như vậy thuộc tính màu tóc có số vector đơn vị nhiều nhất nên sẽ được chọn để phân hoạch.

Sau khi phân hoạch theo màu tóc xong, chỉ có phân hoạch theo tóc vàng (Pvàng) là còn chứa những người cháy nắng và không cháy nắng nên ta sẽ tiếp tục phân hoạch tập này. Ta sẽ thực hiện thao tác tính vector đặc trưng tương tự đối với các thuộc tính còn lại *(chiều cao, cân nặng, dùng kem)*.
Trong phân hoạch Pvàng, tập dữ liệu của chúng ta còn lại là :

Tên	Ch.Cao	Cân Nặng	Dùng kem?	Kết quả
Sarah	T.Bình	Nhẹ	Không	Cháy
Dana	Cao	T.Bình	Có	Không
Annie	Thấp	T.Bình	Không	Cháy

Kartie	Thấp	Nhẹ	Có	Không

$VC_{.Cao}$(Cao) = (0/1,1/1) = **(0,1)**

$VC_{.Cao}$(T.B) = (1/1,0/1) = **(1,0)**

$VC_{.Cao}$(Thấp) = (1/2,1/2)

$VC_{.Nặng}$ (Nhẹ) = (1/2,1/2)

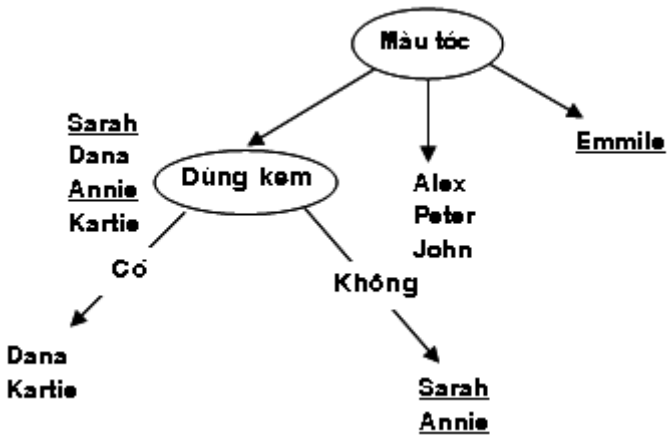
$VC_{.Nặng}$ (T.B) = (1/2,1/2)

$VC_{.Nặng}$ (Nặng) = (0,0)

VKem (Có) = (0/2,2/2) = **(0,1)**

VKem (Không) = (2/2,0/2) = **(1,0)**

2 thuộc tính dùmg kem và chiều cao đều có 2 vector đơn vị. Tuy nhiên, số phân hoạch của thuộc tính dùng kem là ít hơn nên ta chọn phân hoạch theo thuộc tính dùng kem. Cây định danh cuối cùng của chúng ta sẽ như sau :



II.2.2. Độ đo hỗn loạn

Thay vì phải xây dựng các vector đặc trưng như phương pháp của Quinlan, ứng với mỗi thuộc tính dẫn xuất ta chỉ cần tính ra độ đo hỗn loạn và lựa chọn thuộc tính nào có độ đo hỗn loại là thấp nhất. Công thức tính như sau :

$$TA = \sum_j \left(\frac{b_j}{b_t} \times \sum_i \left(-\frac{b_{ri}}{b_j} \times \log_2 \left(-\frac{b_{ri}}{b_j} \right) \right) \right)$$

trong đó :

b_t là tổng số phần tử có trong phân hoạch

b_j là tổng số phần tử có thuộc tính dẫn xuất A có giá trị j.

b_{ri} : tổng số phần tử có thuộc tính dẫn xuất A có giá trị j và thuộc tính mục tiêu có giá trị i.

II.3. Phát sinh tập luật

Nguyên tắc phát sinh tập luật từ cây định danh khá đơn giản. Ứng với mỗi nút lá, ta chỉ việc đi từ đỉnh cho đến nút lá đó và phát sinh ra luật tương ứng. Cụ thể là từ cây định danh kết quả ở cuối phần II.2 ta có các luật sau (xét các nút lá từ trái sang phải)

(Màu tóc *vàng*) và (*có* dùng kem) [redacted]ông cháy nắng

(Màu tóc *vàng*) và (*không* dùng kem) [redacted]áy nắng

(Màu tóc *nâu*) [redacted]ông cháy nắng

(Màu tóc *đỏ*) [redacted]áy nắng

Khá đơn giản phải không? Có lẽ không có gì phải nói gì thêm. Chúng ta hãy thực hiện bước cuối cùng là tối ưu tập luật.

II.4. Tối ưu tập luật

II.4.1. Loại bỏ mệnh đề thừa

Khác so với các phương pháp loại bỏ mệnh đề thừa đã được trình bày trong phần biểu diễn tri thức (chỉ quan tâm đến logic hình thức), phương pháp loại bỏ mệnh đề thừa ở đây dựa vào dữ liệu. Với ví dụ và tập luật đã có ở phần trước, bạn hãy quan sát luật sau :

(Màu tóc *vàng*) và (*có* dùng kem) [redacted]ông cháy nắng

Bây giờ ta hãy lập một bảng (gọi là bảng Contigency), bảng thống kê những người *có dùng kem* tương ứng với tóc màu vàng và bị cháy nắng hay không. Trong dữ liệu đã cho, có 3 người không dùng kem.

	Không cháy nắng	**Cháy nắng**
Màu *vàng*	2	0
Màu khác	1	0

Theo bảng thống kê này thì rõ ràng là thuộc tính tóc vàng (trong luật trên) không đóng góp gì trong việc đưa ra kết luận cháy nắng hay không (cả 3 người dùng kem đều không cháy nắng) nên ta có thể loại bỏ thuộc tính tóc vàng ra khỏi tập luật.

Sau khi loại bỏ mệnh đề thừa, tập mệnh đề của chúng ta trong ví dụ trên sẽ còn :

(*có* dùng kem) [redacted]hông cháy nắng

(Màu tóc *vàng*) và (*không* dùng kem) [redacted]áy nắng

(Màu tóc *nâu*) [redacted]hông cháy nắng

(Màu tóc *đỏ*) [redacted]áy nắng

Như vậy quy tắc chung để có thể loại bỏ một mệnh đề là như thế nào? Rất đơn giản, giả sử luật của chúng ta có n mệnh đề :

A_1 và A_2 và … và An [redacted]

Để kiểm tra xem có thể loại bỏ mệnh đề Ai hay không, bạn hãy lập ra một tập hợp P bao gồm các phần tử thỏa tất cả mệnh đề **A_1 , A_2 , … Ai., Ai_{+1}, …, An** *(lưu ý : không cần xét là có thỏa Ai hay không, chỉ cần thỏa các mệnh đề còn lại là được)*

Sau đó, bạn hãy lập bảng Contigency như sau :

	R	[redacted]
Ai	E	F
[redacted] Ai	G	H

Trong đó

E là số phần tử trong P thỏa cả Ai và R.

F là số phần tử trong P thỏa Ai và không thỏa R

G là số phần tử trong P không thỏa Ai và thỏa R

H là số phần tử trong P không thỏa Ai và không thỏa R

Nếu tổng F+H = 0 thì có thể loại bỏ mệnh đề Ai ra khỏi luật.

II.4.2. Xây dựng mệnh đề mặc định

Có một vấn đề đặt ra là khi gặp phải một trường hợp mà tất cả các luật đều không thỏa thì phải làm như thế nào? Một cách hành động là đặt ra một luật mặc định đại loại như :

Nếu không có luật nào thỏa [redacted]áy nắng (1)

Hoặc

Nếu không có luật nào thỏa [redacted]hông cháy nắng. (2)

(chỉ có hai luật vì thuộc tính mục tiêu chỉ có thể nhận một trong hai giá trị là cháy nắng hay không cháy nắng)

Giả sử ta đã chọn luật mặc định là (2) thì tập luật của chúng ta sẽ trở thành :

(Màu tóc *vàng*) và (*không* dùng kem) [redacted]áy nắng

(Màu tóc *đỏ*) [redacted]áy nắng

Nếu không có luật nào thỏa [redacted]ông cháy nắng. (2)

Lưu ý rằng là chúng ta đã loại bỏ đi tất cả các luật dẫn đến kết luận không cháy nắng và thay nó bằng luật mặc định. Tại sao vậy? Bởi vì các luật này *có cùng kết luận* với luật mặc định. Rõ ràng là chỉ có thể có một trong hai khả năng là cháy nắng hay không.

Vấn đề là chọn luật nào? Sau đây là một số quy tắc.

1) Chọn luật mặc định sao cho nó có thể thay thế cho nhiều luật nhất. (trong ví dụ của ta thì nguyên tắc này không áp dụng được vì có 2 luật dẫn đến cháy nắng và 2 luật dẫn đến không cháy nắng)

2) Chọn luật mặc định có kết luận phổ biến nhất. Trong ví dụ của chúng ta thì nên chọn luật (2) vì số trường hợp không cháy nắng là 5 còn không cháy nắng là 3.

3) Chọn luật mặc định sao cho tổng số mệnh đề của các luật mà nó thay thế là nhiều nhất. Trong ví dụ của chúng ta thì luật được chọn sẽ là luật (1) vì tổng số mệnh đề của luật dẫn đến cháy nắng là 3 trong khi tổng số mệnh đề của luật dẫn đến không cháy nắng chỉ là 2.

BÀI TẬP

BÀI TẬP

CHƯƠNG 1

1) Viết chương trình giải bài toán hành trình người bán hàng rong bằng hai thuật giải GTS_1 và GTS_2 trong trường hợp có n địa điểm khác nhau.

2) Viết chương trình giải bài toán phân công công việc bằng cách ứng dụng nguyên lý thứ tự.

3) Ứng dụng nguyên lý thứ tự, hãy giải bài toán chia đồ vật sau. Có n vật với khối lượng lần lượt là M_1, M_2, … Mn. Hãy tìm cách chia n vật này thành hai nhóm sao cho chênh lệch khối lượng giữa hai nhóm này là nhỏ nhất.

4) Viết chương trình giải bài toán mã đi tuần.

5) Viết chương trình giải bài toán 8 hậu.

6) Viết chương trình giải bài toán Ta-canh bằng thuật giải A^*.

7) Viết chương trình giải bài toán tháp Hà Nội bằng thuật giải A^*.

8)* Viết chương trình tìm kiếm đường đi ngắn nhất trong một bản đồ tổng quát. Bản đồ được biểu diễn bằng một mảng hai chiều A, trong đó A[x,y]=0 là có thể đi được và A[x,y]= 1 là

vật cản. Cho phép người dùng click chuột trên màn hình để tạo bản đồ và xác định điểm xuất phát và kết thúc. Chi phí để đi từ một ô bất kỳ sang ô kế cận nó là 1.

Mở rộng bài toán trong trường hợp chi phí để di chuyển từ ô (x,y) sang một bất kỳ kế (x,y) là A[x,y].

CHƯƠNG 2

1. Viết chương trình minh họa các bước giải bài toán đong nước (sử dụng đồ họa càng tốt).
2. Viết chương trình cài đặt hai thuật toán Vương Hạo và Robinson trong đó liệt kê các bước chứng minh một biểu thức logic.
3. Viết chương trình giải bài toán tam giác tổng quát bằng mạng ngữ nghĩa (lưu ý sử dụng thuật toán ký pháp nghịch đảo Ba Lan)
4. Hãy thử xây dựng một bộ luật phức tạp hơn trong ví dụ đã được trình bày dùng để chuẩn đoán hỏng hóc của máy tính. Viết chương trình ứng dụng bộ luật này trong việc chuẩn đoán hỏng hóc của máy tính (sử dùng thuật toán suy diễn lùi).
5. Hãy cài đặt các frame đặc tả các đối tượng hình học bằng kỹ thuật hướng đối tượng trong ngôn ngữ lập trình mà bạn quen dùng. Hãy xây dựng một ngôn ngữ script đơn giản cho phép người dùng có thể sử dụng các frame này trong việc giải một số bài toán hình học đơn giản.

CHƯƠNG 3

1) Cho bảng số liệu sau

Hãy xây dựng cây định danh và tìm luật để xác định một người là Châu Âu hay Châu Á bằng hai phương pháp vector đặc trưng của Quinlan và độ đo hỗn loạn.

STT	Dáng	Cao	Giới	Châu
1	To	TB	Nam	Á
2	Nhỏ	Cao	Nam	Á
3	Nhỏ	TB	Nam	Âu
4	To	Cao	Nam	Âu
5	Nhỏ	TB	Nữ	Âu
6	Nhỏ	Cao	Nam	Âu
7	Nhỏ	Cao	Nữ	Âu
8	To	TB	Nữ	Âu

2)* Viết chương trình cài đặt tổng quát thuật toán học dựa trên việc xây dựng cây định danh. Chương trình yêu cầu người dùng đưa vào danh sách các thuộc tính dẫn xuất, thuộc tính mục tiêu cùng với tất cả các giá trị của mỗi thuộc tính; yêu cầu người dùng cung cấp bảng số liệu quan sát. Chương trình sẽ liệt kê lên màn hình các luật mà nó tìm được từ bảng số liệu. Sau đó, yêu cầu người dùng nhập vào các trường hợp cần xác định, hệ thống sẽ đưa ra kết luận của trường hợp này.

Lưu ý : Nên sử dụng một hệ quản trị CSDL để cài đặt chương trình này.

TÀI LIỆU THAM KHẢO

1. Bạch Hưng Khang, Hoàng Kiếm, Trí tuệ nhân tạo: Các phương pháp và ứng dụng. Nhà xuất bản Khoa học và Kỹ thuật, 1989

2. John Durkin, Expert System . Prentice Hall, 1994

3. N. Nilson, Artificial Intelligence. McGrawhill, 1971

4. Patrick Henry Winston, Artificial Intelligence. Addison Wesley, 1992

5. Rich Elaine, Artificial Intelligence. Addison Wesley, 1983