MỞ ĐẦU

Trí tuệ nhân tạo là nỗ lực tìm hiểu những vấn đề trí tuệ, nghiên cứu vấn đề này cũng là cách để con người tự tìm hiểu về mình. Không giống như những ngành khoa học khác, mục tiêu ngành Trí tuệ nhân tạo là tạo ra các thực thể thông minh giúp ích cho con người. Thực tế chỉ ra rằng máy tính với độ thông minh nhất định đã có ảnh hưởng lớn đến cuộc sống ngày nay và tương lai phát triển của nền văn minh nhân loại.

Trong các trường đại học, cao đẳng, Trí tuệ nhân tạo đã trở thành môn học chuyên nhành của sinh viên ngành công nghệ thông tin. Để đáp ứng nhu cầu đó, tập thể giáo viên Khoa Công nghệ thông tin – Trường Đại học Sư phạm Kỹ thuật Vĩnh Long đã biên soạn giáo trình này.

Giáo trình môn Trí tuệ nhân tạo này được biên soạn dựa trên kinh nghiệm giảng dạy môn Trí tuệ nhân tạo nhiều năm của giáo viên trong khoa chúng tôi. Ngoài ra, chúng tôi cũng đã tham khảo một số tài liệu của các trường đại học, cao đẳng trong và ngoài nước.

Cấu trúc giáo trình này được biên soạn theo đề cương chi tiết môn học Trí tuệ nhân tạo và Hệ chuyên gia đã được Trường Đại học Sư phạm Kỹ thuật Vĩnh Long ban hành áp dụng cho sinh viên trường. Nội dung bao gồm 5 chương như sau:

- Chương 1: Tổng quan về trí tuệ nhân tạo
- Chương 2: Các phương pháp giải quyết vấn đề
- Chương 3: Các phương pháp biểu diễn tri thức
- Chương 4: Hệ chuyên gia dựa trên luật
- Chương 5: Máy học

Mục tiêu của giáo trình này là làm tài liệu học tập cho sinh viên chuyên ngành Công nghệ thông tin – Trường Đại học Sư phạm Kỹ thuật Vĩnh Long nhưng cũng không loại trừ sự tham khảo của các đối tượng khác.

Chúng tôi đã cố gắng cụ thể hóa các nội dung trong giáo trình với thời lượng mà môn học cho phép. Dù vậy, có lẽ giáo trình vẫn còn nhiều thiếu sót và hạn chế. Chúng tôi rất mong nhận được ý kiến đóng góp của các bạn đồng nghiệp, các sinh viên để giáo trình được hoàn chỉnh hơn theo thời gian.

Chân thành cảm ơn!

Vĩnh Long, tháng 10 năm 2015

MỤC LỤC

Chương 1 TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

1.1 LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN CỦA NGÀNH TRÍ TUỆ NHÂN TẠO

Những năm gần đây, khá nhiều sách, báo, công trình nghiên cứu khoa học đề cập đến các kỹ thuật tính toán, người ta hay nhắc đến nhiều thuật ngữ như: máy tính thông minh, máy tính thế hệ V, hệ chuyên gia, mạng ngữ nghĩa, ... Các ngôn ngữ lập trình như LISP, PROLOG mở đường cho việc áp dụng hàng loạt các hệ thống chương trình có khả năng "thông minh".

Trước đây, mỗi khi nói đến Trí tuệ nhân tạo (TTNT) người ta thường quan tâm đến việc tạo lập các máy tính có khả năng "suy nghĩ", thậm chí trong một số phạm vi hẹp nào đó, có thể cạnh tranh hoặc vượt quá khả năng của bộ não con người. Những hy vọng này trong một thời gian dài đã ảnh hưởng rất nhiều đến các nghiên cứu trong phòng thí nghiệm. Mặc dù những mô hình tương tự các máy tính thông minh đã được đưa ra từ nhiều năm trước, nhưng chỉ từ khi Alan Turing công bố những kết quả nghiên cứu quan trọng đầu tiên, người ta mới bắt đầu thực sự nghiên cứu đến các vấn đề TTNT một cách nghiêm túc. Phát hiện của Turing cho rằng chương trình có thể được lưu trữ trong bộ nhớ để sau đó được thực hiện trên cơ sở các phép toán cơ bản thao tác với các bit 0, 1. Điều này đã tạo nên nền tảng của những máy tính hiện đại. Việc lưu trữ chương trình trong máy cho phép thay đổi chức năng của nó một cách nhanh chóng và dễ dàng thông qua việc nạp một chương trình mới vào bộ nhớ. Theo một nghĩa nào đó, khả năng này làm cho máy tính có khả năng học và suy nghĩ. Đó cũng chính là một trong những biểu hiện quan trọng đầu tiên của những máy tính được trang bị TTNT.

Năm 1956, chương trình dẫn xuất kết luận trong hệ hình thức đã được công bố. Tiếp theo đó, năm 1959 chương trình chứng minh các định lý hình học phẳng và chương trình giải quyết bài toán vạn năng (GPS - General Problem Solving) đã được đưa ra. Tuy vậy chỉ cho đến khoảng năm 1960 khi McCathy ở MIT (Massachussets Institute of Technology) đưa ra ngôn ngữ lập trình đầu tiên dùng cho trí tuệ nhân tạo LISP (List Processing), các nghiên cứu về TTNT mới bắt đầu phát triển mạnh mẽ. Thuật ngữ TTNT do Marvin Minsky một chuyên gia nổi tiếng cũng ở MIT đưa ra năm 1961 trong bài báo " Steps Forwards To Artificial Intelligence". Những năm 60 có thể xem là một mốc quan trọng trong quá trình xây dựng các máy có khả năng suy nghĩ. Các chương trình chơi cờ và các chương trình chứng minh định lý toán học đầu tiên cũng được công bố trong khoảng thời gian này.

Những bế tắc, hạn chế sự thành công của các công trình nghiên cứu TTNT trong những năm 60 chính là do giới hạn khả năng của các thiết bị, bộ nhớ và đặc biệt là yếu tố thời gian thực hiện. Chính những yếu tố này không cho phép tổng quát hóa những

thành công bước đầu đạt được trong các hệ chương trình TTNT đã xây dựng. Tuy rằng vào giữa những năm 70, bộ nhớ máy tính và thời gian tính toán đã được nâng cao đáng kể về chất, song những cách tiếp cận khác nhau đến TTNT vẫn chưa đem tới những thành công thật sự do sự bùng nổ tổ hợp trong quá trình tìm kiếm lời giải cho các bài toán đặt ra.

Cuối những năm 70, một số nghiên cứu cơ bản trong các lĩnh vực như xử lý ngôn ngữ tự nhiên, biểu diễn tri thức, lý thuyết giải quyết vấn đề đã đem lại diện mạo mới cho TTNT. Thị trường tin học đã bắt đầu đón nhận những sản phẩm TTNT ứng dụng đầu tiên mang tính thương mại. Đó là các hệ chuyên gia được áp dụng trong các lĩnh vực khác nhau. Hệ chuyên gia là các phần mềm máy tính, chứa các thông tin và tri thức về một lĩnh vực cụ thể nào đó, có khả năng giải quyết những yêu cầu của người dùng ở một mức độ nào đó với trình độ như một chuyên gia có kinh nghiệm lâu năm. Một trong những hệ chuyên gia đầu tiên được sử dụng thành công trong thực tế là hệ MYCIN, được thiết kế và cài đặt tại trường Đại học Tổng Hợp Stanford.

Một sự kiện quan trọng trong sự phát triển của khoa học TTNT là sự ra đời của ngôn ngữ PROLOG, do Alain Calmerauer đưa ra năm 1972. Năm 1981, dự án của Nhật Bản xây dựng các máy tính thế hệ thứ V lấy ngôn ngữ PROLOG như là ngôn ngữ cơ sở đã làm thay đổi khá nhiều tình hình phát triển TTNT ở Mỹ cũng như châu Âu.

Giai đoạn 1981 trở đi người ta cảm nhận khá rõ nét rằng các chuyên gia về TTNT đang dần chuyển các kết quả nghiên cứu từ phòng thí nghiệm sang cài đặt các ứng dụng cụ thể. Có thể nói đây cũng là giai đoạn cạnh tranh ráo riết của các công ty, các viện nghiên cứu hàng đầu nhằm đưa ra thị trường các sản phẩm phần mềm ứng dụng kỹ thuật TTNT.

Cuối những năm 80, đầu những năm 90 thị trường các sản phẩm dân dụng đã có khá nhiều sản phẩm ở trình độ cao như máy giặt, máy ảnh, . . . sử dụng TTNT. Các hệ thống nhận dạng và xử lý hình ảnh, tiếng nói đang ngày càng thúc đẩy sự phát triển kỹ thuật mạng Neuron. Sự xích lại của hai cách tiếp cận: Tiếp cận mờ trong lập luận xấp xỉ và kỹ thuật mạng Neuron đã và đang gây được sự quan tâm đặc biệt của các chuyên gia tin học. Bên cạnh sự xuất hiện của các hệ chuyên gia, các ứng dụng công nghiệp và quản lý xã hội, quản lý kinh tế cũng đòi hỏi sự ra đời của các hệ thống xử lý tri thức – dữ liệu tích hợp.

Thế giới đang chuyển mình trong những nghiên cứu về TTNT. Tuy vậy, câu hỏi liệu kỹ thuật TTNT có tạo nên những bước nhảy vọt trong công nghệ tin học, đặc biệt là trong công nghệ máy tính như người ta đã mong đợi hay không vẫn chưa có lời giải đáp thỏa đáng.

1.2 ĐỊNH NGHĨA TRÍ TUỆ NHÂN TẠO

Trí tuệ nhân tạo (Artificial Intelligence - AI) có thể được định nghĩa như một ngành của khoa học máy tính liên quan đến việc tự động hóa các hành vi thông minh.

TTNT là một bộ phận của khoa học máy tính và do đó nó phải được đặt trên những nguyên lý lý thuyết vững chắc, có khả năng ứng dụng được của lĩnh vực này. Những nguyên lý này bao gồm các cấu trúc dữ liệu dùng cho biểu diễn tri thức, các thuật toán cần thiết để áp dụng những tri thức đó, cùng các ngôn ngữ và kỹ thuật lập trình dùng cho việc cài đặt chúng.

Tuy nhiên định nghĩa trên phải chấp nhận một thực tế - trí tuệ tự nó là một khái niệm không được định nghĩa một cách rõ ràng. Mặc dù hầu hết chúng ta đều có thể nhận ra các hành vi thông minh khi nhìn thấy chúng nhưng rất khó có thể đưa ra một định nghĩa về trí tuệ.

Vì thế mà vấn đề định nghĩa TTNT tự nó trở thành một định nghĩa trí tuệ: đó có phải là một năng lực duy nhất hay chỉ là tên dùng gọi một tập hợp những khả năng khác nhau và không liên quan gì đến nhau? Thế nào là khả năng sáng tạo? Thế nào là trực giác? Có thể kết luận ngay về tính trí tuệ từ việc quan sát một hành vi được không hay cần phải có biểu hiện của một cơ chế nào đó nằm bên trong? Tất cả những câu hỏi này vẫn chưa được trả lời và chúng đã đặt ra những vấn đề cần có phương pháp luận để giải quyết.

Cho đến nay, TTNT vẫn còn là một ngành khoa học trẻ, những mối quan tâm và những phương pháp giải quyết của nó chưa được rõ ràng so với tất cả các ngành khoa học đã trưởng thành trước đó. Song, một trong những mục tiêu trọng tâm của nó là mở rộng khả năng của khoa học máy tính chứ không là việc tìm cách định nghĩa những giới hạn của nó.

1.3 CÁC LĨNH VỰC ỨNG DỤNG CỦA NGÀNH TRÍ TUỆ NHÂN TẠO

1.3.1 Trò chơi

Ngay từ thời kỳ đầu của việc nghiên cứu vấn đề tìm kiếm trong không gian trạng thái, người ta đã tiến hành nhiều thử nghiệm bằng cách sử dụng các trò chơi cờ thông dụng như *cờ đam* (Checkers), *cờ vua* và *trò đố 15 ô* (15 - puzzle). Ngoài sức quyến rũ do tính chất trí óc vốn có trong các trò chơi cờ, có nhiều tính chất nhất định làm cho chúng trở thành một đối tượng lý tưởng của thời kỳ này. Hầu hết các trò chơi đều sử dụng một tập hợp các luật chơi được xác định rõ ràng. Điều này làm cho việc phát sinh không gian tìm kiếm trở nên dễ dàng và giải phóng nhà nghiên cứu khỏi những sự mơ hồ và phức tạp vốn có trong các bài toán ít cấu trúc hơn. Hình dạng của những bàn cờ sử dụng trong các trò chơi này dễ dàng được biểu diễn vào máy tính. Do đó, việc thử

nghiệm một chương trình chơi trò chơi không phải trả một gánh nặng nào về tài chính hay đạo đức.

Các trò chơi có thể phát sinh ra một số lượng không gian tìm kiếm cực kỳ lớn. Những không gian này đủ lớn và phức tạp để đòi hỏi những kỹ thuật mạnh nhằm quyết định xem những chọn lựa nào cần được xem xét trong không gian bài toán. Những kỹ thuật này được gọi là các heuristic và chúng tạo thành một lĩnh vực lớn trong TTNT. Một heuristic là một chiến lược giải quyết vấn đề tốt nhưng tiềm ẩn khả năng thất bại, chẳng hạn như việc cắm một thiết bị không nhạy vào để kiểm tra nó đã hỏng chưa hay việc cố gắng bảo vệ quân hậu khỏi bị bắt trong trò chơi cờ vua.

Nhiều thứ mà chúng ta gọi là thông minh thuộc về các Heuristic được người ta sử dụng để giải quyết các vấn đề.

Hầu hết chúng ta đều có một số kinh nghiệm với những trò chơi đơn giản này nên chúng ta cũng có khả năng nghĩ ra và kiểm nghiệm tính hiệu quả của những heuristic này. Chúng ta không cần đi tìm và hỏi ý kiến chuyên gia như trong một số lĩnh vực chuyên môn sâu chẳng hạn y học hay toán học (cờ vua là một ngoại lệ dễ thấy đối với quy tắc này). Vì những lý do đó, các trò chơi cung cấp một không gian mênh mông cho việc nghiên cứu các tìm kiếm Heuristic. Các chương trình chơi trò chơi, trái ngược với tính đơn giản của chúng, đưa ra những thử thách riêng của chúng, bao gồm các đấu thủ mà các nước đi của anh ta có thể không dự đoán trước được một cách chắc chắn. Sự có mặt này của đấu thủ càng làm phức tạp hơn mô hình chương trình do sự thêm vào một yếu tố không dự đoán trước được và sự cần thiết phải tính đến những yếu tố tâm lý cũng như là chiến thuật trong chiến lược của trò chơi.

1.3.2 Suy luận và chứng minh định lý tự động

Chúng ta có thể cho rằng chứng minh định lý tự động là một nhánh nghiên cứu có từ lâu đời nhất của Trí tuệ nhân tạo khi tìm lại nguồn gốc của nó qua các tác phẩm "*Nhà lý luận logic* (Logic theorist)" (Newell và Simon 1963a) và "*Công cụ giải quyết vấn đề tổng quát* (General problem solver)" (Newell và Simon 1965b) của Newell và Simon, cho đến trong những nổ lực của Rusell và Whitehead xem toàn bộ toán học như là sự dẫn xuất hình thức thuần tuý của các định lý từ các tiên đề cơ sở. Trong bất cứ trường hợp nào, nó chắc chắn vẫn là một trong những ngành phong phú nhất của lĩnh vực này. Nghiên cứu chứng minh định lý đã đạt được nhiều thành tích trong thời kỳ đầu của việc hình thức hoá các giải thuật tìm kiếm và phát triển các ngôn ngữ biểu diễn hình thức như phép tính vị từ.

Hầu hết sự quyến rũ của chứng minh định lý tự động đều là không đáng tin cậy và không đúng với nguyên tắc chung của logic. Vì là một hệ hình thức, logic tự bổ sung cho mình sự tự động hoá. Người ta có thể khảo sát một số lượng lớn những bài toán

khác nhau, bằng cách biểu diễn mô tả của bài toán và những thông tin cơ sở liên quan như là tiên đề logic và xem những trường hợp bài toán là những định lý cần phải chứng minh. Sự hiểu biết thấu đáo này là cơ sở cho việc nghiên cứu chứng minh định lý tự động và các hệ suy luận toán học.

Một lý do khác cho việc tiếp tục quan tâm đến các máy chứng minh định lý tự động là sự nhận thức rằng một hệ thống kiểu như vậy không nhất thiết phải có khả năng giải quyết những bài toán cực kỳ phức tạp một cách độc lập mà không có sự trợ giúp nào của con người. Nhiều máy chứng minh định lý hiện đại hoạt động như những trợ lý viên thông minh khi chúng cho phép con người thực hiện những công tác đòi hỏi trình độ cao hơn là phân tích một bài toán lớn thành nhiều bài toán con và đặt ra những heuristic để tìm kiếm trong không gian những chứng minh có thể chọn. Máy chứng minh định lý sau đó thực hiện công việc đơn giản hơn nhưng cũng quan trọng chẳng hạn chứng minh các bổ đề, kiểm chứng những giải quyết nhỏ hơn và hoàn thành những khía cạnh hình thức của một chứng minh đã được phác thảo bởi sự hợp tác của nó với con người (Boyer và More 1979).

1.3.3 Các hệ chuyên gia

Kể từ lúc khoa học giải quyết vấn đề được nghiên cứu, người ta đã sớm ý thức một cách sâu sắc và cơ bản về tầm quan trọng của tri thức chuyên ngành. Lấy ví dụ một bác sĩ chẳng hạn, cô ta không thể chẩn đoán bệnh tốt chỉ nhờ vào một số kỹ năng giải quyết vấn đề tổng quát bẩm sinh; mà cô ta đã chẩn đoán tốt là vì cô ta có nhiều kiến thức y học. Tương tự như thế, một nhà địa chất giỏi phát hiện các mỏ khoáng vì anh ta biết áp dụng một cách hiệu quả nhiều tri thức lý thuyết và thực nghiệm về địa lý vào bài toán đang nằm trong tay anh ta. Tri thức chuyên gia về lĩnh vực là sự kết hợp giữa kiến thức lý thuyết về vấn đề đó và một tập hợp các quy tắc giải quyết vấn đề theo kiểu heuristic mà kinh nghiệm khi sử dụng những quy tắc này đã tỏ ra hiệu quả trong lĩnh vực đó. Các hệ chuyên gia được người ta xây dựng bằng cách thu thập các kiến thức từ chuyên gia người và mã hoá nó thành dạng thức mà máy tính có thể áp dụng cho những bài toán tương tự.

Sự tin cậy vào tri thức của chuyên gia chuyên ngành trong các chiến lược giải quyết vấn đề của hệ là một đặc trưng chính của các hệ chuyên gia. Người ta đã viết ra một số chương trình mà ở đó người thiết kế cũng là nguồn tri thức chuyên ngành, nhưng sẽ điển hình hơn nhiều nếu chúng ta xem xét những chương trình được phát sinh từ sự cộng tác giữa một chuyên gia chuyên ngành chẳng hạn như một bác sĩ, một nhà hoá học, một nhà địa chất học hay một kỹ sư, với một chuyên gia riêng về trí tuệ nhân tạo. Chuyên gia chuyên ngành cung cấp kiến thức cần thiết về chuyên ngành thông qua những cuộc thảo luận tổng quát về các phương pháp giải quyết vấn đề của bằng cách biểu diễn những kỹ năng đó trên một tập hợp các bài toán mẫu được chọn lựa cẩn thận. Chuyên gia TTNT hay còn gọi là *kỹ sư tri thức* (knowledge engineer), như người ta

vẫn thường gọi là các nhà thiết kế hệ chuyên gia, có trách nhiệm thể hiện tri thức này vào một chương trình mà chương trình đó phải vừa hiệu quả vừa có vẻ thông minh trong các hành vi của nó. Một chương trình như thế vừa hoàn thành xong, cần phải tinh chế kiến thức chuyên môn của nó thông qua một quá trình cung cấp cho nó những bài toán mẫu để giải, để cho chuyên gia chuyên ngành phê bình hành vi của nó và thực hiện bất cứ thay đổi hay cải biến nào cần thiết đối với tri thức của chương trình. Quá trình này lặp đi lặp lại cho đến khi chương trình đạt được mức độ hoàn thiện mong muốn.

Một trong các hệ chuyên gia sớm nhất khai thác tri thức chuyên ngành để giải quyết vấn đề là DENDRAL được phát triển tại Stanford vào cuối những năm 1960 (Lindsay et al.1980). DENDRAL được thiết kế để phỏng đoán cấu trúc của các phân tử hữu cơ từ công thức hoá học của chúng và các thông tin về khối quang phổ có liên quan đến các liên kết hoá học có mặt trong các phân tử. Vì các phân tử hữu cơ thường rất lớn, nên số lượng cấu trúc có khả năng tồn tại đối với những phân tử này thường là khổng lồ. DENDRAL chú ý vào bài toán của không gian tìm kiếm rộng lớn này bằng cách áp dụng tri thức heuristic của các chuyên gia hoá học vào bài toán làm sáng tỏ cấu trúc. Các phương pháp của DENDRAL đã tỏ ra có một sức mạnh đáng kể khi thường xuyên tìm thấy cấu trúc đúng trong hàng triệu khả năng khác nhau chỉ sau có vài phép thử. Phương pháp này tỏ ra thành công đến mức người ta đã sử dụng những phiên bản của hệ chuyên gia nói trên trong các phòng thí nghiệm hoá học khắp nơi trên thế giới.

Trong khi DENDRAL là một trong số những chương trình đầu tiên sử dụng tri thức chuyên ngành một cách hiệu quả để đạt được khả năng giải quyết vấn đề cấp chuyên gia, thì MYCIN là hệ chuyên gia đã thiết lập nên phương pháp luận cho các *hệ chuyên gia hiện đại* (Contemporary expert systems) (Buchanan and Shortliff 1984). MYCIN sử dụng tri thức y khoa chuyên gia để chẩn đoán và kê đơn điều trị cho bệnh viêm màng não tuỷ sống và những trường hợp nhiễm trùng vi khuẩn trong máu.

MYCIN, được các nhà nghiên cứu phát triển ở Stanford vào giữa những năm 1970, là một trong những chương trình đầu tiên chú ý đến những bài toán suy luận bằng thông tin không chắc chắn hoặc không đầy đủ. MYCIN cung cấp những giải quyết rõ ràng và logic về quá trình suy luận của nó, sử dụng một cấu trúc kiểm tra thích hợp với lĩnh vực chuyên môn của vấn đề và nhận biết đặc tính để đánh giá một cách tin cậy hoạt động của nó. Nhiều kỹ thuật xây dựng hệ chuyên gia đang dùng hiện nay đã được người ta phát triển lần đầu trong dự án MYCIN.

Những hệ chuyên gia cổ điển khác bao gồm chương trình PROSPECTOR dùng để tìm ra những nơi có chứa quặng mỏ và xác định loại quặng mỏ, dựa trên thông tin địa lý về một địa điểm nào đó (duda et al. 1979a, 1979b), chương trình INTERNIST dùng để chẩn đoán trong lĩnh vực nội khoa, DIPMETER ADVISOR dùng để phiên dịch các kết quả của các máy khoan giếng dầu (Smith and Baker 1983) và XCON dùng để định

hình các máy tính hệ VAX. XCON được sử dụng từ năm 1981, tất cả các máy VAX và Digital Equipment Corporation bán thời bấy giờ đều được định hình bằng XCON. Vô số những hệ chuyên gia khác ngày nay đang giải quyết những bài toán trong nhiều lĩnh vực khác nhau như y học, giáo dục, kinh doanh, thiết kế và khoa học (Waterman 1986). Một điều thú vị mà chúng ta có thể nhận thấy là hầu hết các hệ chuyên gia được viết cho những lĩnh vực khá chuyên biệt và ở cấp độ chuyên gia. Nói chung những lĩnh vực này đều được nghiên cứu kỹ và chúng có những chiến lược giải quyết vấn đề đã xác định một cách rõ ràng.

Mặc dù còn tồn tại những hạn chế này các hệ chuyên gia vẫn đang chứng minh giá trị của chúng trong nhiều ứng dụng quan trọng.

1.3.4 Mô hình hoá ngữ nghĩa ngôn ngữ tự nhiên

Một trong những mục tiêu có từ lâu đời của Trí tuệ nhân tạo là tạo ra các chương trình có khả năng hiểu ngôn ngữ của con người. Khả năng hiểu ngôn ngữ tự nhiên không chỉ là một trong những biểu hiện căn bản nhất của trí thông minh con người mà sự tự động hoá nó một cách thành công sẽ gây ra một tác động ngoài sức tưởng tượng đối với năng lực và hiệu quả chính của những chiếc máy tính. Người ta đã bỏ ra nhiều công sức để viết các chương trình có khả năng hiểu ngôn ngữ tự nhiên. Tuy những chương trình này đã có được một số thành công trong những ngữ cảnh hạn chế, nhưng các hệ thống có khả năng sử dụng ngôn ngữ tự nhiên một cách linh hoạt và tổng quát theo cách như con người vẫn còn ở ngoài tầm tay những phương pháp luận hiện nay.

Hiểu ngôn ngữ tự nhiên liên quan đến nhiều thứ hơn nhiều so với chỉ phân tích các câu thành các phần riêng rẽ những nhóm câu của chúng và tìm những từ đó trong từ điển. Khả năng hiểu thực sự tuỳ thuộc vào kiến thức nền tảng rộng lớn về lĩnh vực của bài văn và những thành ngữ dùng trong lĩnh vực đó, cũng như là khả năng ứng dụng những kiến thức tổng quát tuỳ thuộc theo ngữ cảnh để giải quyết những trường hợp bỏ sót hay tối nghĩa, là một đặc điểm bình thường trong lối nói con người.

Ví dụ như chúng ta thử xem xét những khó khăn khi tiến hành một cuộc hội thoại về bóng chày với một người biết tiếng Anh nhưng không biết gì về luật chơi, các đấu thủ hoặc lịch sử của môn bóng chày. Khi đó, liệu người này có thể hiểu được hay không nghĩa của câu: “With none down in the top of the nith anh the go-ahead run at second, the managercalled this relief from the bull pen?” Tuy từng từ riêng lẻ trong câu này là có thể hiểu được, nhưng câu này vẫn được coi là sai ngữ pháp ngay cả đối với người thông minh nhất trong số những người không am hiểu bóng chày.

Công việc tập hợp và tổ chức kiến thức nền tảng này được tiến hành theo cách mà sao cho cách ấy có thể áp dụng được cho sự lĩnh hội ngôn ngữ, đã hình thành nên vấn đề chủ yếu của việc tự động hoá quá trình hiểu ngôn ngữ tự nhiên. Để đáp ứng yêu cầu

này, các nhà nghiên cứu đã phát triển nhiều kỹ thuật dùng để cấu trúc hoá ý nghĩa ngữ nghĩa, các kỹ thuật này được dùng xuyên suốt khoa học TTNT.

Do việc hiểu ngôn ngữ tự nhiên đòi hỏi những khối lượng kiến thức khổng lồ, hầu hết các công trình được người ta thực hiện trong những lĩnh vực vấn đề đã được hiểu rõ và chuyên môn hoá. Một trong những chương trình khai thác sớm nhất phương pháp luận "thế giới qui mô" này là SHRDLU của Winograd, một hệ ngôn ngữ tự nhiên có khả năng "trò chuyện" về hình dáng đơn giản của các khối có nhiều hình dạng và màu sắc khác nhau (Winograd 1973). SHRDLU có thể trả lời được những câu hỏi kiểu như "khối màu gì đang nằm trên hình lập phương màu xanh da trời?" và dự kiến những hành động kiểu như "di chuyển hình chóp màu đỏ lên viên gạch màu xanh lá cây". Những bài toán loại này, liên quan đến việc mô tả và thao tác những sắp xếp đơn giản của các khối đã xuất hiện và thường xuyên gây ngạc nhiên trong giới nghiên cứu TTNT và được người ta biết đến dưới cái tên là những bài toán "thế giới của khối".

Mặc cho SHRDLU thành công với việc trò chuyện về sự sắp xếp của các khối, nhưng phương pháp của nó đã không đủ khái quát được để vượt ra khỏi thế giới các khối. Những kỹ thuật biểu diễn được sử dụng trong chương trình này quá đơn giản nên không đủ để tổ chức nắm bắt ngữ nghĩa của nhiều lĩnh vực phong phú và phức tạp hơn một cách có kết quả. Nhiều sự đầu tư nghiên cứu về hiểu ngôn ngữ tự nhiên trong thời gian gần đây được người ta dành hết cho việc tìm ra những hình thức biểu diễn, mà về cơ bản đủ dùng trong một phạm vi rộng lớn các ứng dụng mà những ứng dụng này tự bản thân chúng còn chưa thích nghi tốt với cấu trúc đặc thù của lĩnh vực đó. Người ta khảo sát một số lượng những kỹ thuật khác nhau (hầu hết đều là những mở rộng hay cải tiến của kỹ thuật mạng ngữ nghĩa) cho mục đích này và dùng chúng vào việc phát triển những chương trình có khả năng hiểu ngôn ngữ tự nhiên trong những lĩnh vực tri thức cấp bách nhưng lý thú. Sau cùng, trong nghiên cứu gần đây (Grosz 1997, Marcus 1980), các mô hình và cách tiếp cận STOCHASTIC, mô tả cách các tập hợp từ "cùng xuất hiện" trong các môi trường ngôn ngữ, đã được dùng để khắc hoạ nội dung ngữ nghĩa của câu. Tuy nhiên, hiểu ngôn ngữ tự nhiên một cách tổng quát là vấn đề vẫn còn vượt quá giới hạn hiện nay của chúng ta.

1.3.5 Mô hình hoá hoạt động của con người

Mặc dù khá nhiều vấn đề đã nói ở trên dùng trí tuệ con người làm điểm tựa tham khảo để xem xét trí tuệ nhân tạo, thực tế đã không diễn biến theo cách mà những chương trình cần phải lấy sự tổ chức của trí óc con người làm kiểu mẫu cho chúng. Thực ra nhiều chương trình TTNT được thiết kế để giải một số bài toán cần thiết mà không cần chú ý đến tính tương tự của chúng so với kiến trúc trí óc con người. Ngay cả các hệ chuyên gia, trong khi nhận được nhiều tri thức từ các chuyên gia con người, cũng không thực sự cố gắng bắt chước những quá trình trí tuệ bên trong của con người. Nếu như sự hoạt động chỉ là những đặc tính mà theo đó một hệ thống sẽ được đánh giá, thì

có thể là không có mấy lý do để mô phỏng các phương pháp giải quyết vấn đề của con người. Trong thực tế, những chương trình sử dụng các phương pháp không theo kiểu con người để giải quyết các bài toán thường thành công hơn những chương trình theo kiểu con người. Tuy nhiên, mô hình của những hệ thống rõ ràng bắt chước một số khía cạnh của cách giải quyết vấn đề theo kiểu con người vẫn là một mảnh đất màu mỡ trong nghiên cứu cho cả hai ngành khoa học trí tuệ nhân tạo và tâm lý học.

Mô hình hóa hoạt động con người, ngoài việc cung cấp cho TTNT nhiều phương pháp luận cơ bản, cũng đã chứng tỏ được rằng nó là một công cụ mạnh để công thức hóa và thử nghiệm những lý thuyết về sự nhận thức của con người. Những phương pháp luận giải quyết vấn đề được các nhà khoa học máy tính phát triển đã đem đến cho các nhà tâm lý học một sự ẩn dụ mới để khảo sát trí tuệ con người. Hơn cả việc mở rộng được các lý thuyết về sự nhận thức trong thứ ngôn ngữ không rõ ràng sử dụng vào đầu thời kỳ nghiên cứu hay là từ bỏ được bài toán mô tả toàn bộ những hoạt động bên trong của trí óc con người (như đề nghị của các nhà hành vi học), nhiều nhà tâm lý học đã đưa ngôn ngữ và lý thuyết khoa học máy tính vào để công thức hóa các mô hình trí tuệ con người. Những kỹ thuật này không chỉ cung cấp một vốn từ vựng cho việc mô tả trí tuệ con người mà sự thể hiện trên máy tính những lý thuyết này đã tạo cho các nhà tâm lý học một cơ hội để thử nghiệm, phê bình và cải tiến một cách thực nghiệm những ý tưởng của họ (Luger 1994).

1.3.6 Lập kế hoạch và robotics

Lập kế hoạch (planning) là một khía cạnh quan trọng trong những cố gắng nhằm chế tạo ra các robot có thể thực hiện được nhiệm vụ của chúng với một trình độ nhất định và khả năng linh hoạt và phản ứng với thế giới bên ngoài. Nói một cách khác ngắn gọn, việc lập kế hoạch giả định rằng robot có khả năng thực hiện những hành động sơ cấp (atomic action) nhất định. Nó cố gắng tìm ra một chuỗi các hành động cho phép hoàn thành một công tác ở cấp độ cao hơn, chẳng hạn như đi qua một căn phòng chứa đầy những chướng ngại vật.

Có nhiều những lý do khiến cho việc lập kế hoạch trở thành một bài toán khó khăn, đáng kể nhất là kích thước quá lớn của không gian những chuỗi bước đi có thể tồn tại. Ngay cả một máy tính cực kỳ đơn giản cũng có khả năng tạo ra một số lượng khổng lồ những chuỗi bước đi có thể. Ví dụ, chúng ta hãy tưởng tượng rằng, một robot có khả năng di chuyển về phía trước, phía sau, bên phải, bên trái và cần xem xét có bao nhiêu cách khác nhau mà robot đó có thể dùng để di chuyển quanh căn phòng đó và robot phải lựa chọn một đường đi quanh chúng theo một phương pháp nào đó có hiệu quả. Viết một chương trình có khả năng tìm ra đường đi tốt nhất một cách thông minh với điều kiện như vậy, mà không bị chôn vùi bởi khối lượng khổng lồ các khả năng dự kiến, đòi hỏi phải có những kỹ thuật phức tạp để biểu diễn tri thức về không gian và kiểm soát việc tìm kiếm trong môi trường cho phép.

Một phương pháp mà con người vẫn áp dụng để lập kế hoạch là phân rã vấn đề từng bước (hierarchical problem decoposition). Nếu bạn đang lập kế hoạch cho chuyến du lịch đến Luân Đôn, thì nói chung những vấn đề như sắp xếp chuyến bay, đến sân bay, liên hệ với hãng hàng không, đi lại tại Luân Đôn sẽ được bạn xem xét một cách riêng lẻ, cho dù tất cả chúng đều là bộ phận của một kế hoạch toàn thể lớn hơn. Từng vấn đề này có thể được tiếp tục phân rã thành những vấn đề con (subproblem) nhỏ hơn như tìm một bản đồ thành phố, xem xét hệ thống giao thông và tìm một nơi ăn ở phù hợp điều kiện về tài chính… Cách làm này không những làm giảm bớt một cách hiệu quả không gian tìm kiếm mà nó còn cho phép chúng ta tiết kiệm được những kế hoạch con có thể dùng trong tương lai.

Trong khi con người lập kế hoạch một cách chẳng mấy khó khăn, thì việc tạo ra một chương trình máy tính có thể làm được công việc như vậy là một thách thức ghê gớm. Một công tác có vẻ đơn giản là phân rã một vấn đề lớn thành nhiều vấn đề con liên quan thực sự cần đến những heuristic phức tạp và kiến thức bao quát về lĩnh vực đang lập kế hoạch. Quyết định xem cần giữ lại những kế hoạch con nào và tổng quát hóa chúng như thế nào cho sự sử dụng trong tương lai là một vấn đề phức tạp tương đương.

Một robot thực hiện một dãy các hành động một cách mù quáng mà không biết phản ứng lại với những thay đổi trong môi trường của nó hoặc không có khả năng phát hiện và sửa chữa trong chính kế hoạch của nó khó có thể được người ta coi là thông minh. Thông thường, một robot sẽ phải làm thành công thức một kế hoạch dựa trên thông tin không đầy đủ và sửa chữa hành vi của nó khi thi hành kế hoạch. Robot có thể không có những giác quan thích hợp để định vị tất cả những chướng ngại vật trên con đường đi đã vạch ra. Một robot như vậy phải bắt đầu di chuyển qua căn phòng dựa vào những gì mà nó "nhận thức" được và điều chỉnh đường đi của nó khi phát hiện ra những chướng ngại vật khác. Thiết lập cho các kế hoạch cho phép có thể phản ứng lại với những điều kiện của môi trường là một nhiệm vụ chủ yếu khác trong lập kế hoạch.

Nói chung, thiết kế robot là một trong những lĩnh vực nghiên cứu của TTNT đã mang lại nhiều hiểu biết sâu sắc hỗ trợ cho phương pháp giải quyết vấn đề theo kiểu *hướng thành viên* (agent - oriented). Bị thất bại bởi những phức tạp trong việc bảo đảm độ lớn của không gian biểu diễn cũng như bởi mô hình của các thuật toán tìm kiếm dùng cho việc lập kế hoạch theo kiểu truyền thống; các nhà nghiên cứu, gồm cả Agre và Chapman (1987) và Brooks (1991a), đã phát biểu lại vấn đề lớn hơn này dựa trên các thuật ngữ về sự tương tác lẫn nhau giữa nhiều thành viên (agent) theo kiểu bán tự quản. Mỗi thành viên chịu trách nhiệm về phần đóng góp của chính nó trong nhiệm vụ của bài toán và thông qua sự phối hợp giữa chúng lời giải tổng quát sẽ hiện ra.

1.3.7 Các ngôn ngữ và môi trường dùng cho TTNT

Nghiên cứu TTNT đã tạo ra một số những sản phẩm phụ, đó là những tiến bộ trong các ngôn ngữ lập trình và các môi trường phát triển phần mềm. Vì nhiều lý do, bao gồm cả qui mô tổng thể của hầu hết các chương trình TTNT, khuynh hướng phát sinh ra các không gian khổng lồ của các thuật toán tìm kiếm và những khó khăn trong việc tiên đoán các hành vi của các chương trình điều khiển bằng heuristics, các nhà lập trình TTNT đã bị thúc ép phải xây dựng nên một tập hợp các phương pháp lập trình.

Các môi trường lập trình bao gồm cả các kỹ thuật cấu tạo tri thức (knowledge – structuring) như *lập trình hướng đối tượng* (Object-oriented Programming) và các cơ cấu tổ chức hệ chuyên gia. Các ngôn ngữ cấp cao như LISP và PROLOG, là các ngôn ngữ tích cực hỗ trợ kiểu phát triển theo module, khiến cho việc quản lý tính đồ sộ và phức tạp của chương trình dễ dàng hơn. Các gói chương trình lần tìm cho phép người lập trình tạo dựng lại quá trình thực thi của một thuật toán phức tạp và cho phép tháo gỡ những phức tạp khi tìm kiếm bằng điều khiển của heuristics. Không có công cụ kỹ thuật đó, khó mà tin được rằng người ta có thể xây dựng nên những hệ thống TTNT gây chú ý như vậy.

Kỹ thuật này hiện nay là những công cụ chuẩn dùng cho công nghệ phần mềm và có quan hệ với TTNT một cách tương đối. Những kỹ thuật khác như là lập trình hướng đối tượng, được quan tâm đáng kể cả trên lý thuyết và thực tiễn. Các ngôn ngữ phát triển cho việc lập trình TTNT gắn bó mật thiết với cấu trúc lý thuyết của lĩnh vực đó.

1.3.8 Máy học

Tuy thành công trong vai trò những máy giải quyết vấn đề, học vẫn còn là một sự nan giải đối với các chương trình TTNT. Khuyết điểm này dường như rất nghiêm trọng, đặc biệt là khi khả năng học là một trong những thành phần quan trọng nhất làm nên hành vi thông minh. Một hệ chuyên gia có thể thực hiện những tính toán lớn và rất tốn kém nhằm giải quyết một bài toán. Tuy thế, không giống như con người, nếu đưa cho nó cùng bài toán ấy hoặc một bài toán tương tự lần thứ hai, nó sẽ không nhớ lời giải lần trước. Nó thực hiện lại chuỗi tính toán đó lần nữa. Điều này lặp lại cho cả lần thứ hai, thứ ba, thứ tư và bất cứ khi nào nó giải quyết bài toán đó – hầu như không thể gọi đó là hành vi của một máy giải quyết vấn đề thông minh.

Hầu hết các hệ chuyên gia đều bị cản trở bởi tính cứng nhắc trong các chiến lược giải quyết vấn đề của chúng và sự khó khăn khi phải thay đổi khối lượng lớn mã chương trình. Giải pháp dễ thấy đối với những khó khăn này là hoặc để cho các chương trình học tập trên chính kinh nghiệm, sự tương tự và những ví dụ hoặc là "nói" cho chúng biết phải làm gì.

Tuy rằng học là một lĩnh vực khó khăn trong nghiên cứu, một vài chương trình được viết đã đề xuất rằng đây không phải là một mục tiêu không thể đạt được. Có thể một chương trình như thế gây chú ý nhất là AM - Automated Mathematician - được thiết kế để khám phá các quy luật toán học (Lenat 1977, 1982). Ban đầu người ta đưa cho AM các khái niệm và tiên đề của lý thuyết tập hợp, sau đó nó đã tìm ra những khái niệm toán học quan trọng như là lực lượng (cardinality), số học số nguyên cùng nhiều kết quả khác của lý thuyết số. AM đã phỏng đoán các lý thuyết mới bằng cách cập nhật cơ sở tri thức hiện hành của nó và sử dụng các heuristic để theo đuổi "khả năng đáng quan tâm" nhất trong hàng loạt các lựa chọn có thể.

Một nghiên cứu khác có ảnh hưởng tới dư luận của Winston về sự quy nạp các khái niệm cấu trúc, chẳng hạn như "hình cung" từ một tập hợp các ví dụ trong trò chơi thế giới của khối (Winston 1975a). Thuật toán ID3 đã tỏ ra thành công trong việc học các mẫu tổng quát từ các ví dụ (Quinlan 1986a). Menta-dendral học các luật để phiên dịch dữ liệu quang phổ khối trong hóa học hữu cơ từ các mẫu dữ liệu về các hợp chất của cấu trúc đã biết. Teiresias, một đại diện khá thông minh của các hệ chuyên gia có thể chuyển đổi lời chỉ đạo cấp cao thành các luật mới cho cơ sở dữ liệu của nó (Davis 1982). Hacke nghĩ ra các kế hoạch để thực hiện các thao tác trong trò *thế giới các khối* thông qua một quá trình lặp lại nhiều lần việc đặt ra một kế hoạch, thử nghiệm nó và hiệu chỉnh bất cứ lỗ hỏng nào phát hiện ra trong kế hoạch dự tuyển (Sussman 1975). Những nghiên cứu trong việc học trên cơ sở giải thích đã cho thấy tính hiệu quả của tri thức ưu tiên trong quá trình học (Mitchell et al. 1986, Dejong and mooney 1986).

Sự thành công của các chương trình máy học chỉ ra 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 tập trong nhiều lĩnh vực thực tế.

1.3.9 Xử lý phân tán song song và tính toán kiểu nảy sinh

Hầu hết các kỹ thuật nói đến trong tài liệu này đều sử dụng tri thức được biểu diễn rõ ràng và các thuật toán tìm kiếm được thiết kế một cách cẩn thận để cài đặt trí tuệ. Một cách tiếp cận rất khác là tìm cách xây dựng các chương trình thông minh bằng cách sử dụng các mô hình tương tự như cấu trúc nơ-ron (neuron) của bộ não con người.

Một sơ đồ neuron đơn giản gồm có một thân tế bào có rất nhiều những chỗ nhô ra theo nhánh, gọi là các tổ chức cây (dendrite), và một nhánh đơn gọi là trục (axon). Các tổ chức cây nhận tín hiệu từ các neuron khác. Khi những xung lực kết hợp này vượt quá một ngưỡng nhất định nào đó, thì neuron phát động và một xung lực, hay còn gọi là "cụm" (spike), chạy xuống trục. Các nhánh ở cuối trục hình thành nên các khớp thần kinh (synapse) với những tổ chức cây của các neuron; các khớp thần kinh có thể thuộc loại kích thích (excitatory) hay ngăn chặn (inhibitory). Một khớp thần kinh kích

thích sẽ cộng thêm vào tổng số tín hiệu đi đến neuron; còn khớp thần kinh ngăn chặn thì trừ bớt đi tổng số này.

Mô tả một neuron như vậy là quá sức đơn giản, nhưng nó thâu tóm tất cả những đặc trưng liên quan đến các mô hình tính toán neuron. Đặc biệt mỗi đơn vị tính toán tính toán một số chức năng đầu vào của nó rồi chuyển kết quả đến các đơn vị liên hệ trong mạng. Thay vì sử dụng các ký hiệu và phép toán rõ ràng, tri thức của các hệ này nảy sinh ra khỏi toàn bộ mạng các kết nối neuron và các giá trị ngưỡng.

Vì nhiều lý do, cấu trúc neuron hiện đang hết sức hấp dẫn để dùng làm cơ chế cài đặt trí tuệ. Các chương trình TTNT truyền thống có khuynh hướng dễ gãy vỡ và nhạy cảm quá đáng khi phải đương đầu với sự nhiễu loạn: thay vì giảm giá trị một cách từ từ, những chương trình như vậy thường thành công hoàn toàn hoặc thất bại hoàn toàn. Trí tuệ con người linh hoạt hơn nhiều; chúng ta có thể tiếp nhận được tốt đầu vào nhiễu loạn, chẳng hạn như nhận ra một khuôn mặt trong một căn phòng tối từ góc nhìn hẹp hay theo dõi duy nhất một cuộc đối thoại trong bữa tiệc ồn ào. Ngay cả khi không thể giải quyết được một số vấn đề, chúng ta nói chung vẫn có thể đưa ra một sự phỏng đoán có lý và coi đó như lời giải của bài toán. Do các cấu trúc neuron thâu tóm tri thức vào trong một số lượng lớn các đơn vị được nghiền thật nhỏ, nên chúng tỏ ra có triển vọng hơn trong việc đối sánh một cách toàn phần các dữ liệu nhiễu loạn và không đầy đủ.

Cấu trúc neuron cũng vững chắc hơn vì tri thức phân bố khá đồng đều xung quanh mạng. Kinh nghiệm của những người đã bị mất một phần não bộ do bệnh tật hay tai nạn đã cho thấy rằng họ không bị mất các vùng nhớ riêng biệt, mà đúng hơn là các quá trình trí não của họ phải chịu đựng nhiều sự giảm sút tổng thể.

1.4 Tổng kết chương

Phần nội dung chương I đã nêu lên vài nét về lịch sử hình thành và phát triển của khoa học TTNT, một vài định nghĩa mang tính khái quát về một lĩnh vực khoa học đầy thử thách và tiềm năng là TTNT. Những lĩnh vực ứng dụng TTNT từ rất lâu đời và vẫn đang phát triển cho đến hiện nay. Các khái niệm như lý luận, quy luật, biểu diễn, … hiện nay vẫn đang được nghiên cứu một cách cẩn thận bởi vì những nhà khoa học máy tính đòi hỏi phải hiểu chúng theo kiểu thuật toán. Trong khi đó, hoàn cảnh chính trị, kinh tế và đạo đức trên toàn cầu hiện nay buộc chúng ta phải đương đầu với trách nhiệm về hậu quả của những sáng chế hay phát minh khoa học. Sự tác động qua lại giữa những ứng dụng và những khát vọng mang tính nhân đạo hơn đối với TTNT tiếp tục đặt ra những vấn đề phong phú và đầy thách thức. Những chương tiếp theo sẽ đi sâu hơn vào những kỹ thuật được dùng trong TTNT đã đề cập đến.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 1

1. Nêu một định nghĩa theo sự khái quát của riêng bạn về Trí tuệ nhân tạo.
2. Nêu một số ưu điểm của hệ chuyên gia trên phương diện đời sống xã hội.
3. Trình bày một số hệ chuyên gia.
4. Trình bày một số ứng dụng cụ thể mà bạn biết cho việc xử lý ngôn ngữ tự nhiên đang áp dụng vào các lĩnh vực cuộc sống hiện nay.
5. Cho biết một vài lĩnh vực bài toán mà bạn thấy cần phải thiết kế một giải pháp hệ chuyên gia. Nêu các hiệu quả có thể đạt được và những khó khăn có thể gặp phải.
6. Theo ý kiến riêng của bạn, hãy trình bày một số hiệu quả có khả năng tác động tiêu cực đối với xã hội của Trí tuệ nhân tạo.
7. Thảo luận về ứng dụng của Trí tuệ nhân tạo trong lĩnh vực robotics.
8. Thảo luận về tiêu chuẩn mà Turing nêu ra trong "Trắc nghiệm Turing" nhằm quy định cho một phần mềm máy tính được coi là "thông minh". Nêu quan điểm của riêng bạn về tiêu chuẩn đối với một phần mềm máy tính "thông minh".

Chương 2 CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

2.1 ĐỊNH NGHĨA TÌM KIẾM TRÊN KHÔNG GIAN TRẠNG THÁI

Một không gian trạng thái (state space) được biểu diễn bằng một nhóm gồm bốn yếu tố [N, A, S, GD], trong đó:

- N (Node) là tập hợp các nút hay các trạng thái của đồ thị. Tập này tương ứng với các trạng thái trong quá trình giải bài toán.
- A (Arc) là tập các cung (hay các liên kết) giữa các nút. Tập này tương ứng với các bước trong quá trình giải bài toán.
- S (Start) là một tập con không rỗng của N, chứa (các) trạng thái ban đầu của bài toán.
- GD (Goal Description) là một tập con không rỗng của N, chứa (các) trạng thái đích của bài toán. Các trạng thái trong GD 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.
    + Đặc tính của đường đi được hình thành trong quá trình tìm kiếm.

Đường đi của lời giải (Solution path) là đường đi qua đồ thị này từ một nút trong S đến một nút trong GD.

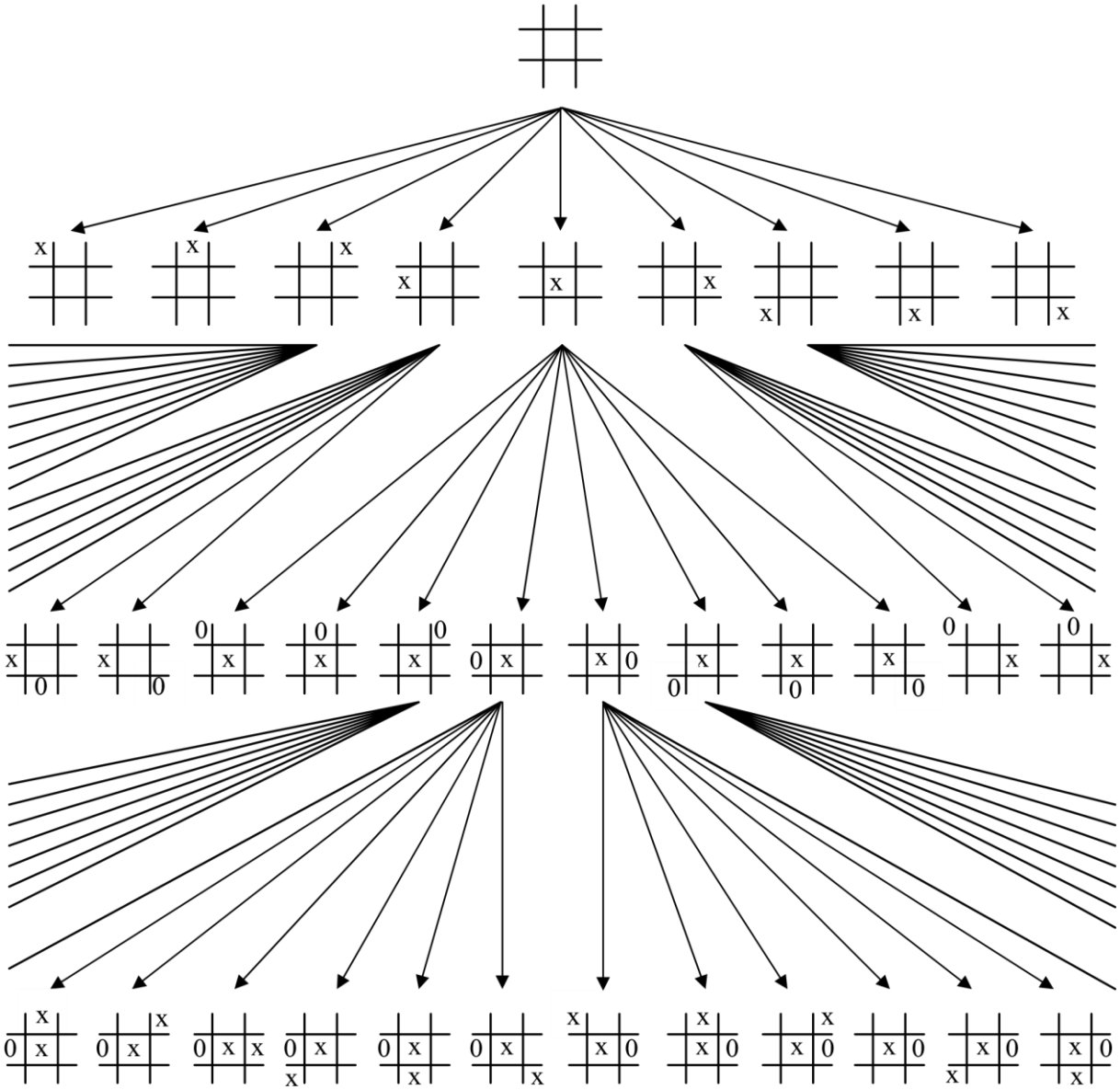
Một đích có thể mô tả như một trạng thái, như bảng thắng cuộc trong trò chơi Tic - tac - toe hay một cấu hình đích trong trò đố 8 ô.

Mặt khác, đích cũng có thể được mô tả như một đặc tính nào đó của chính đường đi lời giải. Như trong bài toán di chuyển của người bán hàng, quá trình tìm kiếm sẽ kết thúc khi tìm được đường đi ngắn nhất qua tất cả các nút của đồ thị. Trong bài toán phân tích ngữ pháp, đường đi của quá trình phân tích thành công đối với một câu sẽ là trạng thái đích.

Các cung của không gian trạng thái tương ứng với các bước trong quá trình giải bài toán và các đường đi qua không gian này sẽ biểu diễn các lời giải trong các giai đoạn hoàn thành khác nhau. Các đường đi sẽ được khảo sát, bắt đầu từ trạng thái xuất phát và tiếp tục đi xuyên qua đồ thị, cho đến khi gặp được một mô tả đích thỏa mãn hoặc đến khi không đi được nữa. Việc tạo ra các trạng thái mới trên đường đi sẽ được thực hiện bằng cách áp dụng các thao tác, như "các di chuyển hợp lệ" trong trò chơi hay quy tắc suy diễn trong bài toán logic và trong hệ chuyên gia chẳng hạn, vào các trạng thái hiện hữu trên đường đi.

Nhiệm vụ của thuật toán tìm kiếm là tìm ra đường đi lời giải trong một không gian bài toán như vậy. Thuật toán tìm kiếm phải lưu vết các đường đi dẫn từ nút xuất phát đến nút đích, vì các đường đi này chứa hàng loạt thao tác dẫn đến lời giải của bài toán. Một đặc trưng phổ biến của đồ thị, đồng thời là vấn đề phát sinh khi thiết kế thuật toán tìm kiếm trên đồ thị, là đôi khi tiếp cận một trạng thái nào đó bằng nhiều đường đi khác nhau.

Thí dụ: Trò chơi tic–tac-toe



Hình 2.1 – Một phần không gian trò chơi Tic-tac-toe

Cho trước một tình huống bàn cờ nào đó, chỉ có một số hữu hạn các nước đi mà người chơi có thể đi. Bắt đầu ván đấu bằng bàn cờ trống, đấu thủ thứ nhất có thể đặt ký hiệu nước đi X vào bất cứ ô nào trong 9 ô trống của bàn cờ. Mỗi trong số những nước đi này tạo ra một bàn cờ khác cho phép đấu thủ thứ hai đến lượt mình đi sẽ có thể chọn 8

cách đặt ký hiệu nước đi O của mình vào... Và sẽ cứ luân phiên như thế. Chúng ta có thể xem mỗi hình thái bàn cờ này như là một *nút* trong đồ thị, các *liên kết* của đồ thị biểu diễn các nước đi hợp lệ từ một thế cờ này sang thế cờ khác. Cấu trúc kết quả tạo thành một *đồ thị không gian trạng thái* cho bài toán. Và lúc đó, một chiến lược chơi hiệu quả sẽ là việc tìm kiếm xuyên suốt đồ thị để tìm ra đường đi dẫn đến kết thúc thắng lợi.

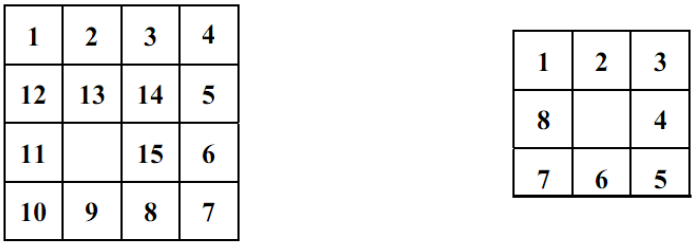
Cách biểu diễn không gian trạng thái của Tic-tac-toe như trong hình trên. Trạng thái xuất phát (S) là một bảng rỗng và đích kết thúc (GD) là một trạng thái dạng bảng có 3 ô x trong một hàng, một cột hay một đường chéo (giả sử đích là một thắng cuộc đối với X). Đường đi từ trạng thái xuất phát đến trạng thái đích sẽ cho ta hàng loạt nước đi cho một ván thắng.

Các trạng thái trong không gian này là các cách sắp xếp tập hợp {ô rỗng, X, O} trong chín ô trống của trạng thái bắt đầu, như vậy ta sẽ có 3^9 cách sắp xếp, nhưng hầu hết chúng không bao giờ xảy ra trong một ván chơi thực tế. Các cung sẽ được tạo ra bởi các nước đi đúng luật trong cuộc chơi, sau khi luân phiên đặt X và O vào một vị trí chưa dùng đến. Không gian trạng thái này là một đồ thị chứ không phải một cây, vì có thể đạt đến một số trạng thái nằm ở mức thứ ba và các mức sâu hơn bằng các đường đi khác nhau. Tuy vậy không có vòng lặp nào trong không gian trạng thái này vì các cung có hướng của đồ thị không cho phép đi lại một nước đã đi, nghĩa là không được phép "đi ngược trở lại" một cấu trúc một khi đã đi đến trạng thái nào đó. Do đó, không cần phải kiểm tra về vòng lặp trong việc tạo ra đường đi. Một cấu trúc đồ thị có đặc tính như vậy gọi là đồ thị có hướng không lặp lại hay DAG (directed acyclic graph) và phổ biến trong các quá trình tìm kiếm không gian trạng thái.

Việc biểu diễn không gian trạng thái cho chúng ta một phương tiện để xác định mức độ phức tạp của bài toán. Trong Tic-Tac-Toe có chín cách đi đầu tiên với tám khả năng có thể xảy ra đối với mỗi cách đi đó; tiếp theo là bảy khả năng có thể xảy ra nữa cho từng cách đi ở lượt thứ hai, … Như vậy có tất cả 9 x 8 x 7 x … hay 9! đường đi khác nhau có thể được tạo ra. Mặc dù máy tính hoàn toàn có đủ khả năng thăm dò hết số lượng đường đi này (362880), nhưng nhiều bài toán quan trọng có mức độ phức tạp theo qui luật số mũ hay giai thừa với qui mô lớn hơn nhiều. Cờ vua có 10^{120} cách đi có thể xảy ra cho một ván chơi, cờ đam có 10^{40} cách đi, một số trong đó có thể không bao giờ xảy ra ở một ván chơi thực tế. Các không gian này rất khó hoặc không thể nào khảo sát cho đến hết được. Chiến lược tìm kiếm đối với một không gian lớn như vậy thường phải dựa vào các heuristic để làm giảm bớt mức độ phức tạp cho quá trình tìm kiếm.

Thí dụ: Trò đố 8 ô

Nguyên bản của trò chơi là trò đố 15 ô như hình 2.2, có 15 viên gạch đánh số khác nhau được đặt vừa vào 16 ô vuông theo bảng. Có một ô vuông để trống nên các viên gạch đó có thể di chuyển loanh quanh để tạo ra các sắp xếp khác nhau. Mục tiêu là tìm ra một chuỗi bước di chuyển các viên gạch vào ô trống để sắp xếp bảng thành một cấu hình đích nào đó. Không gian trạng thái bài toán đủ lớn để xem xét (16! nếu các trạng thái đối xứng được xem xét riêng biệt).

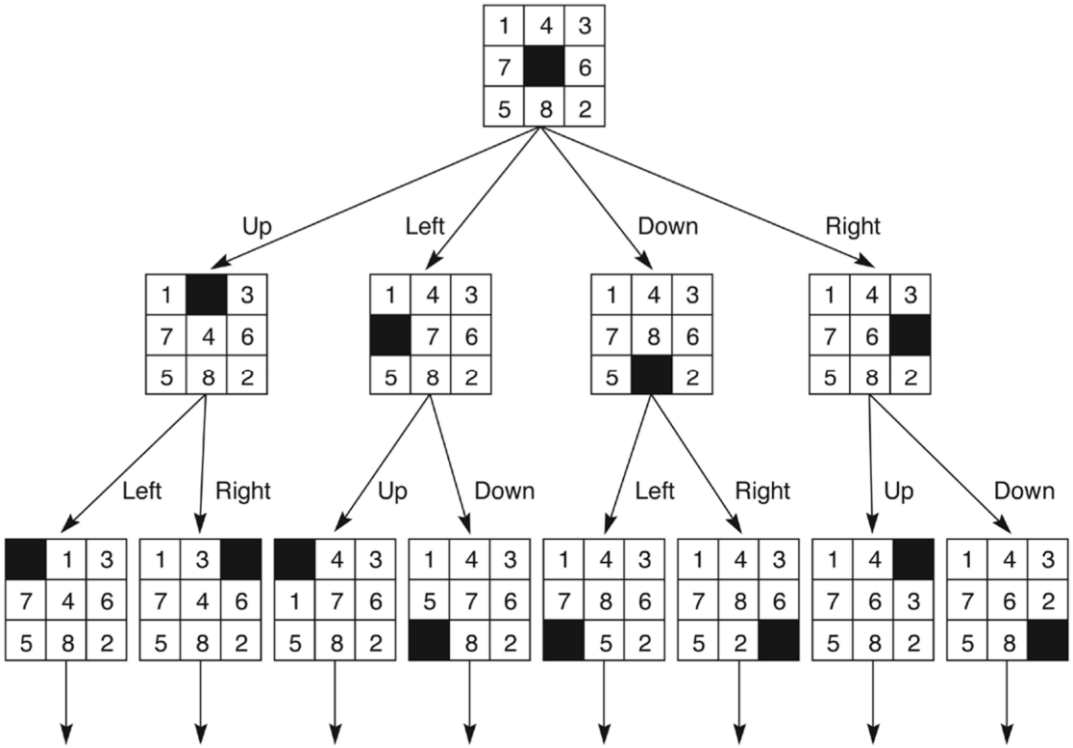


Hình 2.2 – Trò đố 15 ô và trò đố 8 ô

Trò đố 8 ô là một phiên bản với kích thước 3 x 3 của trò đố 15 ô, vì nó tạo ra không gian trạng thái nhỏ hơn so với trò đố 15 ô nên thường được dùng cho nhiều ví dụ. Các cách di chuyển đúng luật là:

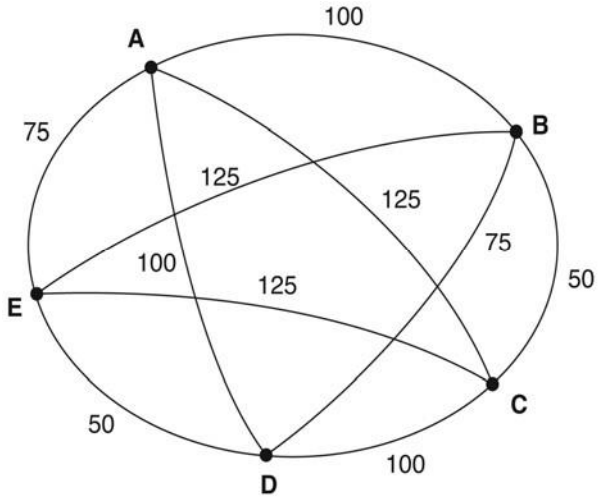
1. Di chuyển ô trống lên phía trên (Up)
2. Di chuyển ô trống về bên phải (Right)
3. Di chuyển ô trống xuống phía dưới (Down)
4. Di chuyển ô trống về bên trái (Left)

Để áp dụng một di chuyển, phải bảo đảm rằng không di chuyển ô trống ra ngoài bảng. Do đó, tất cả bốn cách di chuyển này không phải lúc nào cũng có thể áp dụng. Nếu đã xác định một trạng thái xuất phát và một trạng thái đích thì có thể đưa ra một sơ đồ không gian trạng thái của quá trình giải toán. Giống như trò chơi Tic-tac-toe không gian trạng thái của trò đố 8 ô cũng là một đồ thị (với hầu hết các trạng thái đều có nhiều nút cha), nhưng khác Tic-tactoe là có vòng lặp. Mô tả đích (GD) của không gian trạng thái này là một trạng thái cụ thể, tức một cấu hình bảng cụ thể nào đó. Khi tìm được trạng thái này trên đường đi, quá trình tìm kiếm kết thúc. Đường đi từ trạng thái xuất phát đến trạng thái đích là dãy các bước di chuyển theo yêu cầu.



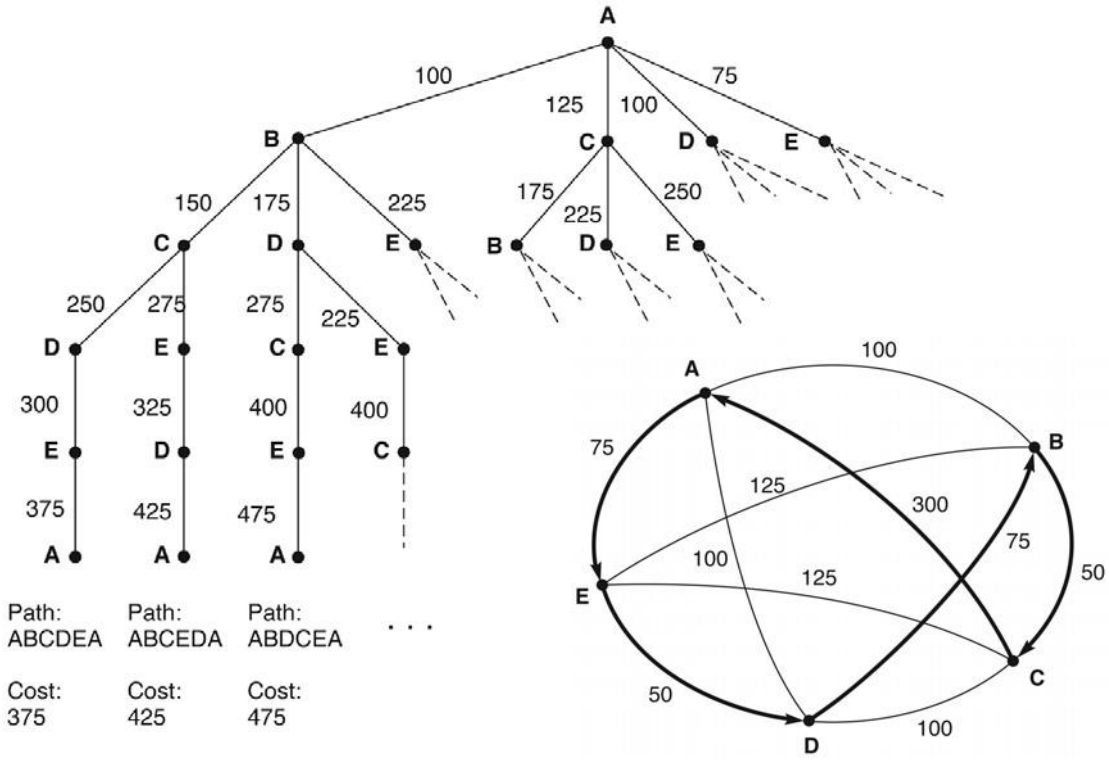
Hình 2.3 – Không gian trạng thái trong trò đố 8 ô.

Thí dụ: Đường đi của người bán hàng



Hình 2.4 – Bài toán đường đi của người bán hàng

Giả sử có một người bán hàng phải giao hàng cho năm thành phố và sau đó phải quay về nhà. Đích của bài toán là tìm con đường ngắn nhất cho người giao hàng đó đi đến từng thành phố, rồi quay về thành phố xuất phát. Hình bên dưới là một trường hợp của bài toán này.



Hình 2.5 – Không gian trạng thái cho quá trình tìm kiếm đường đi

Các nút trong đồ thị biểu diễn các thành phố, mỗi cung đều được ghi kèm một trọng số cho biết phí tổn đi lại trên cung đó. Chi phí này có thể là số km phải di chuyển bằng ô tô hay giá vé máy bay giữa hai thành phố. Để thuận tiện, chúng ta giả thiết người bán hàng sống ở thành phố A và sẽ quay về đây, mặc dù giả thiết này chỉ giảm từ bài toán n thành phố xuống bài toán (n-1) thành phố.

Đường đi [A,D,C,B,E,A], với chi phí tương ứng 450 km là ví dụ về một con đường có thể đi. Mô tả đích yêu cầu một con đường có chi phí thấp nhất. Chú ý rằng mô tả đích này là một đặc tính của toàn bộ con đường, không phải là một trạng thái đơn lẻ. Đây là mô tả đích dạng thứ hai trong định nghĩa về tìm kiếm trong không gian trạng thái đã nói ở trên.

2.2 CÁC CHIẾN LƯỢC TÌM KIẾM TRÊN KHÔNG GIAN TRẠNG THÁI

2.2.1 Tìm kiếm hướng từ dữ liệu và tìm kiếm hướng từ mục tiêu

Một không gian trạng thái có thể được tìm kiếm theo hai hướng: từ các dữ liệu cho trước của một bài toán hướng đến mục tiêu hay từ mục tiêu hướng ngược về các dữ liệu.

Trong *tìm kiếm hướng từ dữ liệu* (data-driven search) hay còn gọi là *suy diễn tiến* (forward chaining), tiến trình giải bài toán bắt đầu với các sự kiện cho trước của bài toán và một tập các luật hợp thức dùng thay đổi trạng thái. Quá trình tìm kiếm được

thực hiện bằng cách áp dụng các luật vào các sự kiện để tạo ra các sự kiện mới, sau đó các sự kiện mới này lại được áp dụng các luật để sinh ra các sự kiện mới hơn cho đến khi chúng có thể đưa ra một giải pháp thỏa mãn điều kiện mục tiêu.

Cũng có thể dùng một phương pháp khác: bắt đầu từ mục tiêu mà chúng ta muốn giải quyết, khảo sát xem có thể dùng những luật hợp thức nào để đạt đến mục tiêu này, đồng thời xác định xem các điều kiện nào phải được thỏa mãn để có thể áp dụng được chúng. Các điều kiện này sẽ trở nên những mục tiêu mới, còn gọi là các mục tiêu phụ (subgoals) trong tiến trình tìm kiếm. Quá trình tìm kiếm cứ tiếp tục như thế, hoạt động theo chiều ngược, qua hết các đích phụ kế tiếp nhau cho đến khi gặp các sự kiện thực của bài toán. Phương pháp này sẽ tìm ra một chuỗi các bước đi hay các luật, dẫn từ các dữ liệu đến một đích, mặc dù nó thực hiện theo thứ tự ngược lại. Phương pháp này gọi là *tìm kiến hướng từ mục tiêu* (goal-driven search) hay *suy diễn lùi* (backward chaining) và nó gợi cho chúng ta về một thủ thuật đơn giản khi cố giải bài toán mê cung bằng cách đi ngược từ cuối lên đầu.

Tóm lại, suy luận hướng từ dữ liệu nắm lấy các sự kiện của bài toán rồi áp dụng các luật và các bước đi hợp thức để tạo ra những sự kiện mới dẫn đến một đích; suy luận hướng từ mục tiêu thì tập trung vào đích, tìm các luật có thể dẫn đến đích, rồi lần ngược ra các luật và các đích phụ kế tiếp để đi đến các sự kiện cho trước của bài toán.

Trong bước phân tích cuối cùng, các hệ giải bài toán hướng từ mục tiêu cũng như hướng từ dữ liệu đều tìm kiếm trong cùng một đồ thị không gian trạng thái; tuy nhiên thứ tự và số lượng trạng thái thực sự cần tìm có thể khác nhau.

Những hệ giải toán đã được biên soạn bằng cách dùng một trong hai phương pháp hướng dữ liệu và hướng mục tiêu quyết định theo cách nào tuỳ thuộc cấu trúc của bài toán phải giải. Nên tìm kiếm hướng từ mục tiêu nếu:

1. Mục tiêu hay giả thiết được cho trong phát biểu bài toán hoặc có thể dễ dàng công thức hoá. Trong một hệ chứng minh định lý toán học chẳng hạn, mục tiêu chính là định lý phải chứng minh. Nhiều hệ thống chẩn đoán bệnh cân nhắc các chẩn đoán có tiềm năng theo cách hệ thống hoá, xác nhận hay loại bỏ chúng bằng phép suy luận hướng đích.
2. Có một số lượng lớn các luật phù hợp với các sự kiện của bài toán, do đó tạo ra một số lượng ngày càng nhiều các kết luận, tức là các đích. Việc chọn sớm một đích có thể loại bỏ hầu hết các nhánh rẽ này, làm cho việc tìm kiếm hướng từ mục tiêu có hiệu quả hơn nhờ thu gọn không gian. Trong một hệ chứng minh định lý toán chẳng hạn, số lượng luật chứa trong một định lý cho trước ít hơn rất nhiều so với số lượng luật có thể áp dụng cho toàn bộ tập các tiên đề.
3. Các dữ liệu của bài toán không được cho trước, nhưng hệ giải toán yêu cầu phải có. Trong trường hợp này, tìm kiếm hướng từ mục tiêu có thể giúp thu thập các

dữ liệu dẫn đường. Trong chương trình chẩn đoán y khoa chẳng hạn, có thể áp dụng hàng loạt các phép thử chẩn đoán. Các bác sĩ chỉ dùng những phép thử nào cần thiết để xác nhận hay loại bỏ một giả thiết cụ thể.

Như vậy tìm kiếm hướng từ mục tiêu sẽ dùng những hiểu biết về mục tiêu cần có để hướng dẫn quá trình tìm kiếm thông qua các luật thích hợp, đồng thời loại bỏ nhiều nhánh của không gian đó.

Tìm kiếm hướng từ dữ liệu thích hợp với bài toán thuộc loại:

1. Tất cả hay hầu hết các dữ liệu đầu được cho trước trong phát biểu ban đầu của bài toán. Các bài toán diễn dịch thường phù hợp với khuôn mẫu này bằng cách đưa ra một tập hợp các dữ liệu và yêu cầu hệ thống cung cấp một cách diễn dịch bậc cao. Các hệ thống dùng để phân tích dữ liệu cụ thể (như các trình PROSPECTOR hay DIPMETER dùng để diễn dịch các dữ liệu địa chất hoặc để tìm xem những khoáng chất nào có nhiều khả năng tìm thấy tại một địa điểm) thích hợp với cách tiếp cận hướng dữ liệu.
2. Có một số lượng lớn đích có tiềm năng, nhưng chỉ có vài cách sử dụng các sự kiện và các thông tin cho trước trong trường hợp bài toán cụ thể. Chương trình DENDRAL, một hệ chuyên gia dùng để tìm kiếm cấu trúc phân tử của các hợp chất hữu cơ dựa vào công thức hóa học, dữ liệu phân tích khối phổ và dựa vào các kiến thức hóa học là một ví dụ về trường hợp này. Đối với một hợp chất hữu cơ bất kỳ, có thể có rất nhiều cấu trúc phân tử. Tuy nhiên, dữ liệu phân tích khối phổ của hợp chất sẽ cho phép DENDRAL loại bỏ hầu hết, chỉ giữ lại một ít trong cấu trúc đó.
3. Rất khó hình thành nên một đích hay một giả thuyết. Trong việc tham vấn hệ chuyên gia DENDRAL chẳng hạn, có rất ít hiểu biết ban đầu về cấu trúc có thể có của một hợp chất.

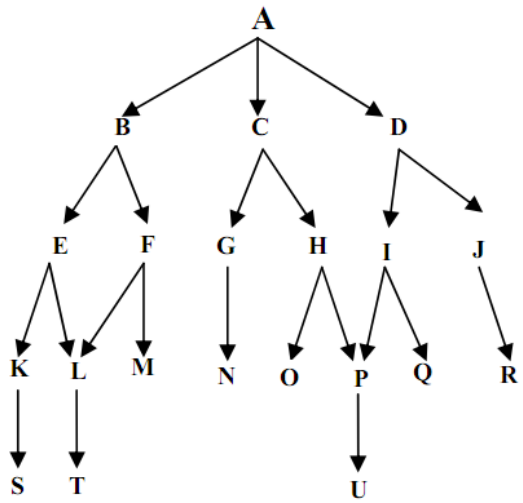
2.2.2 Tìm kiếm trên đồ thị

Khi giải bài toán bằng tìm kiếm hướng từ mục tiêu hay hướng từ dữ liệu, dù dùng phương pháp nào thì hệ giải toán cũng phải tìm một đường đi từ trạng thái xuất phát dẫn đến đích trong đồ thị không gian trạng thái. Trình tự của các cung trong đường đi này tương ứng với các bước theo thứ tự của lời giải. Nếu hệ giải toán đã được cung cấp một "nhà tiên tri" hay một cơ chế không thể sai lầm để chọn đường đi lời giải, thì việc tìm kiếm sẽ không còn cần thiết nữa. Hệ giải toán này sẽ vận hành một cách chính xác qua không gian đó để đến một đích đúng yêu cầu, tạo nên một đường đi khi nó đi qua. Vì không bao giờ có các "nhà tiên tri" như vậy, nên hệ giải toán phải xem xét nhiều đường đi qua không gian cho đến khi tìm được đích. *Lần ngược* (backtrack) là một kỹ thuật dùng để thử một cách có hệ thống tất cả những đường đi trong một không gian trạng thái.

Trong phần kế tiếp sẽ giới thiệu các thuật toán tìm kiếm giống như thuật toán lần ngược, các thuật toán này sẽ dùng các danh sách để theo dõi các trạng thái trong không gian tìm kiếm. Những thuật toán này bao gồm *tìm kiếm sâu* (Depth-First-Search), *tìm kiếm rộng* (Breadth-First-Search), khác với thuật toán lần ngược ở chỗ chúng sẽ cung cấp một cơ chế linh hoạt hơn cho việc thực hiện các chiến lược tìm kiếm đồ thị khác nhau.

2.2.2.1 Tìm kiếm sâu và tìm kiếm rộng

Cùng với việc định hướng tìm kiếm (hướng từ dữ liệu hay hướng từ mục tiêu), một thuật toán tìm kiếm còn phải xác định thứ tự mà theo đó các trạng thái sẽ được khảo sát trong cây hoặc đồ thị. Phần này sẽ đề cập đến hai khả năng đối với thứ tự xem xét các nút trong đồ thị: *Tìm kiếm sâu (DFS) và tìm kiếm rộng (BFS).*



Hình 2.6 – Đồ thị ví dụ cho tìm kiếm rộng và tìm kiếm sâu

Chúng ta cùng khảo sát đồ thị trong hình trên. Các trạng thái được ký hiệu (A,B,C,…) sao cho chúng sẽ được tham khảo theo các ký hiệu đó trong quá trình thảo luận này. Trong tìm kiếm sâu, khi một trạng thái được xem xét, tất cả các con của nó được xét đến rồi đến các thế hệ sau của các con đó đều được xem xét ưu tiên trước bất kỳ một trạng thái anh em nào của nó. Tìm kiếm sâu sẽ tiến sâu hơn vào trong không gian tìm kiếm bất kỳ khi nào còn có thể. Chỉ khi nào không tìm được các con cháu xa hơn của trạng thái đó thì mới xem xét đến các trạng thái anh em của nó. Tìm kiếm sâu sẽ tiến hành kiểm tra các trạng thái trong đồ thị hình 2.6 theo thứ tự A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R. Thuật toán lần ngược cũng đã thực hiện theo kiểu tìm kiếm sâu.

Ngược lại, tìm kiếm rộng sẽ khảo sát không gian này theo từng mức. Chỉ đến khi trong một mức cho trước không còn một trạng thái nào để khảo sát thì thuật toán mới chuyển sang mức tiếp theo. Tìm kiếm rộng trong đồ thị hình trên sẽ xem xét các trạng thái theo thứ tự A, B, C, D, E, F, G, H, I, J, K, L, M, O, P, Q, R, S, T, U.

Chúng ta thực hiện tìm kiếm rộng bằng cách dùng các danh sách *Open* (mở) và *Closed* (đóng) để theo dõi tiến độ trong không gian trạng thái đó. Danh sách Open sẽ liệt kê các trạng thái vừa được sinh ra, nhưng con của chúng chưa được khảo sát. Thứ tự mà các trạng thái bị loại ra khỏi Open sẽ xác định thứ tự tìm kiếm. Danh sách Closed thì ghi các trạng thái đã được xem xét rồi.

Quá trình tìm kiếm rộng trên đồ thị 2.6 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]; (vì L đã có trong Open); Closed = [F,E,D,C,B,A]
8. …

Thuật toán tìm kiếm rộng (BFS):

Procedure breadth-first-search;

Begin % *khởi đầu*

Open:= [start]; Closed:= [ ];

While Open ≠ [ ] do % còn các trạng thái chưa khảo sát

Begin

Lấy trạng thái X ngoài cùng bên trái khỏi Open;

If X là một đích *then* trả lời kết quả (thành công) % *tìm thấy đích* else begin

Phát sinh các con của X;

Đưa X vào Closed;

Bỏ qua các con của X trong Open hoặc Closed; %*kiểm tra vòng lặp*

Đưa các con còn lại vào đầu bên **phải** của Open % *hàng chờ*

end;

End;

Trả lời kết quả (thất bại); % *không còn trạng thái nào*

End;

Các trạng thái con sinh ra nhờ các luật suy diễn, các nước đi hợp lệ của trò chơi hoặc các toán tử chuyển trạng thái. Mỗi lần lặp lại sẽ tạo ra tất cả các con của trạng thái X và bổ sung chúng vào danh sách Open. Chú ý rằng danh sách Open được duy trì dưới dạng *một hàng đợi* (queue), tức cấu trúc dữ liệu "vào trước ra trước" (first – in – first – out : FIFO). Các trạng thái được bổ sung vào bên phải danh sách và được lấy ra từ bên trái. Cách sắp xếp theo hàng này sẽ sắp xếp việc tìm kiếm đến các trạng thái nằm trong danh sách Open lâu nhất, làm cho quá trình trở thành tìm kiếm rộng. Các trạng thái con đã được khảo sát rồi (đã xuất hiện trong danh sách Open hoặc danh sách Closed) đều bị loại bỏ. Nếu thuật toán kết thúc vì điều kiện của vòng lặp "while" không còn được thoả mãn nữa (Open = [ ]) tức là nó đã tìm kiếm xong toàn bộ đồ thị mà không tìm thấy đích mong muốn: Cuộc tìm kiếm thất bại.

Vì tìm kiếm rộng xem xét mọi nút ở từng mức của đồ thị trước khi đi sâu vào không gian đó nên tất cả các trạng thái đều được tiếp cận đến đầu tiên theo con đường ngắn nhất kể từ trạng thái xuất phát. Do đó tìm kiếm rộng sẽ đảm bảo tìm được đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái đích. Hơn nữa, vì tất cả các trạng thái đều được tìm thấy trước theo con đường ngắn nhất cho nên bất kỳ trạng thái gặp lần thứ hai nào cũng được tìm thấy theo con đường có chiều dài bằng hoặc lớn hơn. Vì không có cơ hội để các trạng thái trùng lặp được tìm thấy theo một đường đi tốt hơn, nên thuật toán này sẽ loại bỏ được mọi trạng thái trùng lặp.

Tiếp theo, chúng ta sẽ xây dựng thuật toán tìm kiếm sâu trên không gian trạng thái. Trong khi xem xét thuật toán này, cần lưu ý một điều là các trạng thái con cháu đều được bổ sung vào hay bị loại bỏ ra từ đầu bên trái của danh sách Open: Danh sách Open được duy trì dưới dạng một *ngăn xếp* (stack), tức cấu trúc "vào sau ra trước" (last – in – first – out : LIFO). Việc tổ chức danh sách Open theo dạng ngăn xếp sẽ hướng quá trình tìm kiếm nhằm vào các trạng thái được sinh ra mới nhất, làm cho tìm kiếm đó phát triển theo chiều sâu.

Quá trình tìm kiếm sâu trên đồ thị 2.6 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. …

Tìm kiếm sâu được tiến hành bằng cách cải biến thuật toán đã dùng cho tìm kiếm rộng.

Thuật toán tìm kiếm sâu (DFS):

Procedure depth – first –search;

Begin % *khởi đầu*

Open:= [start]; Closed:= [ ];

While Open ≠ [ ] do % còn các trạng thái chưa khảo sát

Begin

Lấy trạng thái X ngoài cùng bên trái khỏi Open;

If X là một đích *then* trả lời kết quả (thành công) % *tìm thấy đích*
else begin

Phát sinh các con của X;

Đưa X vào Closed;

Bỏ qua các con của X trong Open hoặc Closed; %*kiểm tra vòng lặp*

Đưa các con còn lại vào đầu bên **trái** của Open % *ngăn xếp*

end;

End;

Trả lời kết quả (thất bại); % *không còn trạng thái nào*

End;

Khác với tìm kiếm rộng, tìm kiếm sâu không đảm bảo sẽ tìm được đường đi ngắn nhất đến một trạng thái gặp lần đầu. Vào sâu trong tìm kiếm này, có thể tìm thấy một đường đi khác dẫn đến trạng thái bất kỳ. Nếu độ dài đường đi là quan trọng đối với một trình giải toán, thì khi thuật toán gặp một trạng thái trùng lặp, nó cần giữ lại phiên bản đã gặp theo đường đi ngắn nhất. Có thể thực hiện điều này bằng cách lưu giữ mỗi trạng thái dưới dạng một bộ ba: (trạng thái đó, trạng thái cha, chiều dài đường đi). Khi phát sinh trạng thái con, chỉ cần tăng trị số độ dài đường đi lên một đơn vị và cất giữ lại cùng với trạng thái con đó. Nếu một trạng thái con bắt gặp theo nhiều đường đi, thông tin này có thể được sử dụng để giữ lại phiên bản tốt nhất. Cần chú ý, việc giữ lại phiên bản tốt nhất của một trạng thái trong một tìm kiếm sâu đơn giản sẽ không đảm bảo được việc tìm thấy đích theo đường đi ngắn nhất.

Cũng giống như việc lựa chọn giữa tìm kiếm hướng dữ liệu hay hướng mục tiêu khi đánh giá một đồ thị, việc sử dụng tìm kiếm rộng hay tìm kiếm sâu cũng tùy thuộc vào

bài toán cụ thể phải giải. Những đặc trưng quan trọng bao gồm: việc tìm đường đi ngắn nhất dẫn đến đích, việc phân nhánh của không gian trạng thái, các tài nguyên về thời gian và không gian có sẵn, chiều dài trung bình của các đường dẫn đến nút đích và cả việc liệu chúng ta muốn có tất cả các lời giải hay chỉ cần lời giải đầu tiên tìm thấy. Để đưa ra câu trả lời cho các vấn đề này, mỗi cách đều có những ưu điểm và khuyết điểm riêng.

Tìm kiếm rộng: Vì lúc nào cũng xem xét tất cả các nút ở mức n rồi mới chuyển sang mức n+1 nên tìm kiếm rộng bao giờ cũng tìm được đường đi ngắn nhất đến một nút đích. Trong những bài toán nếu biết rõ có một lời giải đơn giản thì lời giải này sẽ được tìm thấy. Tuy nhiên, nếu bài toán có hệ số phân nhánh không gian lớn, nghĩa là các trạng thái có số lượng con cháu trung bình tương đối cao thì sự bùng nổ tổ hợp này có thể gây cản trở cho việc tìm kiếm một lời giải. Lý do là vì tất cả các nút chưa được mở rộng với mỗi mức tìm kiếm đều phải được giữ lại trong danh sách Open. Do đó, đối với không gian trạng thái có hệ số phân nhánh cao, điều này có thể trở nên rất phức tạp.

Độ phức tạp của không gian tìm kiếm rộng được đo theo số lượng trạng thái trong danh sách Open, đó là một hàm mũ của chiều dài đường đi tại mỗi thời điểm bất kỳ. Nếu mỗi trạng thái trung bình có B con thì số lượng trạng thái ở một mức sẽ bằng B lần số trạng thái ở mức trước đó. Như vậy sẽ có B^n trạng thái ở mức n. Tìm kiếm rộng sẽ phải đưa tất cả trạng thái này vào danh sách Open khi nó bắt đầu xem xét mức n. Điều này có thể không được phép nếu các đường đi lời giải quá dài.

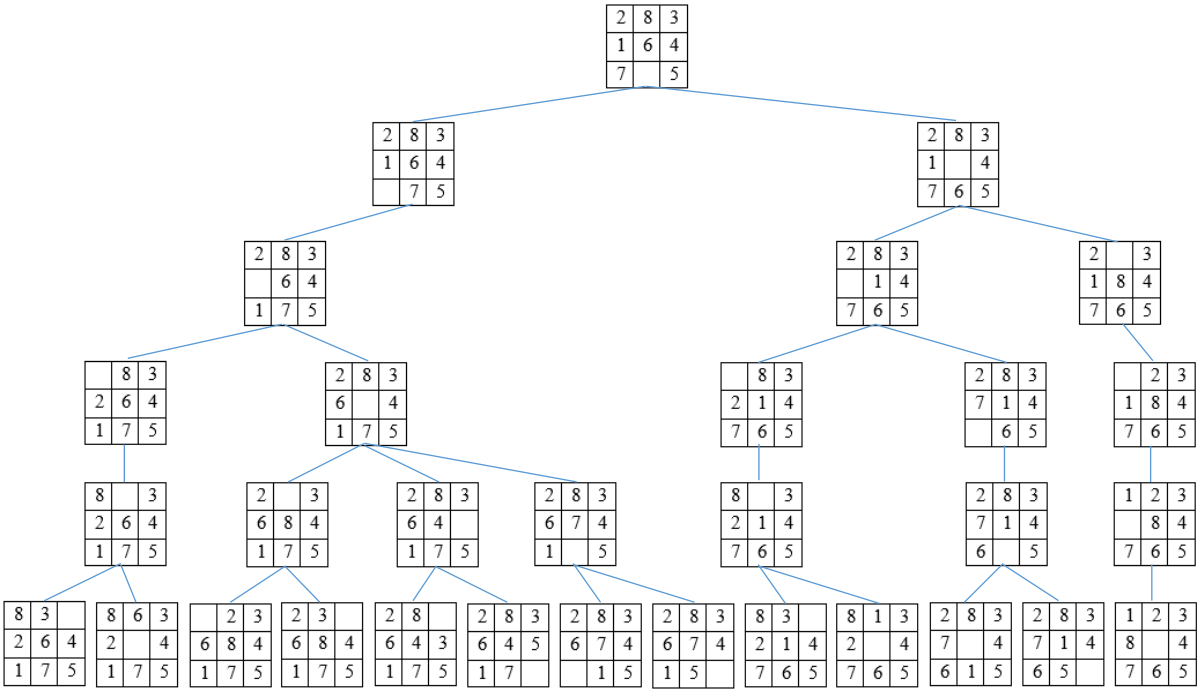
Tìm kiếm sâu: Tìm kiếm sâu sẽ nhanh chóng đi sâu vào không gian tìm kiếm. Nếu biết rõ đường đi lời giải sẽ dài, tìm kiếm sâu sẽ không mất thời gian cho việc tìm kiếm một số lớn các trạng thái "cạn" trong đồ thị. Mặc khác, tìm kiếm sâu cũng có thể sa lầy vào độ sâu "vô ích" khi bỏ qua những đường đi ngắn hơn dẫn đến đích, thậm chí có thể bị sa lầy trong một đường đi dài vô tận mà không dẫn đến đích nào cả.

Tìm kiếm sâu hiệu quả hơn nhiều đối với những không gian trạng thái có hệ số phân nhánh lớn vì nó không phải giữ tất cả các nút ứng với một mức trong danh sách Open. Độ phức tạp của không gian tìm kiếm sâu là một hàm tuyến tính của độ dài đường đi. Tại mỗi mức, Open chỉ chứa con của một tráng thái duy nhất. Nếu đồ thị có số lượng con trung bình là B đối với mỗi trạng thái thì tìm kiếm sâu có độ phức tạp của không gian là B x n trạng thái để đạt đến mức sâu n trong không gian đó.

Câu trả lời tốt nhất cho việc chọn dùng tìm kiếm sâu hay tím kiếm rộng là khảo sát không gian bài toán một cách cẩn thận. Trong cờ vua chẳng hạn, tìm kiếm rộng là không thể được. Trong các trò chơi đơn giản hơn thì tìm kiếm rộng không những là giải pháp có thể mà còn là giải pháp duy nhất để tránh thất bại.

2.2.2.2 Tìm kiếm sâu bằng cách đào sâu nhiều lần – Tìm kiếm sâu dần

Một sự kết hợp hai thuật toán này là sử dụng một giới hạn độ sâu cho quá trình tìm kiếm sâu. Giới hạn sâu sẽ buộc phải chịu thất bại trên một con đường tìm kiếm khi đường đó dẫn tới độ sâu quá một mức nào đó. Điều này gây ra một quá trình quét không gian trạng thái giống như tìm kiếm rộng tại độ sâu đó. Khi biết chắc có một lời giải nằm trong một phạm vi nào đó hoặc khi bị giới hạn về thời gian, như trong không gian quá rộng của cờ vua chẳng hạn, thì biện pháp hạn chế số lượng trạng thái phải được xem xét đến. Lúc đó tìm kiếm sâu với độ sâu giới hạn có thể là thích hợp nhất. Hình sau đây trình bày một tiến trình tìm kiếm sâu của bài toán trò đố 8 ô, trong đó giới hạn sâu bằng 5 sẽ tạo ra quá trình quét toàn bộ không gian ở độ sâu này.



Hình 2.7 – Không gian trạng thái với độ sâu giới hạn bằng 5 cho trò đố 8 ô

Giải pháp này đã đưa đến một thuật toán tìm kiếm khắc phục được nhiều nhược điểm của cả tìm kiếm sâu lẫn tìm kiếm rộng. Phương pháp *tìm kiếm sâu đào sâu nhiều lần (depth – first – interactive – deepening, Korf 1987)* thực hiện tìm kiếm sâu với độ sâu giới hạn bằng một. Nếu không tìm được đích, nó tiếp tục thực hiện tìm kiếm sâu với độ sâu giới hạn bằng hai, … Thuật toán cứ tiếp tục như thế bằng cách mỗi lần lặp thì tăng độ sâu giới hạn thêm một đơn vị. Trong mỗi lần lặp thuật toán áp dụng giải thuật tìm kiếm sâu hoàn chỉnh đến độ sâu giới hạn đó. Giữa hai lần lặp, không có một thông tin nào về không gian trạng thái được giữ lại.

Quá trình tìm kiếm sâu dần với độ sâu giới hạn là 2 trên đồ thị 2.6 như sau:

1. Open = [A]; Closed = []
2. Open = [B,C,D]; Closed = [A]
3. Open = [E,F,C,D];Closed = [B,A]
4. Open = [F,C,D,K,L];Closed = [E,B,A]
5. Open = [C,D,K,L,M];Closed = [F,E,B,A]
6. Open = [G,H,D,K,L,M];Closed = [C,F,E,B,A]
7. Open = [H,D,K,L,M,N];Closed = [G,C,F,E,B,A]
8. Open = [D,K,L,M,N,O,P];Closed = [H,G,C,F,E,B,A]
9. Open = [I,J,K,L,M,N,O,P];Closed = [D,H,G,C,F,E,B,A]
10. …

Thuật toán tìm kiếm sâu bằng cách đào sâu nhiều lần:

Trước hết ta định nghĩa độ sâu của đỉnh n như sau:

- d(n) là độ sâu của một đỉnh bất kỳ trong đồ thị biểu diễn không gian trạng thái,
- n_0 là đỉnh gốc của đồ thị (trạng thái khởi đầu),
- k: độ sâu giới hạn.

Khi đó: $d(n_0)=0$ và d(n)=d(m)+1 nếu n là con của m.

Thuật toán như sau

Procedure depth–more–search;

Begin % *khởi đầu*

Open:= [start]; Closed:= [ ];

DS = k % *k là độ sâu giới hạn*

While Open ≠ [ ] do % *còn các trạng thái chưa khảo sát*

Begin Lấy trạng thái X ngoài cùng bên trái khỏi Open;

If X là một đích then trả lời kết quả (thành công) % *tìm thấy đích*
else begin

Tính độ sâu của X là d(x);

Phát sinh tất cả các con của X là B(x);

Đưa X vào Closed;

Bỏ qua các con của X trong Open hoặc Closed; %*kiểm tra vòng lặp*

If (d(x)<DS) then đưa B(x) vào đầu bên **trái** của Open

Else

If (d(x)=DS) then đưa B(x) vào đầu bên **phải** của Open

Else begin

DS=DS+k;

Đưa B(x) vào đầu bên **trái** của Open;
end;
end;
End;
Trả lời kết quả (thất bại); % *không còn trạng thái nào*
End;

Vì thuật toán tìm kiếm hết không gian từng mức nên vẫn bảo đảm sẽ tìm được con đường ngắn nhất dẫn đến đích. Vì chỉ thực hiện tìm kiếm sâu trong mỗi lần lặp nên độ phức tạp của không gian tại mức n bất kỳ là B x n, trong đó B là số lượng trạng thái con trung bình của một nút.

Điều thú vị là mặc dù phương pháp tìm kiếm sâu đào sâu nhiều lần có vẻ kém hiệu quả về thời gian so với tìm kiếm sâu lẫn tìm kiếm rộng, nhưng thực tế độ phức tạp về thời gian của nó cùng bậc như của hai phương pháp trên: O (B^n).

2.3 TÌM KIẾM HEURISTIC

George Polya định nghĩa heuristic là "sự nghiên cứu về các phương pháp và các qui tắc trong việc khám phá và phát minh" (Polya 1945). Nghĩa này có thể xuất phát từ gốc Hy Lạp của động từ eurisco nghĩa là "tôi phát hiện". Khi Archimedes nhảy ra khỏi bồn tắm và chộp lấy chiếc mũ miện bằng vàng, ông ta đã la lên "Eureka!" có nghĩa "Tôi đã tìm thấy nó!". Trong tìm kiếm không gian trạng thái, heuristic là các luật dùng để chọn những nhánh nào có nhiều khả năng nhất dẫn đến một giải pháp chấp nhận được.

Các chương trình giải quyết những vấn đề trí tuệ nhân tạo sử dụng heuristic cơ bản theo hai dạng:

1. Vấn đề có thể không có giải pháp chính xác vì những điều không rõ ràng trong diễn đạt vấn đề hoặc trong các dữ liệu có sẵn. Chẩn đoán y khoa là một ví dụ. Tập hợp các triệu chứng cho trước có thể do nhiều nguyên nhân gây ra, bác sĩ có thể dùng heuristic để chọn kết quả chẩn đoán nào thích hợp nhất và đưa ra kế hoạch điều trị.
2. Vấn đề có thể có giải pháp chính xác, nhưng chi phí tính toán để tìm ra nó không cho phép. Trong nhiều vấn đề (như cờ vua chẳng hạn), không gian trạng thái phát triển rất nhanh và rất rộng vì số lượng các trạng thái có thể xảy ra tăng theo hàm mũ hoặc giai thừa cùng với độ sâu tìm kiếm. Trong những trường hợp này, các kỹ thuật tìm kiếm thô sơ như tìm kiếm sâu hay tìm kiếm rộng sẽ không tìm được giải pháp trong một giới hạn thời gian. Heuristic sẽ giảm bớt độ phức tạp bằng cách hướng việc tìm kiếm theo con đường có nhiều hứa hẹn nhất. Nhờ đã loại bỏ bớt các trạng thái không hứa hẹn và con cháu của chúng ra khỏi việc xem xét nên thuật toán heuristic có thể khắc phục việc bùng nổ trạng thái và tìm ra một giải pháp có thể chấp nhận được.

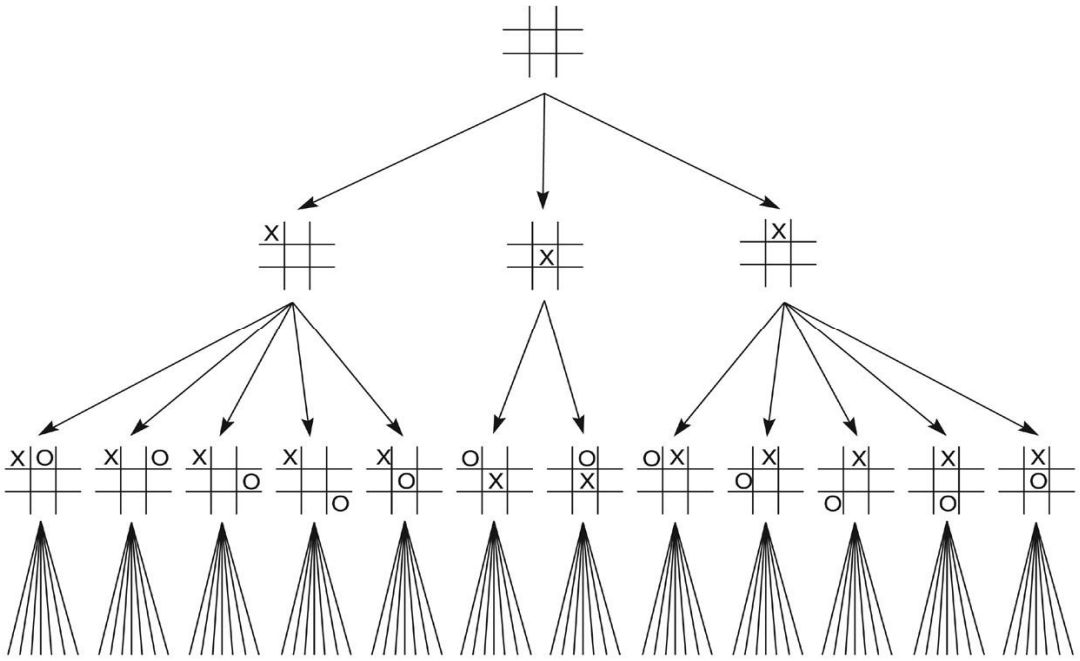
Giống như tất cả các luật khám phá và phát minh khác, heuristic có thể sai lầm. Heuristic chỉ là một phỏng đoán chứa các thông tin về bước tiếp theo sẽ được chọn dùng trong việc giải quyết một vấn đề. Nó thường dựa vào kinh nghiệm hoặc trực giác. Vì các heuristic sử dụng những thông tin hạn chế nên chúng ít khi có khả năng đoán trước chính xác cách hành xử của không gian trạng thái ở những giai đoạn xa hơn. Heuristic có thể dẫn đến một thuật toán tìm kiếm chỉ đạt được giải pháp gần tối ưu hoặc hoàn toàn không tìm được bất kỳ giải pháp nào. Đây là một hạn chế thuộc về bản chất tìm kiếm heuristic.

Các heuristic và việc thiết kế thuật toán để thực hiện tìm kiếm heuristic từ lâu đã là sự quan tâm chủ yếu của các công trình nghiên cứu trí tuệ nhân tạo. Chơi game và chứng minh định lý là hai ứng dụng lâu đời nhất, cả hai đều cần đến các heuristic để thu giảm bớt không gian giải pháp có thể. Không thể nào kiểm tra hết mọi suy luận có thể sinh ra trong lĩnh vực toán hoặc mọi nước đi có thể có trên bàn cờ vua, tìm kiếm heuristic thường là câu trả lời thực tế duy nhất. Gần đây việc tìm kiếm trong các hệ chuyên gia cũng xác nhận mức độ quan trọng của các heuristic như là một phần không thể thiếu trong quá trình giải quyết vấn đề.

Thuật toán heuristic gồm hai phần: Hàm đánh giá heuristic và thuật toán để sử dụng nó trong tìm kiếm không gian trạng thái.

Xét trò chơi Tic-tac-toe, để tìm kiếm đến hết không gian, số lượng trạng thái sẽ là rất lớn nhưng không phải là không thể vượt qua. Mỗi nước đi trong chín nước đầu tiên đều có tám khả năng đặt quân cờ kế tiếp và đến lượt mình mỗi nước đi này lại có bảy khả năng đặt quân cờ cho nước đi tiếp tục ... Một phân tích đơn giản cho biết số lượng các trạng thái cần được xem xét cho quá trình này là 9 x 8 x 7 x ... x 1 = 9!.

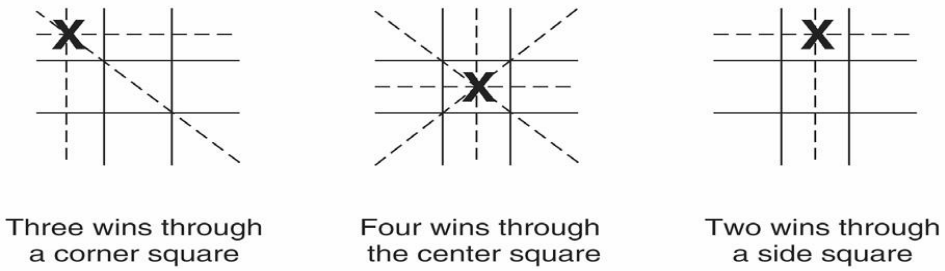
Áp dụng một nhận xét trực quan nhỏ dựa theo tính chất đối xứng của cấu hình bàn cờ có thể làm giảm nhỏ không gian tìm kiếm xuống một ít. Nhiều cấu hình của bài toán tương đương nhau trong các thao tác. Chẳng hạn, thực tế chỉ có ba nước đi cho quân cờ đầu tiên: ô cạnh, ô góc hoặc ô giữa. Các rút gọn đối xứng ở mức thứ hai của các trạng thái sẽ giảm tiếp số lượng đường đi có thể xảy ra trong không gian đó xuống đến tổng số 12 x 7! . Nó đã nhỏ hơn không gian ban đầu nhưng vẫn phát triển theo hàm giai thừa.



Hình 2.8 – Không gian trạng thái bài toán Tic-tac-toe thu giảm bởi tính đối xứng

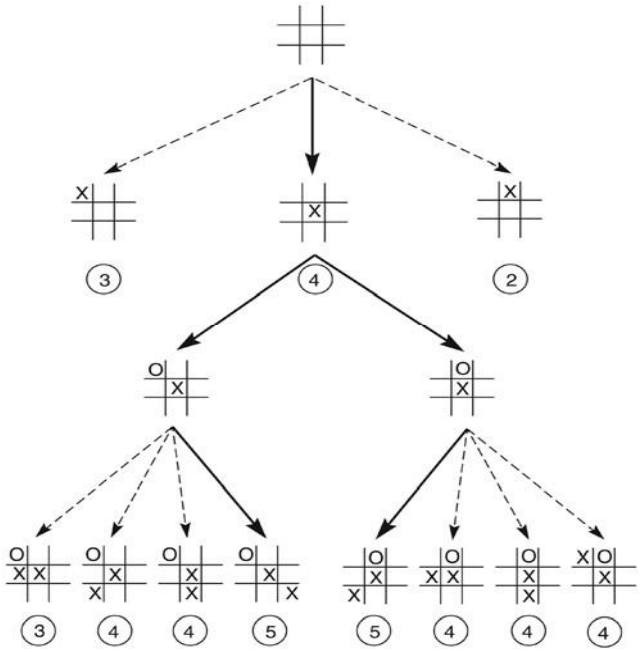
Tuy nhiên, một heuristic đơn giản có thể loại bỏ việc tìm kiếm hầu như toàn bộ: heuristic "nước đi chắc thắng nhất", nghĩa là chọn vị trí đặt quân cờ mà có nhiều đường chắc thắng nhất giao nhau. Trong trường hợp các trạng thái đều có số lượng bằng nhau, chọn trạng thái đầu tiên.

Sau đó thuật toán này sẽ chọn lựa và chuyển đến trạng thái có giá trị heuristic cao nhất (xem hình). Như trong hình vẽ, quân X sẽ chọn đặt vào vị trí trung tâm bàn cờ. Chú ý không chỉ các trạng thái tương đương khác bị loại bỏ mà tất cả con cháu của chúng cũng bị loại. Hai phần ba không gian trạng thái này đã bị loại ngay từ nước đi đầu tiên.



Hình 2.9 – Heuristic "nước đi chắc thắng nhất"

Sau nước đi đầu tiên, đối thủ có thể chọn một trong hai nước đi tương đương nhau. Dù chọn nước đi nào, heuristic đó cũng được áp dụng cho các bước tiếp theo. Khi quá trình tìm kiếm tiếp tục, từng bước đi sẽ đánh giá các con của một nút duy nhất mà không yêu cầu tìm kiếm hết không gian. Mặc dù khó tính chính xác số lượng trạng thái phải được kiểm tra theo cách này nhưng vẫn có thể tính phỏng đoán một giới hạn trên. Trong thực tế số lượng sẽ ít hơn 9! rất nhiều.



Hình 2.10 – Không gian trạng thái đã được thu giảm bởi heuristic

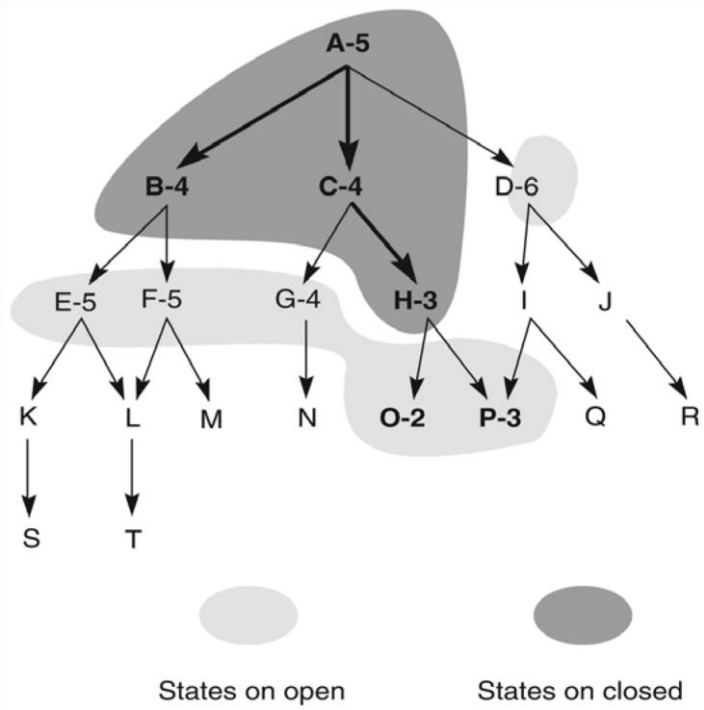
2.3.1 Tìm kiếm leo núi (Hill climbing – Pearl 1984)

Cách đơn giản nhất để thực hiện tìm kiếm heuristic là tìm kiếm "leo núi". Chiến lược leo núi phát triển trạng thái con tốt nhất sẽ được chọn cho bước tiếp theo, không lưu giữ lại bất kỳ thông tin nào về các nút anh em lẫn cha mẹ của nó. Quá trình tìm kiếm sẽ dừng lại khi tiếp cận trạng thái tốt hơn so với mọi trạng thái con của nó. Hình dung một người leo núi hăm hở nhưng mù quáng luôn luôn chọn leo lên đỉnh theo con đường dốc nhất có thể có cho đến khi không còn leo tiếp được nữa. Vì không ghi lại thông tin của quá trình đã xảy ra nên thuật toán này không thể phục hồi lại từ những thất bại trong chiến lược của nó.

Hạn chế chủ yếu của chiến lược leo núi là có xu hướng rơi vào "một cực đại cục bộ". Khi đến được một trạng thái tốt hơn so với mọi trạng thái con của nó, thuật toán dừng lại. Nếu trạng thái này không phải là đích mà chỉ là một điểm cực đại cục bộ, thuật toán sẽ thất bại trong việc tìm lời giải. Như vậy hiệu quả hoạt động chỉ có thể được cải thiện trong một phạm vi giới hạn nào đó, nhưng trong toàn bộ không gian có thể không bao giờ đạt được sự tối ưu tổng thể.

2.3.2 Tìm kiếm tốt nhất đầu tiên (Best – First – Search)

Xét đồ thị không gian tìm kiếm như hình dưới đây (con số cạnh mỗi nút cho biết giá trị ước lượng độ tốt của nút đó trong không gian, giá trị thấp nhất là tốt nhất). Giả sử nút đích cần tìm kiếm là P.



Hình 2.11 – Đồ thị cho giải thuật tìm kiếm tốt nhất đầu tiên

Giống như các thuật toán tìm kiếm sâu và rộng, tìm kiếm tốt nhất cũng dùng các danh sách để lưu giữ trạng thái: danh sách Open chứa các nút được triển khai trong quá trình tìm kiếm và danh sách Closed chứa các nút đã xét. Một bước mới được bổ sung vào thuật toán là sắp xếp các trạng thái trong danh sách Open phù hợp với giá trị heuristic ước lượng "độ tốt" của chúng so với đích. Như vậy mỗi bước lặp của vòng lặp sẽ xem xét trạng thái "có hứa hẹn nhất" trong danh sách Open và loại bỏ trạng thái này ra khỏi Open. Nếu gặp trạng thái đích, thuật toán này sẽ cung cấp con đường lời giải đã dẫn đến đích đó. Nếu ngược lại, phần tử đầu tiên của Open không phải là đích, thuật toán sẽ áp dụng các luật phù hợp để phát sinh con cháu. Trường hợp một trạng thái con nào đó đã có sẵn trong Open hoặc Closed, thuật toán cũng sẽ kiểm tra để chắc chắn rằng sẽ chọn được nút cung cấp con đường lời giải ngắn hơn. Các trạng thái lặp hai lần sẽ không được giữ lại. Nhờ cập nhật kịp thời nguồn gốc của các nút trong Open và Closed nên thuật toán này có nhiều khả năng tìm được đường đi ngắn nhất dẫn đến đích. Khi Open được duy trì dưới dạng một danh sách có sắp xếp, nó thường được tổ chức như là một *hàng ưu tiên* (Priority queue).

Quá trình tìm kiếm cho đồ thị hình 2.11 ta có kết quả sau:

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!

Dưới đây trình bày các bước của giải thuật:

Procedure Best–First–Search;

Begin % *khởi đầu*

Open:= [start]; Closed:= [ ];

While Open ≠ [ ] do % *còn các trạng thái chưa khảo sát*

Begin

Lấy trạng thái đầu tiên X ra khỏi Open;

Đưa X vào Closed;

If X là một đích then trả lời kết quả đường đi % *tìm thấy đích*
else begin

Phát sinh tất cả các con của X là B(x);

Đưa X vào Closed;

For mỗi nút con B(X) của X do

If B(X) không thuộc Open hay Closed then

Begin

Tính heuristic cho B(X);

Đưa B(X) vào Open;

End

Else if B(X) đã có trong Open then

Begin

If B(X) mới có đường đi tốt hơn then

Thay B(X) cũ bằng B(X) mới

End

Else if B(X) đã có trong Closed then

Begin

Xóa B(X) trong Closed;

Đưa B(X) vào Open

End

End

End;

Trả lời kết quả (thất bại); % *không còn trạng thái nào*

End;

Thuật toán tìm kiếm BFS như trên còn được gọi là giải thuật A

2.3.4 Tìm kiếm đường đi trên đồ thị tổng quát – giải thuật A*

A* là phiên bản mở rộng của giải thuật A được áp dụng cho trường hợp đồ thị. Thuật giải A* mở rộng A bằng cách bổ sung cách giải quyết khi xét một nút mà nút này đã có trong Open hoặc Closed và khi xét đến một trạng thái X bên cạnh lưu trữ các giá trị f, g, h để phản ánh độ tốt của một trạng thái giải thuật còn lưu trữ thêm 2 thông số sau:

1. Trạng thái trước của trạng thái X, ký hiệu prev(X): cho biết trạng thái dẫn đến trạng thái X. Trong trường hợp các nhiều trạng thái dẫn đến X thì prev(X) là trạng thái dẫn đến X với chi phí thấp nhất tính từ trạng thái bắt đầu, nghĩa là:

$$g(X)=g(prev(X)) + cost(prev(X),X) \text{ là nhỏ nhất.}$$

2. Danh sách các trạng thái kế tiếp của X: lưu trữ các trạng thái kế tiếp X_k của X sao cho chi phí từ trạng thái bắt đầu đến X_k thông qua X là thấp nhất, nói cách khác, $prev(X_k)=X$.

Khi đó, thuật giải A* được mô tả như sau:

```
Procedure A*
Begin
    Open=[start]; Closed=[];
    g[start]=0; h[start]; f[start]=h[start];
    While Open ≠ [ ] do                    % còn các trạng thái chưa khảo sát
    Begin
        Chọn X trong Open sao cho f(X) nhỏ nhất;
        Lấy X ra khỏi Open và đưa X vào Closed;
        If X là đích then trả lời kết quả đường đi
        Else
          Begin
            Tạo ra danh sách tất cả các con của X, gọi mỗi trạng thái con này là Xk;
            For mỗi con Xk của X do
            Begin
                Tính g(Xk)=g(X)+cost(X,Xk);
                If tồn tại Xk' trong Open trùng Xk và g(Xk)<g(Xk') then
                Begin
                    Đặt g(Xk')=g(Xk);
                    Tính lại f(Xk')= g(Xk')+ h(Xk');
                    Cập nhật prev(Xk')=X;
                End;
                If tồn tại Xk' trong Closed trùng Xk và g(Xk)<g(Xk') then
```

Begin

Đặt $g(X_{k'})=g(X_k)$;

Tính lại $f(X_{k'})= g(X_{k'})+ h(X_{k'})$;

Cập nhật $prev(X_{k'})=X$;

Lan truyền sự thay đổi đến các trạng thái khác trong Open và Closed;

End;

If X_k chưa có trong Open và Closed then

Begin

Thêm X_k vào Open;

Tính $f(X_k)= g(X_k)+ h(X_k)$;

Đặt $prev(X_k)=X$;

End;

End;

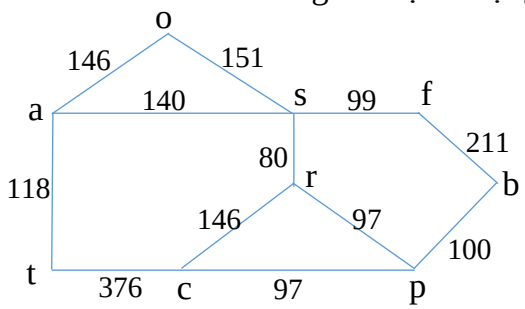
End;

End;

End;

Đường đi ngắn nhất được tìm thấy khi tìm thấy đích và xây dựng đường đi này bằng cách lần ngược theo danh sách prev.

Ví dụ: tìm đường đi ngắn nhất từ a đến b trong đồ thị có trọng số như sau:



Hình 2.12 Đồ thị cho giải thuật A*

Giá trị ước lượng Heuristic từ trạng thái hiện hành đến trạng thái b như sau:

Trạng thái	a	b	c	f	o	p	r	s	t
Giá trị h	366	0	160	178	380	98	193	253	329

Áp dụng thuật toán A*, các bước tìm kiếm lời giải như sau:

B1. Khởi tạo:

start=a; Open=[a]; Closed=[];

g[a]=0; h[a]=366; f[a]=366;

B2. Chọn X=a; Open=[]; Closed=[a];

Con a: o, s, t;

g(o)=g(a)+cost(a,o)=0+146=146;

Tương tự: g(s)=140; g(t)=118;

Do o, s, t không trong Open và Closed nên: Open=[o, s, t];

f(o)=g(o)+h(o)=146+380=526;

Tương tự: f(s)=**393**; f(t)=447;

prev(o)=prev(s)=prev(t)=a;

B2. Chọn X=s; Open=[o, t]; Closed=[a, s];

Con s: a, o, f, r;

g_m(a)=g(s)+cost(s,a)=140+140=280;

Tương tự: g_m(o)=291; g(f)=239; g(r)=220;

a thuộc Closed, g_m(a)>g(a) nên bỏ qua a mới;

o thuộc Open, g_m(o)>g(o) nên bỏ qua o mới;

f, r không thuộc Open và Closed nên: Open=[o, t, f, r];

f(f)=g(f)+h(f)=417;

Tương tự f(r)=**413**;

prev(f)=prev(r)=o;

B3. Chọn X=r; Open=[o, t, f]; Closed=[a, s, r];

Con r: c, s, p;

g_m(s)=g(r)+cost(r,s)=220+80=300;

Tương tự: g(c)=366; g(p)=317;

s thuộc Closed, g_m(s)>g(s) nên bỏ qua s mới;

c, p không thuộc Open và Closed nên: Open=[o, t, f, c, p];

f(c)=g(c)+h(c)=366+160=526;

Tương tự: f(p)=**415**;

prev(c)=prev(p)=r;

B4. Chọn X=p; Open=[o, t, f, c]; Closed=[a, s, r, p]

Con p: c, r, b;

g_m(c)=g(p)+cost(p,c)=317+97=414;

Tương tự: g_m(r)=414; g(b)=417;

c thuộc Closed, g_m(c)>g(c) nên bỏ qua c mới;

r thuộc Open, g_m(r)>g(r) nên bỏ qua r mới;

Do b không thuộc Open và Closed nên: Open=[o, t, f, c, b];

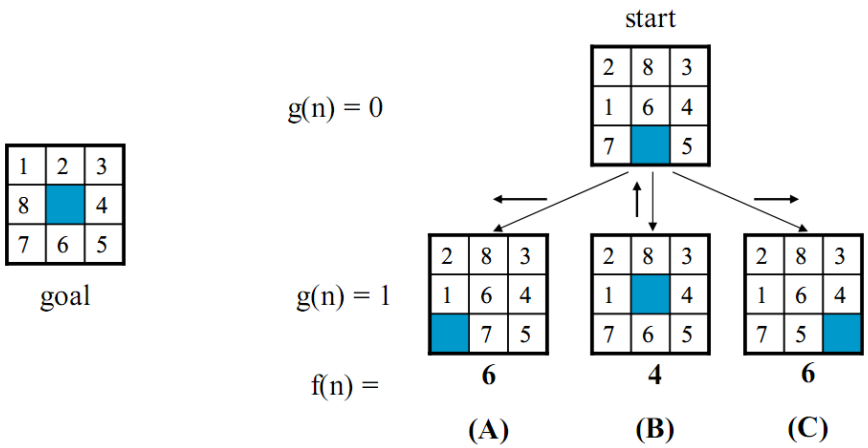
f(b)=g(b)+h(b)=417+0=**417**;

prev(b)=p;

B5. Chọn X=b. Vì b là đích nên giải thuật kết thúc, vậy đường đi ngắn nhất từ a đến b là: a – s – r – p – b

2.3.5 Cài đặt hàm đánh giá heuristic (heuristic evaluation function)

Bây giờ ta đánh giá hiệu quả của vài heuristic khác nhau được dùng để giải trò đố 8 ô. Hình dưới đây trình bày trạng thái xuất phát và trạng thái đích của trò chơi cùng với ba trạng thái đầu tiên trong quá trình tìm kiếm.



Hình 2.13 – Trạng thái bắt đầu và kết thúc trong trò đố 8 ô

Heuristic đơn giản nhất sẽ đếm *số ô sai khác so với trạng thái đích* trong từng trạng thái. Trạng thái có số ô sai khác ít nhất sẽ gần đích hơn và là trạng thái tốt nhất để kiểm tra kế tiếp.

Tuy nhiên heuristic này không sử dụng hết các thông tin trong một cấu hình bàn cờ vì nó không đưa vào khoảng cách mà các ô sai khác. Một heuristic "tốt hơn" là sẽ cộng tất cả các khoảng cách đó lại thành *tổng số ô mà một ô phải di chuyển về vị trí đúng* của nó trong trạng thái đích (khoảng cách Manhattan).

Cả hai heuristic này đều có hạn chế là không thể biết rõ những khó khăn khi đổi chỗ hai ô. Đó là trường hợp hai ô nằm cạnh nhau và vị trí đúng của chúng là phải đổi chỗ cho nhau, ta phải mất nhiều (chứ không phải hai) nước đi mới đặt chúng lại được đúng vị trí. Một heuristic muốn tính toán điều này phải nhân ít nhất là gấp đôi khoảng cách đối với mỗi trường hợp có hai ô đổi chỗ trực tiếp.

Heuristic "khoảng cách Manhattan" cho chúng ta một dự đoán có vẻ chính xác hơn so với heuristic số ô sai khác so với trạng thái đích. Mục đích của chúng ta là dùng những thông tin hạn chế có sẵn trong một mô tả trạng thái để đưa ra những chọn lựa thông minh. Việc thiết kế các heuristic tốt là một vấn đề mang tính kinh nghiệm, óc phán đoán và trực giác, nhưng giá trị cuối cùng của một heuristic phải được đo bằng hiệu quả thực sự của nó trong từng tình huống bài toán.

Vì heuristic có thể sai lầm nên có khả năng thuật toán tìm kiếm sẽ dẫn đến một con đường không đưa đến đích. Vấn đề này đã xuất hiện trong tìm kiếm sâu, ở đó một giới hạn độ sâu đã được sử dụng để phát hiện những con đường thất bại. Ý tưởng này cũng có thể áp dụng cho tìm kiếm heuristic. Nếu hai trạng thái có giá trị heuristic bằng nhau thì nên kiểm tra trạng thái nào gần trạng thái gốc của đồ thị hơn. Trạng thái gần gốc hơn sẽ có nhiều khả năng là con đường ngắn nhất dẫn đến đích. Khoảng cách từ trạng thái xuất phát có thể đo được bằng cách duy trì một số đếm chiều sâu cho từng trạng thái đếm. Số đếm này bằng 0 đối với trạng thái xuất phát và tăng lên một đơn vị sau mỗi mức tìm kiếm. Nó ghi lại độ dời thực tế phải thực hiện để đi từ trạng thái xuất phát đến từng trạng thái. Số đếm này có thể cộng thêm vào giá trị heuristic của từng trạng thái để hướng việc tìm kiếm theo khuynh hướng chọn những trạng thái gần trạng thái xuất phát hơn trong đồ thị.

Do đó hàm đánh giá f sẽ bao gồm tổng của hai phần: f(n) = g(n) + h(n)

Trong đó g(n) đo chiều dài thực từ trạng thái n bất kỳ về trạng thái xuất phát và h(n) là ước lượng heuristic cho khoảng cách từ trạng thái n đến trạng thái đích.

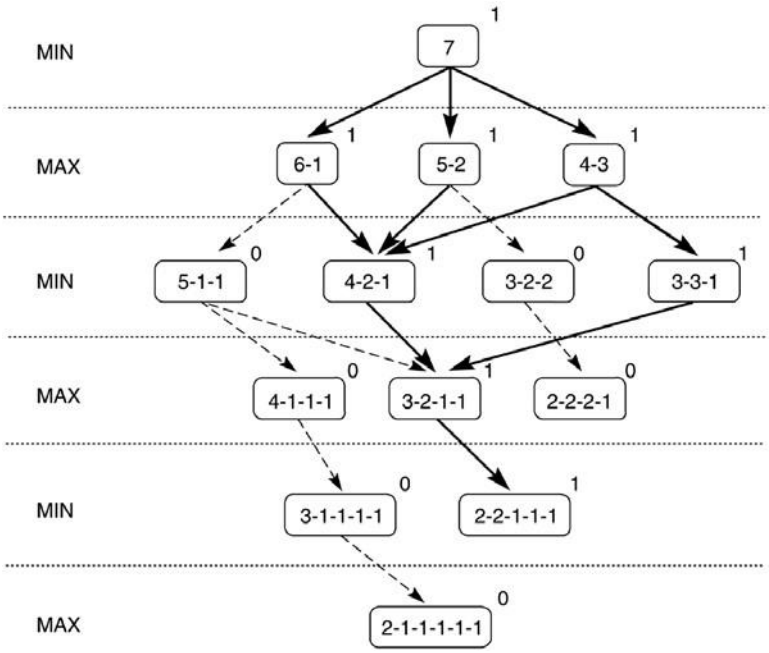
Chẳng hạn, trong bài toán trò đố 8 ô trên, chúng ta có thể gọi h(n) là số ô cần phải dời chỗ. Khi hàm đánh giá này được áp dụng cho từng trạng thái con (A), (B), (C) trong hình 2.13, các giá trị cho hàm f lần lượt là 6, 4 và 6.

Một heuristic dùng hàm đánh giá f(n) như trên kết hợp với thuật toán tìm kiếm tốt nhất đầu tiên best-first-search, được gọi là *thuật toán A* (algorithm A)

2.3.6 Sử dụng Heuristic trong các trò chơi

2.3.6.1 Thủ tục minimax

Xét các trò chơi hai đối thủ đối kháng, chẳng hạn trò chơi *nim*. Để chơi nim, một số token (vật biểu hiện như đồng xu, lá bài, mảnh gỗ, ...) được đặt trên bàn giữa hai đối thủ. Ở mỗi nước đi, người chơi phải chia đống token thành hai đống nhỏ có số lượng khác nhau. Ứng với một số token vừa phải, không gian trạng thái này có thể triển khai đến cùng. Hình sau biểu diễn không gian trạng thái của trò chơi có 7 token.



Hình 2.14 – Không gian trạng thái của trò chơi nim

Khi chơi các trò chơi có thể triển khai hết không gian trạng thái, khó khăn chủ yếu là phải tính toán phản ứng của đối thủ. Một cách xử lý đơn giản nhất là giả sử đối thủ của bạn cũng sử dụng kiến thức về không gian trạng thái giống như bạn và áp dụng kiến thức đó kiên định để thắng cuộc. Mặc dù giả thiết này có những hạn chế của nó nhưng nó cũng cho chúng ta một cơ sở hợp lý để dự đoán hành vi của đối thủ. Minimax sẽ tìm kiếm không gian của trò chơi này theo giả thiết đó.

Hai đối thủ trong một trò chơi được gọi là MIN và MAX. MAX đại diện cho đối thủ quyết giành thắng lợi hay cố gắng tối đa hóa ưu thế của mình. Ngược lại MIN là đối thủ cố gắng tối thiểu hóa điểm số của MAX. Ta giả thiết MIN cũng dùng cùng những thông tin như MAX.

Khi áp dụng thủ tục Minimax, chúng ta đánh dấu luân phiên từng mức trong không gian tìm kiếm phù hợp với đối thủ có nước đi ở mức đó. Trong ví dụ trên, MIN được quyền đi trước, từng nút lá được gán giá trị 1 hay 0 tùy theo kết quả đó là thắng cuộc đối với MAX hay MIN. Minimax sẽ truyền các giá trị này lên cao dần trên đồ thị qua các nút cha mẹ kế tiếp nhau theo luật sau:

- Nếu trạng thái cha mẹ là nút MAX, gán cho nó giá trị tối đa của các con cháu của nó.
- Nếu trạng thái cha mẹ là nút MIN, gán cho nó giá trị tối thiểu của các con cháu của nó.

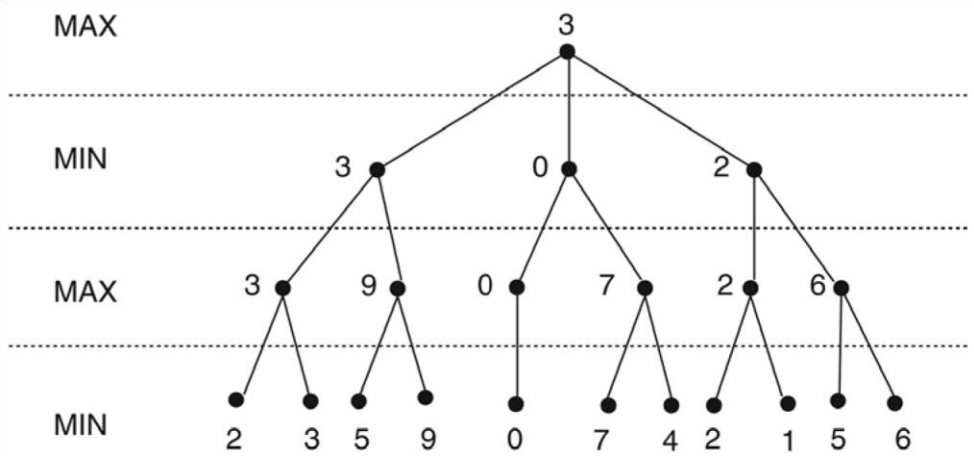
Giá trị được gán cho từng trạng thái bằng cách đó sẽ chỉ rõ giá trị của trạng thái tốt nhất mà đối thủ này có thể hy vọng đạt được. Các giá trị này sẽ được dùng để lựa chọn

các nước đi có thể có. Kết quả của việc áp dụng Minimax vào đồ thị không gian trạng thái đối với trò chơi Nim được thể hiện như hình trên. Vì tất cả các nước đi đầu tiên có thể xảy ra cho MIN sẽ dẫn đến các nút có giá trị 1 nên đối thủ MAX luôn có thể bắt trò chơi giành thắng lợi cho mình bất kể nước đi đầu tiên của MIN là như thế nào (đường đi thắng lợi của MAX được cho theo mũi tên đậm).

2.3.6.2 Áp dụng minimax đến độ sâu lớp cố định

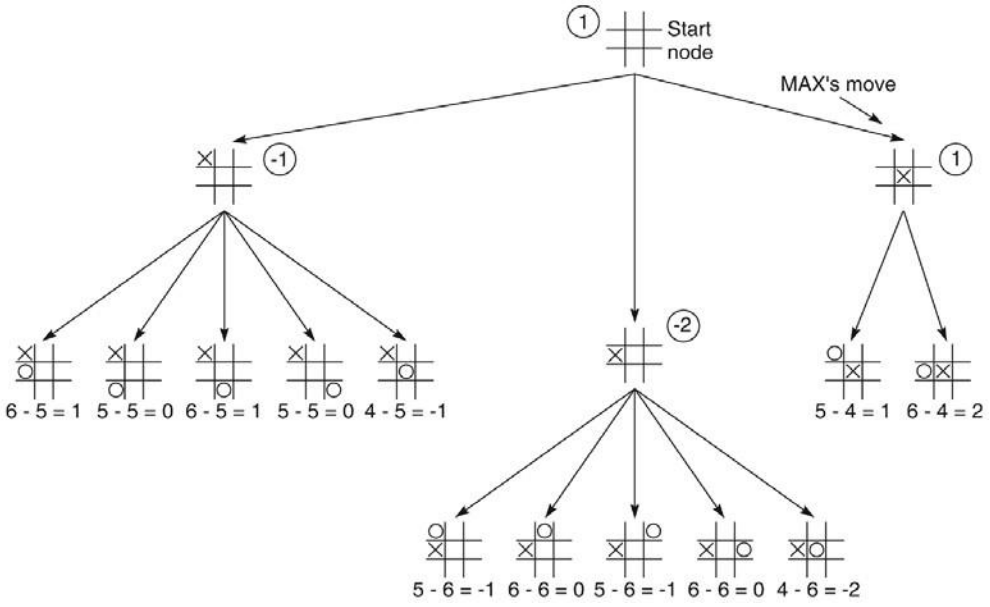
Khi áp dụng Minimax cho các trò chơi phức tạp, hiếm khi có khả năng mở rộng đồ thị không gian trạng thái đến các nút lá. Thay vào đó không gian trạng thái này chỉ có thể được triển khai đến một số mức xác định phụ thuộc tiềm năng về thời gian và bộ nhớ chẳng hạn. Chiến lược này được gọi là *tính trước n nước đi* (n –move lookahead). Vì giá trị các nút trong đồ thị con này không phải là trạng thái kết thúc của trò chơi nên chúng không phản ánh giá trị thắng cuộc hay thua cuộc. Chúng chỉ có thể được gán một giá trị phù hợp với một hàm đánh giá heuristic nào đó. Giá trị được truyền ngược về nút gốc không cung cấp thông tin thắng cuộc hay thua cuộc mà chỉ là giá trị heuristic của trạng thái tốt nhất có thể tiếp cận sau n nước đi kể từ nút xuất phát. Việc tính trước này sẽ làm tăng hiệu quả của heuristic vì nó được áp dụng vào một phạm vi lớn hơn trong không gian trạng thái. Minimax sẽ hợp nhất tất cả các giá trị của các nút con cháu của một trạng thái thành một giá trị duy nhất cho trạng thái đó.

Trong các đồ thị trò chơi được tìm kiếm bằng mức hay lớp, MAX và MIN luân phiên nhau chọn các nước đi. Mỗi nước đi của một đối thủ sẽ xác định một lớp mới trên đồ thị. Các chương trình trò chơi nói chung đều dự tính trước một *độ sâu lớp cố định* (thường được xác định bằng các giới hạn về không gian hoặc thời gian của máy tính). Các trạng thái trên mức đó được đánh giá theo các heuristic và các giá trị này sẽ được truyền ngược lên bằng thủ tục Minimax, sau đó thuật toán tìm kiếm sẽ dùng các giá trị vừa nhận được để chọn lựa một nước trong số các nước đi kế tiếp. Bằng cách tối đa hóa cho các cha mẹ MAX và tối thiểu hóa cho các cha mẹ MIN, những giá trị này đi lùi theo đồ thị đến con của trạng thái hiện hành. Sau đó trạng thái hiện hành dùng chúng để tiến hành lựa chọn trong các con của nó. Hình sau trình bày quá trình Minimax trên một không gian trạng thái giả thuyết tính trước bốn lớp.



Hình 2.15 – Minimax đối với một không gian trạng thái giả định

Hình 2.16 giới thiệu một ứng dụng của Minimax độ sâu lớp cố định vào trò chơi Tic-tac-toe.



Hình 2.16 – Minimax hai lớp được áp dụng vào nước đi mở đầu trò chơi Tic-tac-toe

Ở đây sử dụng một heuristic hơi phức tạp hơn, nó cố đo mức độ tranh chấp trong trò chơi. Heuristic chọn một trạng thái cần đo, tính tất cả các đường thắng mở ra cho MAX, rồi trừ đi tổng số các đường thắng mở ra cho MIN. Giải thuật tìm kiếm sẽ cố gắng tối đa hóa sự chênh lệch (hiệu số) đó. Nếu có một trạng thái bắt buộc thắng cuộc cho MAX, nó sẽ được đánh giá là $+\infty$, còn với trạng thái bắt buộc thắng cuộc cho MIN thì được đánh giá là $-\infty$. Hình 2.18 trình bày heuristic này được áp dụng cho hai mức bắt đầu không gian trạng thái.

2.3.6.3 Thủ tục cắt tỉa alpha – beta (α-β prunning)

Minimax yêu cầu phải có sự phân tích qua hai bước đối với không gian tìm kiếm: Bước đầu truyền xuống đến độ sâu của lớp áp dụng heuristic và bước sau để truyền ngược các giá trị trên cây. Minimax lần theo tất cả các nhánh trong không gian bao gồm cả những nhánh mà một thuật toán thông minh hơn có thể bỏ qua hay tỉa bớt. Các nhà nghiên cứu trong lĩnh vực chơi game đã xây dựng một kỹ thuật tìm kiếm gọi là *cắt tỉa alpha –beta* nhằm nâng cao hiệu quả tìm kiếm trong các bài toán trò chơi hai đối thủ.

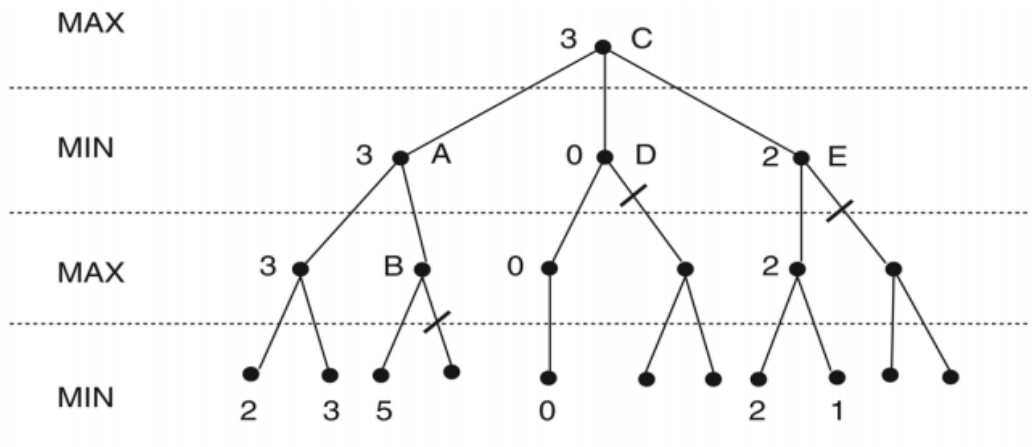
Ý tưởng của tìm kiếm alpha – beta rất đơn giản: Thay vì nếu như tìm kiếm toàn bộ không gian đến một độ sâu lớp cố định, tìm kiếm alpha – beta thực hiện theo kiểu tìm kiếm sâu. Có hai giá trị, gọi là alpha và beta được tạo ra trong quá trình tìm kiếm. Giá trị alpha liên quan với các nút MAX và có khuynh hướng không bao giờ giảm. Ngược lại giá trị beta liên quan đến các nút MIN và có khuynh hướng không bao giờ tăng. Giả sử có giá trị alpha của một nút MAX là 6, MAX không cần phải xem xét giá trị truyền ngược nào nhỏ hơn hoặc bằng 6 có liên quan với một nút MIN nào đó bên dưới. Alpha là giá trị thấp nhất mà MAX có thể nhận được sau khi cho rằng MIN cũng sẽ nhận giá trị tốt nhất của nó. Tương tự nếu MIN có giá trị beta là 6 nó cũng không cần xem xét các nút nằm dưới nó có giá trị lớn hơn hoặc bằng 6.

Để bắt đầu thuật toán tìm kiếm alpha – beta, ta đi xuống hết độ sâu lớp theo kiểu tìm kiếm sâu, đồng thời áp dụng đánh giá heuristic cho một trạng thái và tất cả các trạng thái anh em của nó. Giả thuyết tất cả đều là nút MIN. Giá trị tối đa của các nút MIN này sẽ được truyền ngược lên cho nút cha mẹ (là một nút MAX). Sau đó giá trị này được gán cho ông bà của các nút MIN như là một giá trị beta kết thúc tốt nhất. Tiếp theo thuật toán này sẽ đi xuống các nút cháu khác và kết thúc việc tìm kiếm đối với nút cha mẹ của chúng nếu gặp bất kỳ một giá trị nào lớn hơn hoặc bằng giá trị beta này. Quá trình này gọi là *cắt tỉa beta* (β cut). Cách làm tương tự cũng được thực hiện cho việc *cắt tỉa alpha* (α cut) đối với các nút cháu của một nút MAX.

Hai luật cắt tỉa dựa trên các giá trị alpha và beta là:

1. Quá trình tìm kiếm có thể kết thúc bên dưới một nút MIN nào có giá trị beta nhỏ hơn hoặc bằng giá trị alpha của một nút cha MAX bất kỳ của nó.
2. Quá trình tìm kiếm có thể kết thúc bên dưới một nút MAX nào có giá trị alpha lớn hơn hoặc bằng giá trị beta của một nút cha MIN bất kỳ của nó.

Việc cắt tỉa alpha – beta như vậy thể hiện quan hệ giữa các nút ở lớp n và các nút ở lớp n+2 và do quan hệ đó toàn bộ các cây con bắt nguồn ở lớp n+1 đều có thể loại khỏi việc xem xét. Chú ý rằng giá trị truyền ngược thu được hoàn toàn giống như kết quả Minimax, đồng thời tiết kiệm được các bước tìm kiếm một cách đáng kể.



Hình 2.17 – Thực hiện giải thuật cắt tỉa alpha – beta

Các bước định trị nút C như sau:

- A có β = 3 (Trị nút A sẽ không lớn hơn 3)
- B bị cắt tỉa β, vì 5>3
- C có α = 3 (Trị nút A sẽ không nhỏ hơn 3)
- D bị cắt tỉa α, vì 0<3
- E bị cắt tỉa α, vì 2<3
- Trị nút C là 3

2.4 Tổng kết chương

Chương này đã giới thiệu các cơ sở lý thuyết trong tìm kiếm không gian trạng thái, sử dụng lý thuyết đồ thị để phân tích cấu trúc và mức độ phức tạp của các chiến lược giải quyết vấn đề bài toán. Các cách thức có thể sử dụng để mô hình hóa việc giải quyết vấn đề dưới dạng một tìm kiếm trên đồ thị trạng thái của bài toán đó cũng đã được nêu ra. Đồng thời cũng so sánh giữa hai cách suy luận hướng dữ liệu và hướng mục tiêu, giữa tìm kiếm sâu và tìm kiếm rộng.

Phần cuối chương, các phương pháp tìm kiếm heuristic được giới thiệu thông qua các trò chơi đơn giản như trò đố 8 ô, Tic-tac-toe, … và cũng đã được phát triển đến các không gian bài toán phức tạp hơn. Chương này cũng đã trình bày việc áp dụng heuristic cho các trò chơi đối kháng có hai người chơi, dùng cách rút gọn tối thiểu trên độ sâu lớp và cắt tỉa alpha - beta để thực hiện việc tính trước các nước đi và dự đoán hành vi của đối thủ. Việc áp dụng các heuristic này đã làm cho không gian bài toán trở nên ngắn gọn hơn, thời gian tìm kiếm một lời giải có thể chấp nhận được là tối thiểu và quá trình tìm kiếm vì thế cũng trở nên đơn giản hơn khá nhiều.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

1. Xác định phương pháp tìm kiếm hướng dữ liệu hay tìm kiếm hướng mục tiêu là thích hợp hơn trong việc giải quyết các vấn đề dưới đây. Giải thích sự lựa chọn của bạn:
   a. Chẩn đoán các trục trặc về thiết bị trong một xe ô tô.
   b. Bạn gặp một người cho biết là anh em họ xa với bạn, có cùng một ông tổ tên là John. Bạn cần kiểm tra lại thông tin này.
   c. Một người khác cũng cho biết là anh em họ xa của bạn. Anh ta không biết ông tổ tên gì nhưng biết rằng không quá tám đời. Bạn hãy tìm ông tổ này hoặc xác định không có ông tổ đó.
   d. Một chứng minh định lý trong hình học phẳng.
   e. Một chương trình dùng để kiểm tra và diễn giải kết quả đọc của máy dò đường biển bằng sóng phản âm, chẳng hạn dùng thông báo cho một tàu ngầm lớn biết sắp có một tàu ngầm nhỏ, một con cá voi hay một đàn cá ở cự ly nào đó.
   f. Một hệ chuyên gia giúp cho con người phân loại cây trồng theo đặc tính, chủng loại, …
2. Giả sử cần viết chương trình tìm ra cách thức xoay một khối rubic 3x3x3 (được ghép bởi 27 nút) về đúng 6 mặt màu. Biết rằng bộ nhớ trong của máy tính bị hạn chế, bạn sẽ chọn giải thuật duyệt đồ thị nào trong hai giải thuật tìm kiếm sâu và tìm kiếm rộng để giải quyết vấn đề? Giải thích ngắn gọn sự lựa chọn của bạn.
3. Xét bài toán trò đố 8 ô như sau:

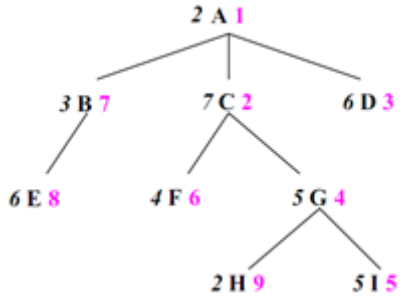
   Start

1	2	3
6	7	
8	5	4

   Goal

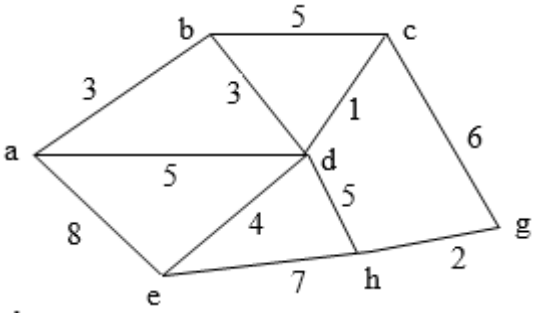
1	2	3
8		4
7	6	5

   Dùng các hàm lượng giá heuristic sau, hãy triển khai không gian trạng thái của bài toán theo giải thuật leo núi đến mức 5:
   a. h1 = số lượng các vị trí sai khác so với trạng thái goal.
   b. h2 = tổng số độ dời ngắn nhất của các ô về vị trí đúng (khoảng cách Manhattan)
4. Trong cây tìm kiếm dưới đây, mỗi nút có 2 giá trị đi kèm: giá trị bên trái của nút (in nghiêng) thể hiện giá trị heuristic (phản ánh độ tốt) của nút, giá trị bên phải nút thể hiện thứ tự nút được duyệt qua. Hãy viết danh sách thứ tự các nút được duyệt khi sử dụng phương pháp tìm kiếm rộng, tìm kiếm sâu, tìm kiếm tốt nhất đầu tiên, tìm kiếm leo núi; so sánh và cho biết ta đã dùng giải thuật tìm kiếm nào trên cây?



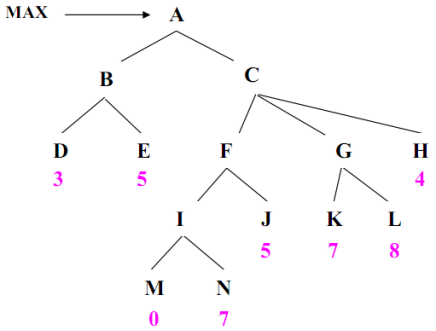
5. Cho đồ thị như hình bên và bảng ước lượng Heuristic như sau:

Điểm	a	b	c	d	e	g	h
h_0	14	10	6	7	10	0	3



Hãy xác định đường đi ngắn nhất từ a đến g bằng giải thuật A*

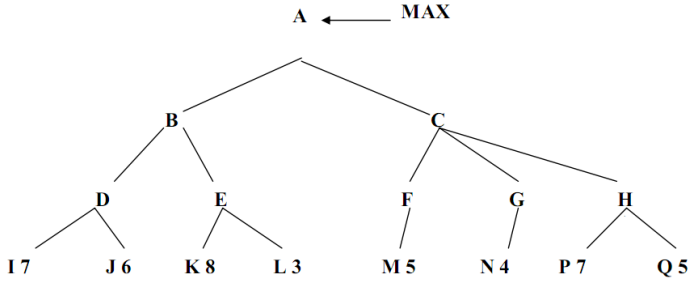
6. Thực hiện giải thuật Minimax trên cây sau đây:



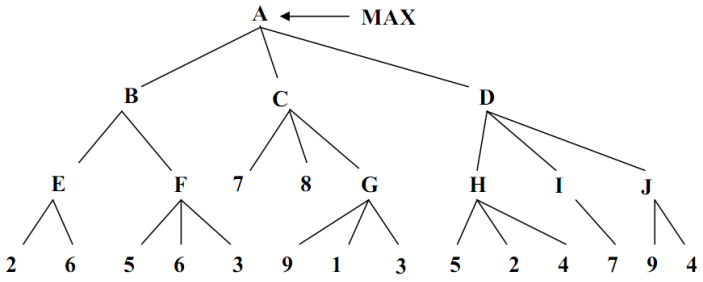
Sẽ có gì khác biệt nếu như ta dùng giải thuật cắt tỉa alpha – beta để định trị nút gốc cho cây?

7. Hãy áp dụng thủ tục cắt tỉa alpha – beta cho các cây sau đây:

a.



b.



Chương 3 CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

3.1 TRI THỨC VÀ DỮ LIỆU

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 với con người.

Thông tin là tất cả những gì mà con người có thể tiếp nhận được một cách trực tiếp thông qua các giác quan của mình hoặc gián tiếp thông qua các phương tiện kỹ thuật như tivi, radio, cassette … thông tin đối với con người luôn luôn có ý nghĩa.

Nói cách khác, thông tin là quan hệ giữa các dữ liệu; các dữ liệu tổ chức, sắp xếp theo một thứ tự nào đó hoặc tập hợp lại theo một thứ tự nào đó sẽ tạo ra thông tin. Nhưng nếu quan hệ này được chỉ ra một cách rõ ràng thì được gọi là tri thức.

Ví dụ: Trong toán học, bản thân các con số 1, 1, 3, 2, 13, 5, 8, 34, 21 là các dữ liệu. Tuy nhiên, khi đặt chúng lại với nhau theo trật tự như bên dưới thì giữa chúng bắt đầu có mối quan hệ với nhau:

Dữ liệu: 1, 1, 2, 3, 5, 8, 13, 21, 34

Mối quan hệ này có thể biểu diễn bằng công thức sau: $U_n=U_{n-1}+U_{n-2}$. Công thức này là một tri thức.

3.2 CÁC LOẠI TRI THỨC

3.2.1 Tri thức sự kiện

Tri thức sự kiện là một khẳng định về một sự kiện, hiện tượng hay một khái niệm nào đó trong một hoàn cảnh không gian hoặc thời gian nhất định.

Ví dụ: Khẳng định về hiện tượng: “Mặt trời lặn ở hướng Tây”. Khái niệm: “Tam giác đều là tam giác có ba góc bằng nhau”.

3.2.2 Tri thức mô tả

Cho biết một đối tượng, sự kiện, vấn đề, khái niệm … được thấy, cảm nhận, cấu tạo như thế nào.

Ví dụ: Mô tả “Cái bàn có 4 chân”, “Con người có 2 tay 2 chân”, …

3.2.3 Tri thức thủ tục

Tri thức thủ tục là tri thức mô tả cách giải quyết một vấn đề, quy trình xử lý các công việc, lịch trình tiến hành các thao tác … Các dạng của tri thức thủ tục thường dùng là các luật, chiến lược, lịch trình.

Ví dụ: Thuật toán giải phương trình bậc 2, quy trình kiểm tra xe máy "IF xe máy không khởi động được THEN đầu tiên kiểm tra bugi", …

3.2.4 Tri thức Heuristic

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.

Ví dụ: Nếu xe máy không khởi động được thì ta suy đoán bị hư hệ thống điện. Trong y khoa, nếu bệnh nhân bị ho và sốt cao thì chẩn đoán bệnh nhân bị viêm phổi

Tri thức **heuristic** là tri nông cạn do không đảm bảo hoàn toàn chính xác hoặc tối ưu theo một nghĩa nào đó về cách giải quyết vấn đề. Tri thức **heuristic** thường được coi là một mẹo nhằm dẫn dắt tiến trình lập luận. Chẳng hạn một số giải thuật tìm đường đi ngắn nhất, giải thuật A^* có thể được coi là lời giải của một vấn đề tốt nhưng chưa hẳn tối ưu.

3.3 CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC

3.3.1 Logic mệnh đề

3.3.1.1 Mệnh đề

Mệnh đề là một phát biểu có thể khẳng định tính đúng hoặc sai.

Các ký hiệu (symbol) của phép tính mệnh đề là các ký hiệu mệnh đề: P, Q, R, S, … (thông thường nó là các chữ cái in hoa nằm gần cuối bảng chữ cái tiếng Anh), các ký hiệu chân lý – chân trị (truth symbol): True, False hay các phép toán kết nối như: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, =

Các ký hiệu mệnh đề (propositional symbol) biểu thị các mệnh đề (proposition) hay các phát biểu về thế giới thực mà giá trị của chúng có thể là đúng hoặc sai.

Thí dụ: Các mệnh đề

P="Chiếc xe hơi kia màu đỏ"

R="Mặt trời mọc ở hướng đông"

3.3.1.2 Câu

Câu trong phép tính mệnh đề được cấu tạo từ những ký hiệu sơ cấp (atomic symbol) theo các luật sau đây :

- Tất cả các ký hiệu mệnh đề và ký hiệu chân lý đều là câu (sentences): True, P, Q và R là các câu.
- Phủ định của một câu là một câu: ¬ P và ¬ False là các câu
- Hội (và) của hai câu là một câu: P ∧ ¬ P là một câu
- Tuyển (hoặc) của hai câu là một câu: P ∨ ¬ P là một câu
- Kéo theo của một câu để có một câu khác là một câu: P ⇒ Q là một câu.
- Tương đương của hai câu là một câu: P ∨ Q = R là một câu

Các câu hợp lệ được gọi là các *công thức dạng chuẩn* (Well-Formed Formula) hay WFF.

Trong các câu phép tính mệnh đề, các ký hiệu ( ) và [ ] dùng để nhóm các ký hiệu vào các biểu thức con và nhờ đó kiểm soát được thứ tự của chúng trong việc đánh giá biểu thức và diễn đạt. Ví dụ (P ∨ Q) = R hoàn toàn khác với P ∨ (Q = R).

3.3.1.3 Biểu thức

Một biểu thức là một câu hay công thức dạng chuẩn, của phép tính mệnh đề khi và chỉ khi nó có thể được tạo từ những ký hiệu hợp lệ thông qua một dãy những luật này.

Thí dụ: ((P ∧ Q) ⇒ R = ¬ P ∨ ¬ Q ∨ R là một câu dạng chuẩn trong phép tính mệnh đề vì:

P, Q, R là các mệnh đề và do đó là các câu.

P ∧ Q, hội của hai câu là một câu.

(P ∧ Q) ⇒ R, kéo theo của một câu là một câu.

¬ P và ¬ Q, phủ định của các câu là câu.

¬ P ∨ ¬ Q, tuyển của hai câu là câu.

¬ P ∨ ¬ Q ∨ R, tuyển của hai câu là câu.

((P ∧ Q) ⇒ R = ¬ P ∨ ¬ Q ∨ R, tương đương của hai câu là câu.

Đây là câu xuất phát, nó đã được xây dựng thông qua một loạt các luật hợp lệ và do đó nó có dạng chuẩn.

3.3.1.4 Ngữ nghĩa của phép tính mệnh đề

Sự gán giá trị chân lý cho các câu mệnh đề được gọi là một sự *diễn giải* (interpretation), một sự khẳng định chân trị của chúng trong một *thế giới khả hữu* (possible world) nào đó. Một cách hình thức, một diễn giải là một ánh xạ từ các ký hiệu mệnh đề vào tập hợp {T, F}.

Phép gán giá trị chân lý cho các mệnh đề phức tạp thường được mô tả thông qua *bảng chân trị* như sau :

P	Q	¬ P	P ∧ Q	P ∨ Q	P ⇒ Q	P = Q
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

Thí dụ: Chân trị của mệnh đề (¬ P ∨ Q) = (P ⇒ Q) được cho như trong bảng sau :

P	Q	¬ P	¬ P ∨ Q	P ⇒ Q	(¬ P ∨ Q) = (P ⇒ Q)
F	F	T	T	T	T
F	T	T	T	T	T
T	F	F	F	F	T
T	T	F	T	T	T

Hai biểu thức trong phép tính mệnh đề là tương đương nhau nếu chúng có cùng giá trị trong mọi phép gán chân trị.

Một số công thức biến đổi tương đương của các mệnh đề:

Luật phủ định của phủ định: ¬(¬P) = P

Luật tương đương phép tuyển: (P∨Q) = (¬P ⇒ Q) hoặc (¬P ⇒ Q) = (¬P∨Q)

Luật tương phản: (P ⇒ Q) = (¬Q ⇒ ¬P)

Luật De Morgan: ¬(P ∨ Q) = (¬P ∧ ¬Q), và

¬(P ∧ Q) = (¬P ∨ ¬Q)

Luật giao hoán: (P ∧ Q) = (Q ∧ P), và

(P∨Q) = (Q∨P)

Luật kết hợp: ((P ∧ Q) ∧ R) = (P ∧ (Q ∧ R)), và

$((P \vee Q) \vee R) = (P \vee (Q \vee R))$

Luật phân phối: $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$, và

$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$

Những luật tương đương trên được dùng để biến đổi các biểu thức mệnh đề sang một dạng khác về mặt cú pháp nhưng tương đương về mặt logic. Có thể dùng các luật này thay cho các bảng chân trị để chứng minh hai biểu thức mệnh đề là tương đương nhau.

3.3.2 Logic vị từ

Trong phép tính mệnh đề, mỗi ký hiệu câu sơ cấp P, Q, … biểu thị một mệnh đề và chúng ta không thể tác động vào từng phần riêng lẻ của câu. Phép tính vị từ (predicate calculus) cung cấp cho chúng ta khả năng này. Chẳng hạn, đặt mệnh đề "hôm qua trời mưa" là P, từ đó chúng ta có thể tạo ra một vị từ chỉ thời tiết mô tả quan hệ giữa một ngày và thời tiết trong ngày ấy: thời_tiết (hôm_qua, mưa). Thông qua các luật suy diễn, chúng ta sẽ có thể thao tác trên các biểu thức phép tính mệnh đề, truy xuất và suy ra những câu mới.

3.3.2.1 Ký hiệu vị từ

Ký hiệu vị từ là tập hợp gồm các chữ cái, chữ số, ký hiệu "_" và bắt đầu bằng chữ cái.

Thí dụ: 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. Hằng được ký hiệu bắt đầu bằng chữ thường*: helen, yellow, rain, ...*
- Biến: dùng để chỉ một lớp tổng quát các đối tượng/thuộc tính. Biến được ký hiệu bắt đầu bằng chữ hoa: X, People, Students, …
- Hàm: dùng để chỉ một hàm trên các đối tượng. Hàm được ký hiệu bắt đầu bằng chữ thường: 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. Vị từ được ký hiệu bắt đầu bằng chữ thường: likes, equals, part_of, …
- Biểu thức hàm: là một ký hiệu hàm theo sau bởi n đối số.

  Ví dụ: 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 nằm trong cặp 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 nguyên tử, nguyên tử hay mệnh đề

Thí dụ: 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.

- Câu được tạo ra bằng cách kết hợp các câu sơ cấp sử dụng:
  + Các phép kết nối logic: ¬, ∧, ∨, ⇒, =
  + Các lượng từ:
    - Lượng từ phổ biến ∀: dùng để chỉ một câu là đúng với mọi giá trị của biến. Ví dụ: ∀ X likes(X, ice-cream).
    - Lượng từ tồn tại ∃: dùng để chỉ một câu là đúng với một số giá trị nào đó của biến. Ví dụ: ∃Y friends(Y,tom).

Thí dụ: likes(helen, chocolate) ∧ ¬ likes(bart, chocolate).

∃ X foo(X,two,plus(two,three)) ∧ equal(plus(three,two),five)

(foo(two, two,plus(two,three))) ⇒ (equal(plus(three,two),five)= true

3.3.2.2 Ngữ nghĩa - Phép tính vị từ

Tương tự như phép tính mệnh đề, ngữ nghĩa của phép tính vị từ cung cấp một cơ sở để xác định chân trị của các biểu thức dạng chuẩn. Chân trị của các biểu thức phụ thuộc vào ánh xạ từ các hằng, các biến, các vị từ và các hàm vào các đối tượng và quan hệ trong lĩnh vực được đề cập.

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 nguyên tố, sử dụng bảng chân lý cho cho các phép nối kết và:

- Giá trị của câu ∀ X <câu> là true nếu <câu> là T cho tất cả các phép gán có thể được cho X.
- Giá trị của câu ∃ X <câu> là true nếu tồn tại một phép gán cho X làm cho <câu> có giá trị T.

Thí dụ: Cho trước một tập hợp các quan hệ gia đình như sau:

mother (eve, abel) mother(eve,cain)

father (adam,abel) father(adam,cain)

∀X ∀Y father(X,Y) ∨ mother(X,Y) ⇒ parent(X,Y)

$\forall X\ \forall Y\ \exists Z$ parent(Z,X) $\wedge$ parent(Z,Y) $\Rightarrow$ sibling(X,Y)

Ta 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) (Tuy nhiên, kết quả này không có nghĩa)

3.3.2.3 Phép tính vị từ bậc nhất (First – order predicate calculus)

Phép tính vị từ bậc nhất cho phép các biến lượng giá 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.

Thí dụ:

Phép tính vị từ không hợp lệ: $\forall$(Likes) Likes(helen, ice-cream)

Phép tính vị từ hợp lệ:

Nếu ngày mai trời không mưa, Tom sẽ đi biển.

$\neg$weather(rain, tomorrow) $\Rightarrow$ go(tom, sea)

Tất cả các cầu thủ bóng rổ đều cao.

$\forall$ X (basketball_player(X) $\Rightarrow$ tall(X))

Có người thích coca-cola.

$\exists$ X person(X) $\wedge$ likes(X, coca-cola)

Không ai thích thuế.

$\neg\ \exists$ X likes(X, taxes)

Hầu hết bất kỳ câu đúng ngữ pháp nào cũng có thể biểu diễn trong phép tính vị từ bậc nhất bằng cách sử dụng các ký hiệu, các phép kết nối và ký hiệu biến.

3.3.2.4 Các luật suy diễn

Ngữ nghĩa của phép tính vị từ cung cấp một cơ sở cho lý thuyết hình thức về suy diễn logic. Khả năng suy ra những biểu thức đúng mới từ một tập hợp các khẳng định đúng là một đặc trưng quan trọng của phép tính vị từ. Logic vị từ dùng các luật suy diễn sau:

Luật Modus Ponens (MP):

$$\frac{P \quad P \Longrightarrow Q}{Q}$$

Thí dụ: Nếu ta có quan sát sau đây "nếu trời mưa thì sân ướt" (P ⇒ Q) và "trời đang mưa" (P) thì ta dễ dàng suy ra được "sân ướt" (Q).

Luật Modus Tollens (MT):

$$\frac{P \Longrightarrow Q \quad \neg Q}{\neg P}$$

Luật triển khai phổ biến (Universal Instantiation):

$$\frac{\forall X,\ P(X) \quad a\ thuộc\ miền\ xác\ định\ của\ X}{P(a)}$$

Thí dụ: Cho trước (1) ∀X (man(X) ⇒ mortal(X))

(2) man(socrates)

=> (3) man(socrates) ⇒ mortal(socrates) từ (1),(2) bằng luật UI.

(4) mortal(socrates) từ (3) và (2) bằng luật MP.

3.3.3 Hợp giải

Hợp giải là một kỹ thuật chứng minh định lý được biểu diễn bằng đại số mệnh đề hay đại số vị từ. Hợp giải được phát triển vào giữa thập niên 60 do Robinson đề xướng. Hợp giải sử dụng phản chứng để chứng minh một vấn đề. Hay nói cách khác, để chứng minh một câu (nghĩa là chứng minh câu này đúng), hợp giải sẽ chỉ ra rằng phủ định của câu sẽ tạo ra một mâu thuẫn với các câu đã biết.

Thủ tục hợp giải chứng minh bằng phản chứng (refutation). Nghĩa là để chứng minh một câu, nó sẽ chứng minh rằng phủ định của câu đó sẽ tạo ra một mâu thuẫn với các câu đã cho. Nhưng trước khi áp dụng thủ tục Robinson, các câu sẽ được chuyển về một dạng chuẩn mà ta gọi là dạng mệnh đề (clause form)

3.3.3.1 Chuyển về dạng mệnh đề (Clause form)

Nhu cầu chuyển câu về dạng mệnh đề:

Giả sử chúng ta có một luật như sau: "Tất cả những người Roman biết Marcus thì hoặc là sẽ ghét Caesar hoặc nghĩ rằng bất cứ ai ghét người khác là điên rồ", ta có thể biểu diễn tri thức này dưới dạng một công thức dạng chuẩn (wff) như sau:

$$\forall X\ [\text{roman}(X) \wedge \text{know}(X,\text{marcus}) \rightarrow$$

$$[\text{hate}(X,\text{caesar}) \vee (\forall Y\ (\exists Z\ \text{hate}(Y,Z) \rightarrow \text{thinkcrazy}(X,Y))\ )]] \qquad (1)$$

Để sử dụng công thức này trong một chứng minh đòi hỏi một quá trình đối sánh phức tạp. Quá trình này sẽ dễ dàng hơn nếu công thức được biểu diễn ở dạng đơn giản hơn. Công thức sẽ dễ dàng thao tác hơn nếu chúng nó:

- Phẳng hơn, nghĩa là có ít thành phần được nhúng vào.
- Các lượng tử biến (∀,∃) được tách khỏi phần còn lại của công thức để ta khỏi bận tâm xem xét chúng.

Dạng chuẩn Conjunctive Normal Form (CNF) hội đủ cả hai tính chất này.

CNF (Dạng chuẩn hội): một biểu thức mệnh đề được xem là ở dạng chuẩn hội nếu nó là một chuỗi các mệnh đề kết nối nhau bằng quan hệ AND (∧). Mỗi mệnh đề có dạng một tuyển OR (∨) của các biến mệnh đề.

Thí dụ: Các biểu thức sau ở dạng CNF :

- $(\neg a \vee c) \wedge (\neg a \vee \neg b \vee e) \wedge (c \vee \neg d \vee \neg e)$
- $(\neg \text{dog}(X) \vee \text{animal}(X)) \wedge (\neg \text{animal}(Y)\ \text{die}(Y)) \wedge (\text{dog}(\text{fido}))$

Công thức (1) được biểu diễn tương đương ở dạng CNF như sau:

$$\neg\text{roman}(X) \vee \neg\text{know}(X,\text{Marcus}) \vee \text{hate}(X,\text{Caesar}) \vee \neg\text{hate}(Y,Z) \vee \text{thinkcrazy}(X,Y)$$

Một vấn đề trong thực tế khi được biểu diễn trong hệ thống thường là ở dạng các câu *đúng* cùng một lúc (vì vậy đó là hội của các câu), mỗi câu có thể được biểu diễn bằng một tuyển. Vì vậy, tri thức của bài toán có thể được biểu diễn như là hội của các tuyển.

Ta có sẵn một giải thuật để chuyển bất kỳ một câu trong Logic vị từ hay công thức dạng chuẩn (WFF) về dạng CNF, mà vẫn không làm mất tính tổng quát nếu chúng ta sử dụng một thủ tục chứng minh như hợp giải để thao tác trên các WFF ở dạng này.

Vì vậy, để có thể sử dụng thủ tục Robinson, ta phải chuyển toàn bộ tri thức bài toán về dạng CNF hay nói khác hơn, chuyển từng câu về dạng *mệnh đề (clauses)*.

Định nghĩa mệnh đề:

Một mệnh đề được định nghĩa như là một WFF ở dạng CNF nhưng không có sự hiện diện của phép hội (∧) hay nói khác hơn mỗi mệnh đề là tuyển (∨) của các ***biến mệnh đề*** *(literal)*.

Trong thí dụ ở trên, dog(X), animal(X),.. là các biến mệnh đề.

Giải thuật chuyển về dạng mệnh đề:

Bước 1. Loại bỏ dấu → sử dụng công thức tương đương a → b = ¬a ∨ b

Bước 2. Thu hẹp phạm vi của toán tử ¬ về cho từng mục (term) đơn, sử dụng các tương đương:

a. ¬(¬p) = p
b. Luật De Morgan: ¬(a ∧ b) = ¬a ∨ ¬b hay ¬(a ∨ b) = ¬a ∧ ¬b
c. ¬∀X p(X) = ∃X ¬p(X) hay ¬∃X p(X) = ∀X ¬p(X)

Bước 3. Chuẩn hóa các biến sao cho mỗi lượng tử chỉ kết nối với một biến duy nhất. Vì các biến chỉ đơn giản là các tên để 'giữ chỗ', nên quá trình này không làm ảnh hưởng đến chân trị của WFF.

Ví dụ: ∀X p(X) ∨ ∀X q(X) có thể chuyển thành ∀X p(X) ∨ ∀Y q(Y)

Bước 4. Dịch chuyển tất cả các lượng tử về bên trái của công thức nhưng vẫn giữ nguyên thứ tự của chúng.

Bước 5. Xóa bỏ các lượng tử tồn tại (∃). Chúng ta có thể loại bỏ lượng tử tồn tại bằng cách thay thế biến đó bằng một hàm sinh ra giá trị mong muốn. Ta chỉ cần một hàm mới cho mỗi lần thay thế như vậy.

Ví dụ: ∃Y president(Y) được chuyển thành president(S1)

Với S1 là một hàm tạo ra giá trị thỏa mãn vị từ president.

Trong trường hợp lượng tử tồn tại xuất hiện bên trong phạm vi của lượng tử phổ biến, thì giá trị thỏa mãn vị từ này sẽ phải phụ thuộc vào giá trị của biến lượng tử tồn tại.

Ví dụ: ∀X ∃Y father_of (Y,X) được chuyển thành ∀X father_of (S2(X),X)

Các hàm (S1, S2) này được gọi là ***hàm Skolem***. Đôi khi các hàm Skolem không có đối số được gọi là ***hằng Skolem*** như S2.

Bước 6. Bỏ đi các tiền tố (lượng tử phổ biến)

Bước 7. Chuyển công thức về dạng hội của các tuyển. Sử dụng luật phân phối

(a ∧ b) ∨ c = (a ∨ c) ∧ (b ∨ c) hay (a ∨ b) ∧ c = (a ∧ c) ∨ (b ∧ c)

Ví dụ:

(winter ∧ wearingboots) ∨ (summer ∧ wearingsandals)

⇔ [(winter ∨ (summer ∧ wearingsandals)]∧[wearingboots ∨ (summer ∧ wearingsandals)]

⇔ (winter ∨ summer) ∧ (winter ∨ wearingsandals)

∧ (wearingboots ∨ summer) ∧ (wearingboots ∨ wearingsandals)

Bước 8. Tạo ra các mệnh đề tách biệt tương ứng với từng toán hạng tuyển trên.

Ví dụ: từ kết quả ở bước trên, ta có thể tách thành 4 mệnh đề.

Bước 9. Chuẩn hoá các biến trong tập hợp các mệnh đề vừa tạo ở bước 8, nghĩa là đặt lại tên cho các biến sao cho không có hai mệnh đề có cùng tên biến.

Thí dụ: Áp dụng giải thuật trên, hãy chuyển câu sau về dạng mệnh đề:

∀X [roman (X) ∧ know (X,marcus) →

[hate(X,caesar) ∨ (∀Y (∃Z hate(Y,Z) → thinkcrazy(X,Y)))]]

1. Loại bỏ dấu →

∀X [¬ **(**roman (X) ∧ know (X,marcus)**)** ∨

[hate(X,caesar) ∨ (∀Y (¬**(**∃Z hate(Y,Z)**)** ∨ thinkcrazy(X,Y)))]]

2. Đưa ¬ vào trong

∀X [**(**¬roman (X) ∨ ¬know (X,marcus)**)** ∨

[hate(X,caesar) ∨ (∀Y (∀Z **(**¬hate(Y,Z) **)** ∨ thinkcrazy(X,Y)))]]

3. Chuẩn hoá các biến
4. Dịch chuyển tất cả các lượng tử về bên trái:

∀X ∀Y ∀Z [**(**¬roman (X) ∨ ¬know (X,marcus) **)** ∨

[hate(X,caesar) ∨ (¬hate(Y,Z) ∨ thinkcrazy(X,Y))]]

5. Xoá bỏ các lượng tử tồn tại
6. Bỏ đi lượng tử phổ biến

[**(**¬roman (X) ∨ ¬know (X,marcus) **)** ∨

[hate(X,caesar) ∨ (¬hate(Y,Z) ∨ thinkcrazy(X,Y))]]

7. Chuyển thành hội của các tuyển: vì trong công thức trên không còn toán tử And, nên ở đây ta chỉ đơn giản là bỏ đi các dấu ngoặc là ta có được công thức ở dạng mệnh đề như sau:

¬roman (X) ∨ ¬know(X,marcus) ∨ hate(X,caesar) ∨ ¬hate(Y,Z) ∨ thinkcrazy(X,Y)

3.3.3.2 Cơ sở của Hợp giải (Resolution)

Thủ tục hợp giải là một quá trình lặp đơn giản: ở mỗi lần lặp, hai mệnh đề, gọi là mệnh đề cha, được so sánh (hay *giải quyết - resolved*), để tạo ra mệnh đề kết quả, trong đó

những biến mệnh đề mâu thuẫn nhau sẽ bị loại bỏ. Để hiểu lý do loại bỏ hai biến mệnh đề mâu thuẫn này, ta xét một trường hợp sau.

Giả sử trong hệ thống có hai mệnh đề (nghĩa là cùng lúc cả hai mệnh đề đều phải đúng):

$$(\text{winter} \vee \text{summer}) \text{ và } (\neg\text{winter} \vee \text{cold})$$

Tại bất kỳ thời điểm nào, chỉ có một trong hai biến mệnh đề *winter* và ¬*winter* là đúng.

Nếu *winter* đúng, thì buộc *cold* phải đúng. Còn nếu ¬*winter* đúng thì buộc *summer* phải đúng. Vì vậy, từ hai mệnh đề trên có thể dẫn xuất thành:

$$\text{summer} \vee \text{cold}$$

Đây chính là dẫn xuất mà hợp giải sẽ sử dụng. Hợp giải sẽ thao tác trên hai mệnh đề có chứa cùng biến mệnh đề (trong ví dụ trên là *winter*). Biến mệnh đề này phải ở dạng khẳng định trong một mệnh đề và ở dạng phủ định trong mệnh đề kia. Kết quả đạt được bằng cách kết hợp hai mệnh đề cha trừ đi biến mệnh đề chung đó.

Nếu mệnh đề kết quả là rỗng thì xem như đã tìm được *sự mâu thuẫn* (contradiction), nghĩa là mục tiêu đã được chứng minh.

Trong hai phần kế tiếp, ta sẽ xét giải thuật hợp giải sử dụng trong hai trường hợp, đó là trường hợp vấn đề được biểu diễn bằng ngôn ngữ logic mệnh đề và trường hợp vấn đề được biểu diễn bằng ngôn ngữ logic vị từ.

3.3.3.3 Giải thuật hợp giải dùng cho Logic mệnh đề

Cho trước: Tập hợp các tiên đề (axioms) F viết dưới dạng các câu trong phép tính mệnh đề.

Yêu cầu: Chứng minh P

Giải thuật Hợp giải dùng cho Phép tính mệnh đề (Propositional Logic):

Bước 1. Chuyển tất cả các câu trong F về dạng mệnh đề (clause form)

Bước 2. Lấy phủ định P và chuyển về dạng mệnh đề. Thêm nó vào tập các mệnh đề vừa tạo ở bước 1.

Bước 3. Lặp lại cho đến khi tìm thấy sự mâu thuẫn hoặc không thể tiếp tục:

3.1 Chọn hai mệnh đề. Gọi là các mệnh đề cha.

3.2 Hợp giải chúng. Mệnh đề kết quả là tuyển của tất cả các biến mệnh đề trong các mệnh đề cha với lưu ý nếu có bất kỳ các cặp biến mệnh đề L và ¬L, một nằm

trong mệnh đề cha này, một nằm trong mệnh đề cha kia, thì chọn một cặp và xóa cả hai L và ¬L ra khỏi mệnh đề kết quả.

3.3 Nếu mệnh đề kết quả là rỗng, thì xem như đã tìm được sự mâu thuẫn. Nếu không, thêm mệnh đề kết quả đó vào trong tập hợp các mệnh đề hiện có.

Thí dụ:

Cho cơ sở tri thức bao gồm các câu:

(1) P

(2) $(P \wedge Q) \rightarrow R$

(3) $(S \vee T) \rightarrow Q$

(4) T

Hãy chứng minh R.

Bước 1: Chuyển về dạng mệnh đề

(1) $\Leftrightarrow$ P

(2) $\Leftrightarrow$ $\neg P \vee \neg Q \vee R$

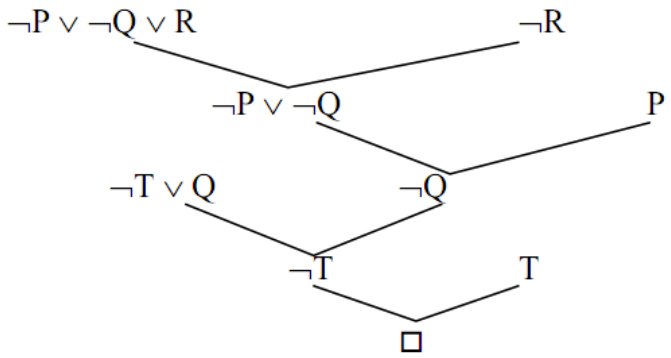
(3) $\Leftrightarrow$ $\neg S \vee Q$

$\neg T \vee Q$

(4) $\Leftrightarrow$ T

Bước 2: Phủ định mệnh đề cần chứng minh ta được: ¬R

Bước 3: Hợp giải



Kết luận: Cơ sở tri thức sau khi bổ sung ¬R dẫn đến mâu thuẫn. Vậy R đúng.

3.3.3.4 Giải thuật hợp giải dùng cho Logic vị từ

Trong trường hợp biểu diễn bằng Logic mệnh đề, việc xác định hai biến mệnh đề mâu thuẫn (không thể đúng cùng lúc) là rất dễ dàng, chỉ cần đơn giản tìm *L* và ¬*L*. Trong khi biểu diễn bằng Logic vị từ, quá trình đối sánh này phức tạp hơn vì cần phải xem xét các đối số của vị từ. Chẳng hạn như, man(John) và ¬man(John) thì mâu thuẫn, còn man(John) và ¬man(Spot) thì không. Vì vậy, để xác định các cặp mâu thuẫn, ta cần

một thủ tục để so sánh các biến mệnh đề và tìm xem liệu có tồn tại tập phép thế nào làm cho chúng giống nhau.

Cho trước: Tập hợp các tiên đề F viết dưới dạng các câu trong phép tính vị từ.

Yêu cầu: Chứng minh P

Giải thuật Hợp giải dùng cho Phép tính vị từ (Predicate Logic):

Bước 1. Chuyển tất cả các câu trong tập F về dạng mệnh đề (clause form)

Bước 2. Lấy phủ định của P và chuyển về dạng mệnh đề. Thêm nó vào tập các mệnh đề vừa tạo ở bước 1.

Bước 3. Lặp lại cho đến khi tìm thấy sự mâu thuẫn hay không thể tiếp tục:

3.1 Chọn hai mệnh đề. Gọi là các mệnh đề cha.

3.2 Hợp giải chúng. Mệnh đề kết quả là tuyển của tất cả các biến mệnh đề trong các mệnh đề cha với các phép thế phù hợp với lưu ý: nếu có một cặp biến mệnh đề T1 và ¬T2, sao cho T1 nằm trong mệnh đề cha này, còn ¬T2 nằm trong mệnh đề cha kia và nếu T1 và T2 là hai biến mệnh đề có thể đồng nhất *(unifiable),* thì xóa cả hai T1 và ¬T2 ra khỏi mệnh đề kết quả. Ta nói T1 và T2 là các *biến mệnh đề bù nhau* (complementary literals). Sử dụng tập phép thế trả ra bởi giải thuật đồng nhất để tạo ra mệnh đề kết quả. Nếu có nhiều hơn một cặp biến mệnh đề bù nhau thì chỉ xóa một cặp.

3.3 Nếu mệnh đề kết quả là rỗng, thì xem như đã tìm được sự mâu thuẫn.

3.4 Nếu không, thêm mệnh đề kết quả đó vào trong tập hợp các mệnh đề hiện có.

Việc lựa chọn mệnh đề nào để đưa vào hợp giải trước sẽ ảnh hưởng đến quá trình chứng minh cho mục tiêu. Vì vậy, để tăng tốc độ cho quá trình chứng minh này, một số chiến lược được đưa ra hỗ trợ cho việc lựa chọn này:

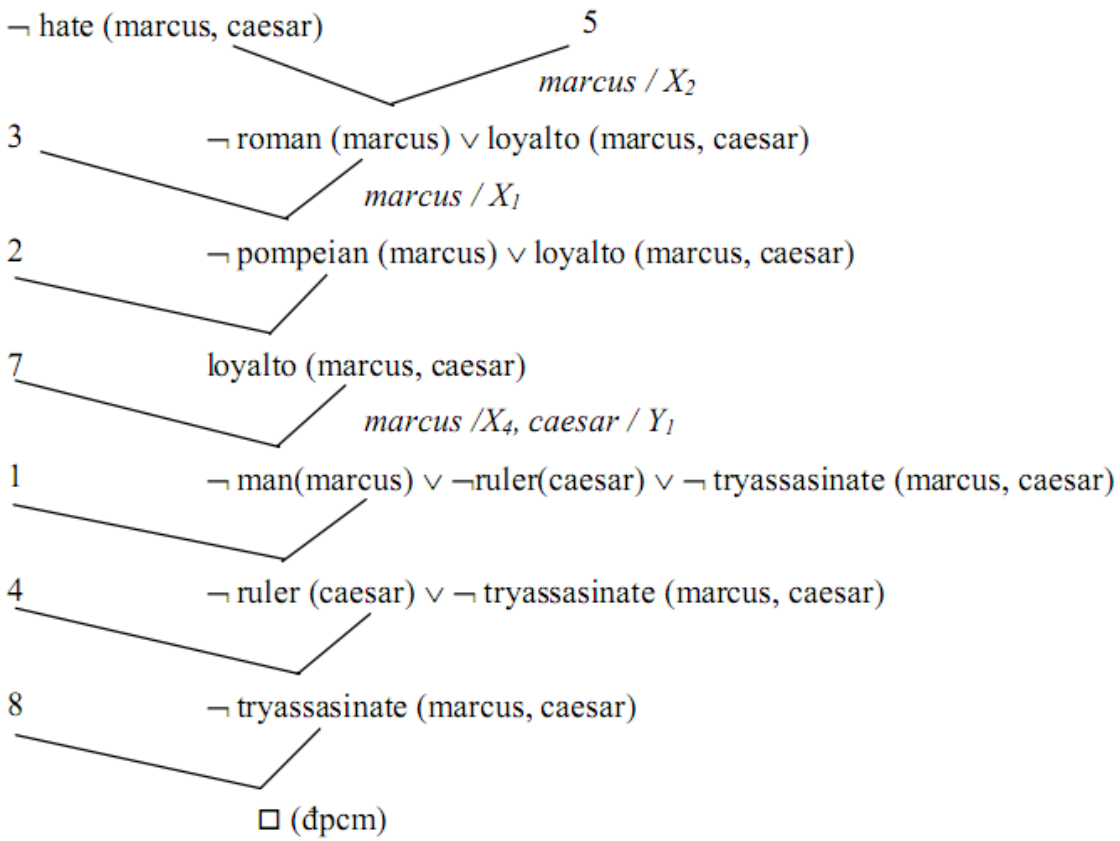
- Chỉ hợp giải những cặp mệnh đề có chứa các biến mệnh đề bù nhau.
- Loại bỏ các mệnh đề ngay khi chúng vừa được tạo ra trong quá trình hợp giải. Có hai loại mệnh đề có thể loại bỏ được là: mệnh đề luôn luôn đúng (tautology) và mệnh đề được tạo thành từ các mệnh đề khác (ví dụ P ∨ Q được tạo thành từ P).
- Mỗi khi có thể, hãy hợp giải với một trong những mệnh đề là một phần của câu mà ta cần phản chứng hoặc với một mệnh đề được sinh ra do hợp giải với mệnh đề như vậy. Chiến lược này gọi là set-of-support. Nó phát sinh từ trực giác cho rằng sự mâu thuẫn mà chúng ta tìm kiếm phải liên quan đến câu mà ta đang muốn chứng minh.

- Mỗi khi có thể, hợp giải với mệnh đề chỉ có một biến mệnh đề. Hợp giải này sẽ tạo ra mệnh đề mới với ít biến mệnh đề hơn các mệnh đề cha của nó. Vì vậy có thể nó sẽ gần đến mục tiêu là một mệnh rỗng hơn. Chiến lược này gọi là unit-preference.

Thí dụ: Giả sử ta có các câu dạng mệnh đề như sau:

1. man (marcus)
2. pompeian (marcus)
3. ¬pompeian (X_1) ∨ Roman (X_1)
4. ruler (caesar)
5. ¬roman (X_2) ∨ loyalto (X_2, caesar) ∨ hate (X_2, caesar)
6. loyato (X_3, fl(X_3))
7. ¬man (X_4) ∨ ¬ ruler (Y_1) ∨ ¬ tryassasinate (X_4, Y_1) ∨ ¬loyalto (X_4, Y_1)
8. tryassasinate (marcus, caesar)

Chứng minh: hate (marcus, caesar)



Đến đây, chúng ta đã biết cách thức sử dụng hợp giải một cách cơ bản để chứng minh cho các vấn đề được biểu diễn ở ngôn ngữ logic mệnh đề hay logic vị từ. Phần kế tiếp chúng ta sẽ thảo luận về tính chất dừng của thủ tục này, nghĩa là thủ tục có khả năng phát hiện các trường hợp không tồn tại sự mâu thuẫn và kết thúc việc chứng minh. Hai

phần kế tiếp nữa sẽ thảo luận về sự mở rộng của ngôn ngữ logic vị từ hỗ trợ cho sự tính toán và trả lời câu hỏi.

3.3.3.5 Hợp giải có thể phát hiện trường hợp không tồn tại sự mâu thuẫn

Đối với những vấn đề không chứng minh được, nghĩa là quá trình hợp giải không thể kết thúc bằng kết quả rỗng, thì hợp giải vẫn có khả năng phát hiện được và cho phép thủ tục dừng, đồng thời đưa ra kết luận là không thể chứng minh được.

Giả sử với câu hỏi đưa ra là 'Did Marcus hate Caesar?'. Trong trường hợp này, ta có thể sẽ đưa ra câu cần chứng minh là ¬ hate (marcus, caesar).

Để chứng minh câu này, ta thêm vào tập hợp các mệnh đề một mệnh đề: hate (marcus, caesar)

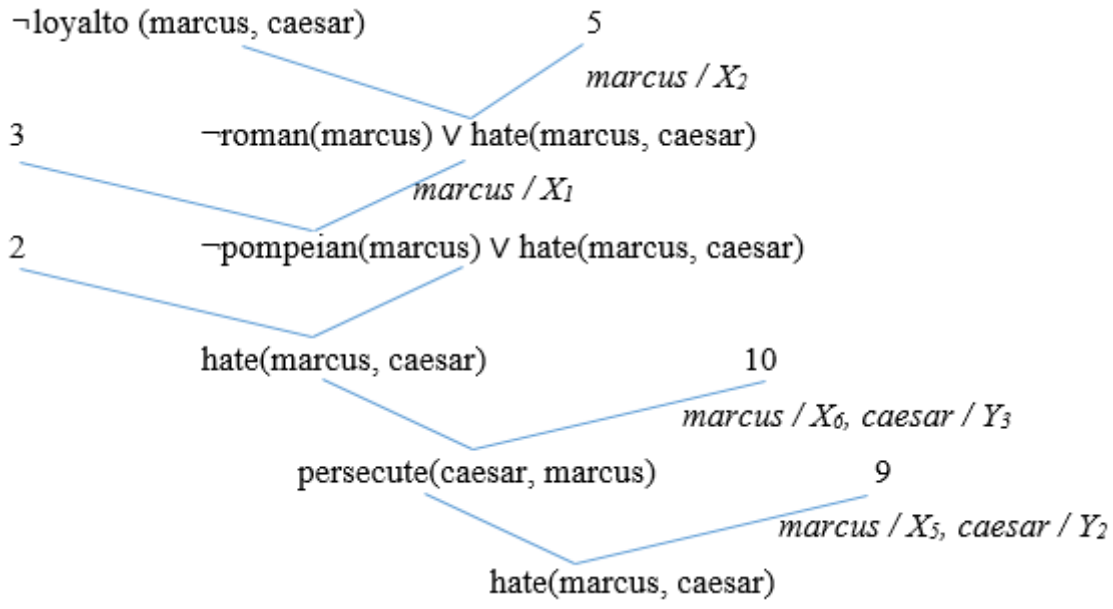
Và bắt đầu chứng minh. Nhưng ngay lập tức ta sẽ thấy là việc chứng minh không thể được tiếp tục vì không có mệnh đề nào chứa biến mệnh đề ¬hate. Vì vậy, ta có thể kết luận hate (marcus, caesar) sẽ không tạo ra sự mâu thuẫn nào với các câu đã cho.

Đôi khi hợp giải không phát hiện tình huống này ngay bước khởi đầu, mà có thể sau một vài bước hợp giải như trong ví dụ sau.

Thí dụ: Giả sử trong cơ sở tri thức của ta có thêm hai câu:

9. persecute(X,Y) → hate(Y,X) ⇔ ¬persecute(X_5,Y_2) ∨ hate(Y_2,X_5)
10. hate(X,Y) → persecute(Y,X) ⇔ ¬hate(X_6,Y_3) ∨ persecute(Y_3,X_6)

Ta cần chứng minh: loyalto (marcus, caesar)



Bây giờ để khẳng định là không tồn tại sự mâu thuẫn ta phải phát hiện rằng mệnh đề kết quả *hate(marcus, caesar)* chính là mệnh đề đã được tạo ra trước đó. Hay nói cách khác, mặc dù ta có thể tạo ra mệnh đề kết quả, nhưng không tạo ra mệnh đề mới.

3.4 Tổng kết chương

Trong chương này, chúng ta đã được giới thiệu phép tính vị từ như một ngôn ngữ biểu diễn dùng cho việc giải quyết vấn đề trong AI. Chúng ta đã định nghĩa các khái niệm về biến, hàng, hàm, biểu thức, … và ngữ nghĩa của ngôn ngữ này. Dựa trên ngữ nghĩa của phép tính vị từ, các luật suy diễn cũng cho phép suy diễn các câu từ một tập hợp các biểu thức cho trước.

Phần cuối chương ta đã khảo sát một phương pháp suy luận tự động, đó là phương pháp chứng minh Hợp Giải (resolution). Một ứng dụng quan trọng của hợp giải đó là làm nền tảng cho trình thông dịch của ngôn ngữ PROLOG đang được sử dụng hiện nay.

Hợp giải là một kỹ thuật chứng minh định lý được biểu diễn bằng đại số mệnh đề hay đại số vị từ. Hợp giải được phát triển vào giữa thập niên 60 do Robinson đề xướng. Hợp giải sử dụng phản chứng để chứng minh một vấn đề. Hay nói cách khác, để chứng minh một câu (nghĩa là chứng minh câu này đúng), hợp giải sẽ chỉ ra rằng phủ định của câu sẽ tạo ra một mâu thuẫn với các câu đã biết.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

1. Chuyển các câu sau đây thành câu trong logic vị từ:
   a. Tất cả các con mèo đều là động vật.
   b. Không có con chó nào là loài bò sát.
   c. Tất cả các nhà khoa học máy tính đều thích một hệ điều hành nào đó.
   d. Mọi đứa trẻ đều thích Coca-cola.
   e. Không có đứa trẻ nào thích ăn rau.
   f. Một số người thích kẹo, một số khác thì không.
   g. Không có sinh viên nào học mà thi rớt môn này.
2. Đưa câu vị từ sau về dạng mệnh đề:

$$\forall X \,(\, [\, a(X) \wedge b(X)] \rightarrow [c(X,I) \wedge \exists Y \,(\exists Z[c(Y,Z)] \rightarrow d(X,Y))] \,) \vee \forall X(e(X))$$

3. Ta có cơ sở tri thức như sau:
   a. Ông Nam là thợ hớt tóc hoặc ca sĩ.
   b. Thợ hớt tóc thì có mái tóc ấn tượng.
   c. Ca sĩ có nhiều fan và nhiều email.
   d. Ai có nhiều email thì sẽ luôn bận rộn.
   e. Ông Nam luôn có nhiều thời gian rảnh để giải trí.

   Hãy biểu diễn cơ sở tri thức trên bằng logic vị từ và sử dụng hợp giải để chứng minh ông Nam là thợ hớt tóc.

4. Giả sử các hiểu biết của một chuyên gia trong một tình huống nào đó được phát biểu dưới dạng các biểu thức logic mệnh đề sau đây:
   a. $a \wedge ((a \wedge x) \rightarrow d)$
   b. $(a \rightarrow (b \wedge c)) \wedge x$
   c. $(c \rightarrow a) \rightarrow (d \wedge e)$
   d. $b \rightarrow ((d \wedge x) \rightarrow f)$
   e. $(e \wedge b) \rightarrow f$
   f. $(e \wedge y) \rightarrow g$
   g. $d \rightarrow f$
   h. $f \rightarrow (\neg a \vee g)$

Hãy chứng minh g đúng bằng thủ tục hợp giải.

5. Câu chuyện dưới đây được lấy từ quyển sách *Algorithms + Data structures = Programs* (Thuật toán + Cấu trúc dữ liệu = Chương trình) của N. Wirth (1976):

"Tôi cưới một góa phụ (W), bà ta có một cô con gái đã lớn (D). Cha tôi (F), người thường xuyên đến thăm chúng tôi đã phải lòng cô con riêng của vợ tôi và cưới cô ta. Vì thế cha tôi trở thành con rể tôi và con ghẻ tôi trở thành mẹ tôi. Vài tháng sau đó, vợ

tôi sinh một đứa con trai (S_1), nó trở thành em rể của bố tôi, cũng như trở thành chú tôi. Vợ của bố tôi, tức là con ghẻ của tôi, cũng sinh một đứa con trai (S_2).”

Sử dụng phép tính vị từ, hãy tạo ra một tập hợp các biểu thức biểu diễn hoàn cảnh trong câu chuyện trên. Hãy đưa ra các biểu thức định nghĩa các quan hệ gia đình cơ bản như định nghĩa bố vợ … và sử dụng modus ponens trên hệ này để chứng minh kết luận “Tôi cũng chính là ông tôi”.

6. Tìm hiểu chương trình tư vấn tài chính và xác định phương thức đầu tư thích hợp cho Jane trong tình huống sau: Jane có bốn thành viên gia đình phụ thuộc, thu nhập bằng lương hàng tháng là 30.000$ và tài khoản tiết kiệm của cô ta là 20.000$.
7. Giải thích các kết quả suy luận trong ví dụ ở mục 3.3.2.2

Chương 4 HỆ CHUYÊN GIA DỰA TRÊN LUẬT

4.1 MỞ ĐẦU

Trong cuộc sống, sở dĩ các chuyên gia có thể giải quyết vấn đề ở một mức độ cao vì họ có rất nhiều tri thức về lĩnh vực họ hoạt động. Thực tế hiển nhiên và đơn giản này chính là cơ sở nền tảng cho việc thiết kế các máy giải quyết vấn đề dựa trên tri thức mà ta thường gọi là hệ chuyên gia. Một hệ chuyên gia sử dụng tri thức của một lĩnh vực cụ thể để cung cấp việc giải quyết vấn đề với "chất lượng chuyên gia" trong lĩnh vực đó.

Thông thường, các nhà thiết kế Hệ chuyên gia thu thập tri thức này, bao gồm lý thuyết đến cả các kinh nghiệm, kỹ xảo, phương pháp làm tắt, chiến lược heuristic đã tích lũy được của các chuyên gia con người qua quá trình làm việc của họ trong một lĩnh vực chuyên môn. Từ tri thức này, người ta cố gắng cài đặt chúng vào hệ thống để hệ thống có thể mô phỏng theo cách thức các chuyên gia làm việc. Tuy nhiên, không giống với con người, các chương trình hiện tại không tự học lấy kinh nghiệm: mà tri thức phải được lấy từ con người và mã hóa thành ngôn ngữ hình thức. Đây là nhiệm vụ chính mà các nhà thiết kế Hệ chuyên gia phải đương đầu.

Do bản chất heuristic và tri thức chuyên sâu của việc giải quyết vấn đề cấp độ chuyên gia, các hệ chuyên gia nói chung có khả năng:

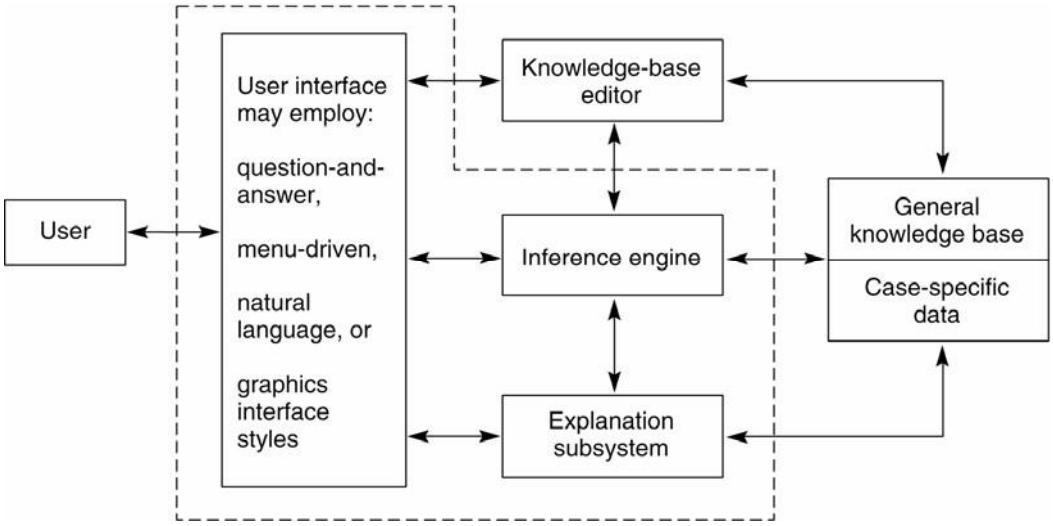
- Cung cấp sự kiểm tra đối với các quá trình suy luận của chúng, bằng cách hiển thị các bước trung gian và bằng cách trả lời câu hỏi về quá trình giải.
- Cho phép sửa đổi dễ dàng, cả khi thêm và xóa các kỹ năng giải quyết vấn đề vào cơ sở tri thức (knowledge based).
- Suy luận một cách heuristic, sử dụng tri thức (thường là không hoàn hảo) để tìm lời giải hữu ích cho vấn đề.

Người ta đã xây dựng các hệ chuyên gia để giải quyết hàng loạt những vấn đề trong các lĩnh vực như y học, toán học, công nghệ, hóa học, địa chất, khoa học máy tính, kinh doanh, luật pháp, quốc phòng và giáo dục. Các chương trình này đã giải quyết một lớp rộng các loại vấn đề như:

- Diễn giải (interpretation) – hình thành những kết luận hay mô tả cấp cao từ những tập hợp dữ liệu thô.
- Dự đoán (prediction) – tiên đoán những hậu quả có thể xảy ra khi cho trước một tình huống.
- Chẩn đoán (diagnosis) – xác định nguyên nhân của những sự cố trong các tình huống phức tạp dựa trên các triệu chứng có thể quan sát được.

- Thiết kế (design) – tìm ra cấu hình cho các thành phần hệ thống, đáp ứng được các mục tiêu trong khi vẫn thỏa mãn một tập hợp các ràng buộc về thiết kế.
- Lập kế hoạch (planning) – tìm ra một chuỗi các hành động để đạt được một tập hợp các mục tiêu, khi được cho trước các điều kiện khởi đầu và những ràng buộc trong thời gian chạy (run-time).
- Theo dõi (monitoring) – so sánh hành vi quan sát được của hệ thống với hành vi mong đợi.
- Bắt lỗi và sửa chữa (debuging and repair) – chỉ định và cài đặt những phương pháp chữa trị cho các trục trặc.
- Hướng dẫn (instruction) – phát hiện và sửa chữa những thiếu sót trong quan niệm của học viên về một chủ đề lĩnh vực nào đó.
- Điều khiển (control) – chỉ đạo hành vi của một môi trường phức tạp.

4.1.1 Thiết kế của một Hệ chuyên gia dựa trên luật (Rule–Based ES)



Hình 4.1 Kiến trúc của một hệ chuyên gia tiêu biểu.

Hình 4.1 cho thấy các module quan trọng nhất tạo nên một hệ chuyên gia dựa trên luật. Người dùng tương tác với hệ chuyên gia thông qua một *giao diện người dùng* (user interface), giao diện này đơn giản hóa việc giao tiếp và che giấu phần lớn sự phức tạp của hệ thống (ví dụ như cấu trúc bên trong của cơ sở các luật). Các hệ chuyên gia sử dụng một số lượng phong phú các kiểu giao diện, bao gồm hỏi và trả lời, điều khiển bởi trình đơn, ngôn ngữ tự nhiên hay đồ họa, … Việc quyết định sử dụng giao diện như thế nào là sự thỏa hiệp giữa nhu cầu của người dùng với những đòi hỏi của cơ sở tri thức và hệ suy diễn.

Trái tim của hệ chuyên gia là *cơ sở tri thức tổng quát* (general knowledge base), chứa tri thức giải quyết vấn đề của một ứng dụng cụ thể. Trong một hệ chuyên gia dựa trên luật, tri thức này được biểu diễn dưới dạng các luật if… then…. Cơ sở tri thức bao

gồm tri thức tổng quát (general knowledge) cũng như thông tin của một tình huống cụ thể (case-specific).

Động cơ suy diễn (inference engine) áp dụng tri thức cho việc giải quyết các bài toán thực tế. Về căn bản nó là một trình thông dịch cho cơ sở tri thức. Trong hệ sinh (production system), động cơ suy diễn thực hiện chu trình điều khiển nhận dạng – hành động (recognize-act control cycle). Việc tách biệt cơ sở tri thức ra khỏi động cơ suy diễn là rất quan trọng vì nhiều lý do:

- Sự tách biệt của tri thức dùng để giải quyết vấn đề và động cơ suy diễn sẽ tạo điều kiện cho việc biểu diễn tri thức theo một cách tự nhiên hơn. Ví dụ, các luật if…then… gần gũi với cách con người mô tả những kỹ thuật giải quyết vấn đề của họ hơn so với một chương trình đưa luôn tri thức này vào phần mã máy tính cấp thấp.
- Bởi vì cơ sở tri thức được cách ly khỏi các cấu trúc điều khiển cấp thấp của chương trình, các nhà xây dựng Hệ chuyên gia có thể tập trung một cách trực tiếp vào việc nắm bắt và tổ chức giải quyết vấn đề hơn là phải thực hiện trên các chi tiết của việc cài đặt vào máy tính.
- Sự phân chia tri thức và điều khiển cho phép thay đổi một phần cơ sở tri thức mà không tạo ra các hiệu ứng lề trên các phần khác của chương trình.
- Sự tách biệt này cũng cho phép một phần mềm điều khiển và giao tiếp có thể được sử dụng cho nhiều hệ thống khác nhau. Một trình cốt lõi của Hệ chuyên gia (expert system shell) có tất cả các thành phần của hình 4.1, trừ phần cơ sở tri thức và dữ liệu của tình huống cụ thể. Các nhà lập trình có thể sử dụng "trình cốt lõi rỗng của Hệ chuyên gia" và tạo ra một cơ sở tri thức mới thích hợp với ứng dụng của họ. Các đường chấm chấm trong hình 4.1 biểu thị các module của trình cốt lõi.
- Sự module hóa này cho phép chúng ta thử nghiệm nhiều chế độ điều khiển khác nhau trên cùng một cơ sở luật.

Một *hệ con giải thích* (explanation subsystem) cho phép chương trình giải thích quá trình suy luận của nó cho người dùng. Các giải thích này bao gồm các lập luận biện minh cho các kết luận của hệ thống (trả lời cho câu hỏi *how*), giải thích vì sao hệ thống cần dữ liệu đó (trả lời cho câu hỏi *why*), …

Nhiều Hệ chuyên gia còn bao gồm một *trình soạn thảo cơ sở tri thức* (knowledge-base editor).

Trình soạn thảo này giúp các nhà lập trình xác định và hiệu chỉnh lỗi trong quá trình làm việc của chương trình, thường là bằng cách truy xuất những thông tin cung cấp bởi hệ con giải thích. Chúng cũng có thể hỗ trợ cho việc bổ sung tri thức mới, giúp duy trì cú pháp luật chính xác và thực hiện các kiểm tra tính nhất quán trên cơ sở tri thức đã cập nhật.

Việc sử dụng trình cốt lõi Hệ chuyên gia có thể giúp ta giảm đáng kể thời gian thiết kế và cài đặt chương trình. Ví dụ, Hệ chuyên gia MYCIN dùng để chẩn đoán bệnh viêm màng não xương sống và nhiễm trùng máu được phát triển trong khoảng 20 năm-người (personyear). EMYCIN (Empty MYCIN) là một chương trình cốt lõi của MYCIN được tạo ra bằng cách loại bỏ tri thức chuyên ngành khỏi chương trình MYCIN. Sử dụng EMYCIN, các kỹ sư tri thức đã cài đặt PUFF, một chương trình phân tích các vấn đề về phổi cho các bệnh nhân, trong khoảng 5 năm-người. Đây là một sự tiết kiệm đáng kể và là một khía cạnh quan trọng chứng tỏ sức sống trên phương diện thương mại của công nghệ hệ chuyên gia.

Đối với người lập trình, điều quan trọng là phải lựa chọn đúng chương trình cốt lõi Hệ chuyên gia để phát triển một Hệ chuyên gia mới. Các bài toán khác nhau đòi hỏi các quá trình lập luận khác nhau: tìm kiếm hướng đích so với tìm kiếm hướng dữ liệu chẳng hạn. Chiến lược điều khiển cung cấp bởi trình cốt lõi cần phải thích hợp với ứng dụng; phương pháp suy luận y học dùng cho ứng dụng PUFF rất giống với phương pháp mà Hệ chuyên gia MYCIN sử dụng; điều đó làm cho việc sử dụng trình cốt lõi EMYCIN trở nên thích hợp. Nếu trình cốt lõi có quá trình suy luận không thích hợp, thì việc sử dụng nó có thể là một sai lầm và tệ hơn là bắt đầu từ xây dựng từ đầu.

4.1.2 Các vấn đề phù hợp để xây dựng Hệ chuyên gia

Các Hệ chuyên gia thường đòi hỏi sự đầu tư đáng kể về tiền bạc và sức lực con người. Những cố gắng để giải quyết một bài toán quá phức tạp, quá ít hiểu biết hoặc có những yếu tố không phù hợp khác đối với công nghệ hiện tại có thể dẫn đến những thất bại hao tiền tốn của. Các nhà nghiên cứu đã xây dựng một tập hợp các chỉ dẫn có tính không hình thức cho việc xác định xem khi nào thì một bài toán thích hợp để giải quyết bằng Hệ chuyên gia:

1. Sự cần thiết phải có một giải pháp phải biện minh cho chi phí và sức lực của việc xây dựng Hệ chuyên gia, vì nếu không đó sẽ là một sự lãng phí. Công ty máy tính DEC đã bỏ ra tiền và công sức để xây dựng Hệ chuyên gia XCON để tự động hóa công tác tạo cấu hình cho máy tính. Kết quả, XCON đã giúp công ty tiết kiệm được tài chính và duy trì lòng tin của khách hàng. Tương tự, người ta đã cây dựng nhiều Hệ chuyên gia trong các lĩnh vực như khai thác khoáng sản, kinh doanh quốc phòng và y học là những nơi tồn tại tiềm năng to lớn cho việc tiết kiệm tiền bạc, thời gian và sinh mạng con người.

2. Hiểu biết chuyên môn của con người không sẵn có ở mọi nơi cần đến nó. Có rất nhiều Hệ chuyên gia đã được xây dựng trong ngành y, vì sự chuyên môn và tính phức tạp trong kỹ thuật của y học hiện đại đã khiến cho các bác sĩ gặp nhiều khó khăn trong việc theo kịp những tiến bộ của các phương pháp chẩn đoán và điều trị. Số lượng các chuyên gia đáp ứng được các yêu cầu này rất hiếm và phải trả thù lao cao cho công

việc của họ, do đó mà các hệ chuyên gia được nhìn nhận như là một cứu cánh. Nhờ có Hệ chuyên gia, các bác sĩ ở địa phương có thể chẩn đoán và điều trị bệnh ở mức độ chuyên gia. Trong ngành địa chất, bằng cách bố trí Hệ chuyên gia tại những nơi khai thác, nhiều vấn đề có thể được giải quyết mà không cần các chuyên gia con người phải có mặt. Tương tự, sự lãng phí những kiến thức chuyên môn có giá trị do việc chuyển đổi nhân viên hay những người sắp về hưu có thể biện minh cho việc xây dựng các hệ chuyên gia.

3. Vấn đề có thể được giải quyết bằng cách sử dụng các kỹ thuật suy luận ký hiệu. Những giải pháp vấn đề không nên đòi hỏi sự khéo léo tay chân hay khả năng cảm thụ. Tuy rằng chúng ta đã có những robot và các hệ nhìn, hiện tại chúng vẫn thiếu khả năng tinh tế và linh hoạt của con người. Các Hệ chuyên gia chỉ thích hợp với các vấn đề mà con người có thể giải quyết thông qua sự suy luận trên các ký hiệu.

4. Phạm vi xác định vấn đề được cấu trúc tốt và không đòi hỏi sự suy luận theo lẽ thường tình (commonsense reasoning). Mặc dù các hệ chuyên gia đã được xây dựng trong nhiều lĩnh vực đòi hỏi tri thức kỹ thuật chuyên môn hóa, nhiều sự suy luận theo lẽ thường của con người vẫn nằm ngoài khả năng hiện tại của chúng. Các lĩnh vực mang tính kỹ thuật cao có ưu điểm là được hiểu biết và hình thức hóa kỹ lưỡng: các thuật ngữ được định nghĩa đầy đủ và các chủ đề đã có sẵn những mô hình khái niệm rõ ràng và cụ thể.

5. Vấn đề có thể không giải quyết được bằng cách sử dụng các phương pháp tính toán truyền thống. Không nên sử dụng công nghệ Hệ chuyên gia để "phát minh lại chiếc bánh xe" (reinvent the wheel). Nếu một bài toán có thể giải quyết một cách thỏa đáng bằng những kỹ thuật truyền thống hơn như các phương pháp số, thống kê, … thì chúng không phải là một ứng viên cho việc thiết kế Hệ chuyên gia.

6. Có sự hợp tác và hiểu ý nhau giữa các chuyên gia. Tri thức sử dụng bởi các chuyên gia hiếm khi tìm thấy trong các sách giáo khoa mà chỉ có được nhờ kinh nghiệm và sự đánh giá của con người khi làm việc trong lĩnh vực đó. Điều quan trọng là các chuyên gia đó sẵn sàng và có khả năng chia sẻ tri thức. Các chuyên gia có thể sẽ không hợp tác khi họ cảm thấy bị đe dọa bởi hệ thống, sợ rằng hệ thống có thể thay thế họ hoặc nghĩ rằng đề án không thể thành công, chỉ lãng phí thời gian của họ, khi đó họ sẽ không bỏ ra thời gian và nỗ lực cần thiết. Một điều quan trọng khác là ban quản lý phải hỗ trợ dự án và chi phép các chuyên gia sử dụng thời gian làm việc chính thức của họ để cộng tác với các kỹ sư tri thức.

7. Vấn đề cần giải quyết phải có kích thước và quy mô đúng mức. Vấn đề không được vượt quá khả năng của công nghệ hiện tại. Ví dụ, một chương trình cố gắng nắm bắt mọi hiểu biết chuyên môn của một bác sĩ y khoa sẽ không khả thi trong khi đó một

chương trình cố vấn cho các bác sĩ trong việc sử dụng một phần nào đó của thiết bị chẩn đoán sẽ thích hợp hơn.

Mặc dù một vấn đề lớn có thể không phù hợp với giải pháp của hệ chuyên gia, ta có thể phân chia nó thành các vấn đề nhỏ hơn, độc lập nhau và phù hợp với Hệ chuyên gia. Điều này tương ứng với chiến lược phân rã "top-down" sử dụng trong công nghệ phần mềm truyền thống. Một cách giải quyết khác là ta bắt đầu bằng một chương trình đơn giản có khả năng giải quyết một phần của vấn đề rồi từng bước tăng dần chức năng của nó để xử lý phạm vi lớn hơn của vấn đề. Chiến lược này gọi là "đường kẻ mỏng" (thin line), ám chỉ mục tiêu của nó là tạo ra một khuôn mẫu hy sinh bề rộng của khả năng ứng dụng nhằm ưu tiên việc tạo ra một lời giải đầy đủ cho tập hợp nhỏ các bài toán thử nghiệm. Kỹ thuật này tỏ ra hiệu quả để khám phá các vấn đề phức tạp và chưa được hiểu rõ. Người ta sử dụng thành công kỹ thuật này trong việc tạo ra XCON: ban đầu chương trình được thiết kế chỉ để định cấu hình cho các máy tính VAX 780; sau đó nó được mở rộng để xử lý luôn tất cả các hệ máy VAX và hệ máy PDP11.

4.2 VÍ DỤ VỀ CHẨN ĐOÁN LỖI TRÊN HỆ CHUYÊN GIA DỰA TRÊN LUẬT

Để có một ví dụ cụ thể hơn về giải quyết vấn đề trong hệ chuyên gia, ta xét một Hệ chuyên gia nhỏ dùng để chẩn đoán những trục trặc trong xe hơi, gọi tắt là Hệ chuyên gia "Chẩn đoán xe hơi":

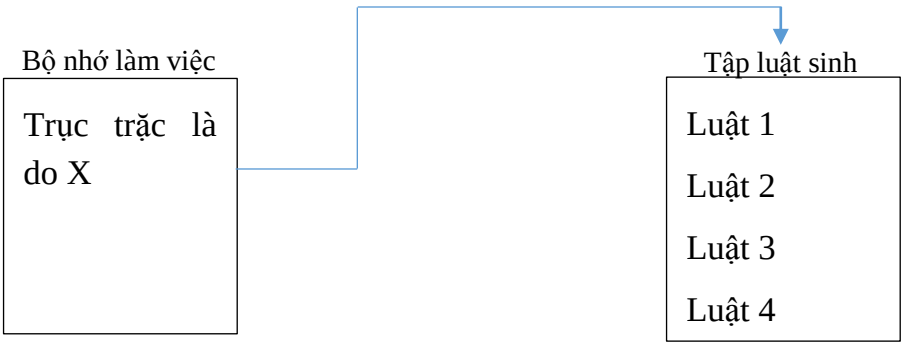
Luật 1 **IF** động cơ nhận được xăng **AND** động cơ không khởi động được
THEN trục trặc là do bugi.

Luật 2 **IF** động cơ không khởi động được **AND** đèn không sáng
THEN trục trặc là do acquy hoặc dây cáp

Luật 3 **IF** động cơ không khởi động được **AND** đèn sáng
THEN trục trặc là do môtơ khởi động

Luật 4 **IF** còn xăng trong bình chứa nhiên liệu **AND** còn xăng trong bộ chế hòa khí
THEN động cơ nhận được xăng

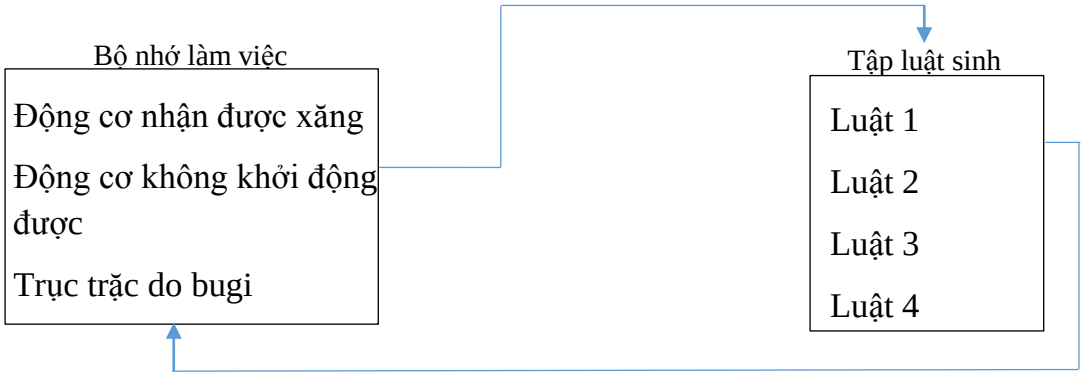
Trong chế độ điều khiển hướng từ mục tiêu, đầu tiên mục tiêu cao nhất là "trục trặc là do X" sẽ được đưa vào bộ nhớ làm việc như hình 4.2:



Hình 4.2 Hệ sinh tại thời điểm ban đầu của một lần chẩn đoán.

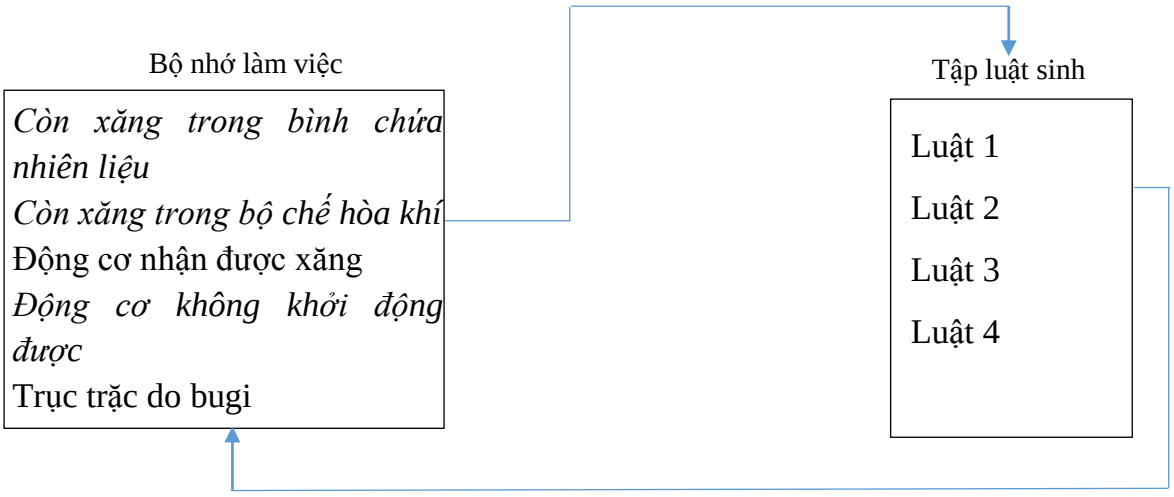
Có 3 luật đối sánh (match) với biểu thức này trong bộ nhớ làm việc: luật 1, 2 và 3.

Nếu ta chọn luật ưu tiên theo số thứ tự của nó, thì luật 1 sẽ được sử dụng, khi đó X sẽ được gắn kết (bound) với giá trị *bugi* và những tiền đề (vế trái) của luật 1 được đặt vào bộ nhớ làm việc như hình 4.3.



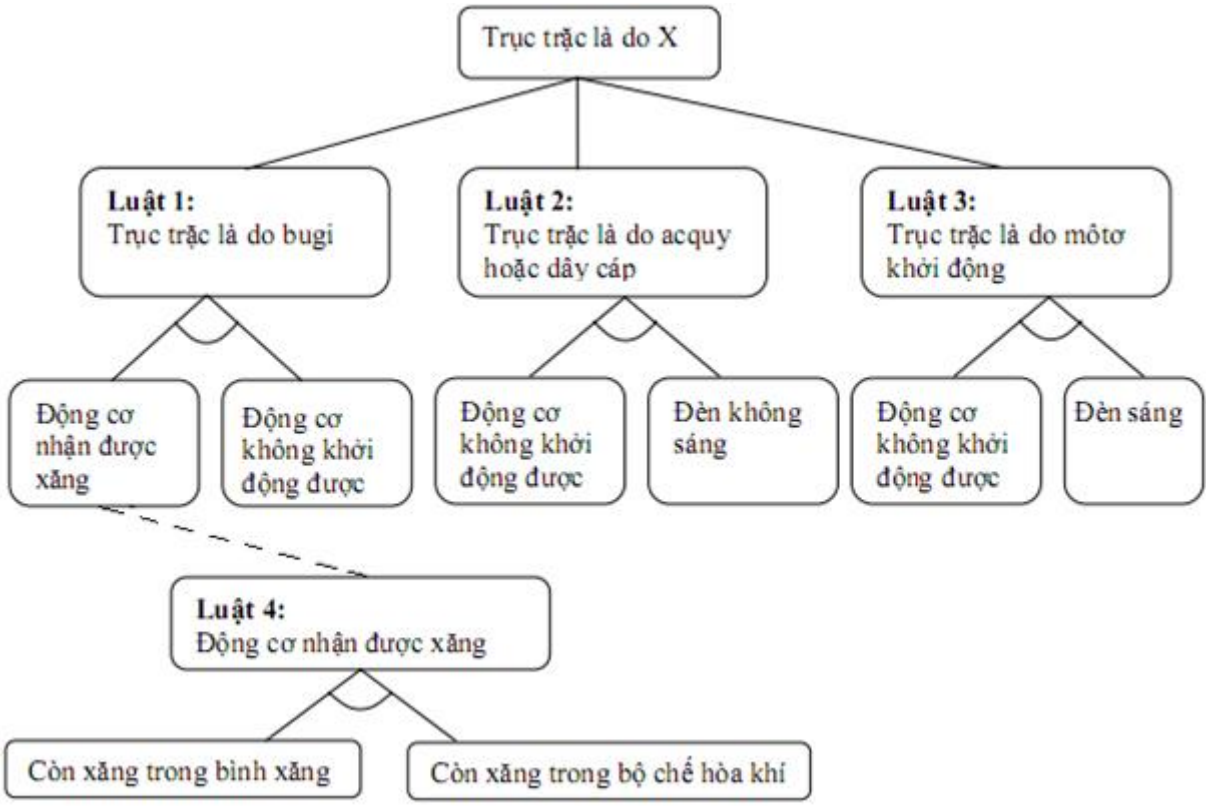
Hình 4.3 Hệ sinh sau khi luật 1 được sử dụng

Để chứng minh mục tiêu con là động cơ nhận được xăng thì luật 4 được đốt cháy và những tiền đề của luật này được đặt vào bộ nhớ làm việc như hình 4.4.



Hình 4.4 Hệ sinh sau khi sử dụng luật 4.

Tại thời điểm này, có ba mục trong bộ nhớ làm việc (các mục in nghiêng trong hình 4.4) là không đối sánh với bất kỳ kết luận của luật nào. Trong tình huống này, Hệ chuyên gia sẽ truy vấn trực tiếp người dùng về những mục tiêu mới này. Nếu người dùng xác nhận cả ba mục tiêu này đều đúng, thì Hệ chuyên gia sẽ xác định một cách thành công rằng trục trặc xe là do bugi. Trong quá trình tìm ra lời giải này, hệ thống đã kiểm tra nhánh trái nhất của đồ thị Và/Hoặc trong hình 4.5.



Hình 4.5. Đồ thị Và/Hoặc được tìm kiếm trong ví dụ chẩn đoán xe ôtô.

Tiếp theo máy tính yêu cầu người dùng cho biết về các mục tiêu con hiện có trong bộ nhớ làm việc (trả lời của người dùng được gạch dưới):

- *còn xăng trong bình chứa nhiên liệu?* yes
- *còn xăng trong bộ chế hòa khí?* yes
- *động cơ khởi động được không?* why

Đến đây, người dùng đánh vào câu hỏi why, nghĩa là muốn hỏi, tại sao anh cần biết động cơ có khởi động được hay không? Khi đó hệ thống trả lời

Đã xác định được rằng:

1. *động cơ nhận được xăng, vì vậy nếu*
2. *động cơ không khởi động được ,*

Thì có thể kết luận: trục trặc là do bugi

Giả sử người dùng tiếp tục hỏi:

động cơ nhận được xăng như thế nào?

Khi đó, hệ thống sẽ trả lời:

Điều này có từ luật 4

IF còn xăng trong bình chứa nhiên liệu AND còn xăng trong bộ chế hòa khí

THEN động cơ nhận được xăng

Người dùng đã cho biết: còn xăng trong bình chứa nhiên liệu

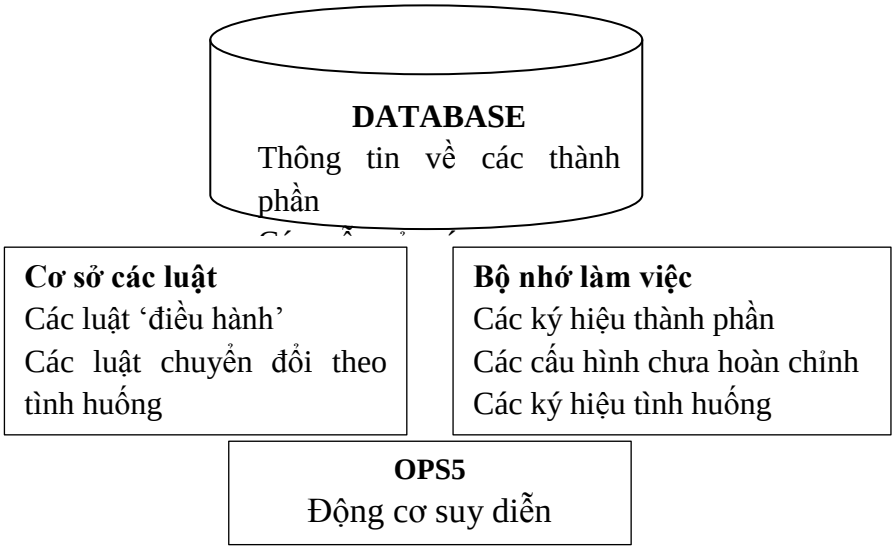
Người dùng đã cho biết: còn xăng trong bộ chế hòa khí

Một lần nữa, kiến trúc hệ sinh cho chúng ta một cở sở thiết yếu để có những giải thích này. Mỗi chu trình của vòng lặp điều khiển sẽ chọn và sử dụng một luật khác. Chương trình có thể dừng lại sau mỗi chu trình và cho phép người dùng kiểm tra. Vì mỗi luật biểu diễn một bó tri thức đầy đủ, nên luật hiện tại cho ta một ngữ cảnh để giải thích. Ngoài ra, tính module của mỗi luật cũng tạo điều kiện cho việc thêm hay sửa đổi một luật được thực hiện một cách dễ dàng mà không hề ảnh hưởng đến các luật khác.

4.3 HỆ CHUYÊN GIA R1/XCON

Hệ chuyên gia R1 hay XCON được đưa ra vào năm 1982 là một trong những Hệ chuyên gia thương mại thành công đầu tiên. Như đã giới thiệu trong các phần trên, ban đầu chương trình được thiết kế chỉ để định cấu hình cho các máy tính VAX 780; sau đó nó được mở rộng để xử lý luôn cho các hệ máy VAX và hệ máy PDP-11. Đầu vào của hệ thống là đơn đặt hàng của khách hàng và sau đó áp dụng các luật để sinh ra một cấu hình hoàn chỉnh, thống nhất cùng với các sơ đồ biểu diễn mối quan hệ về không gian và logic của các bộ phận trong máy (McDermott 1981).

Kiến trúc của Hệ chuyên gia XCON được mô tả như hình 4.6.



Hình 4.6 Kiến trúc của HỆ CHUYÊN GIA XCON

XCON là một Hệ chuyên gia dựa trên luật, nó chứa một *cơ sở các luật* bao gồm khoảng 10000 luật được viết với ngôn ngữ OPS5 (OPS – Official Production System). Các luật này dùng để điều khiển quá trình tạo cấu hình cho máy, nó có một đặc điểm đặc biệt là tiền đề của luật thường bắt đầu bằng một điều kiện kiểm tra tình huống hiện tại, chẳng hạn như:

IF *Tình huống mới nhất hiện tại là đang phân phối các thiết bị Massbus*

& có một ổ đĩa cổng đơn mà chưa được gán cho một Massbus

& không có ổ đĩa cổng kép được gán

& số lượng thiết bị mà mỗi Massbus cần hỗ trợ đã được biết

& có một Massbus đã được gán ít nhất một ổ đĩa và cần phải hỗ trợ ổ đĩa bổ sung

& các loại cáp cần thiết để kết nối ổ đĩa với các thiết bị trước đó đã được biết đến

THEN gán đĩa cho Massbus

Trong luật trên, điều kiện đầu tiên (được in nghiêng) là điều kiện kiểm tra xem tình huống mới nhất hiện tại (most current active context) có phải là đang phân phối các thiết bị Massbus hay không? Nếu đúng thì động cơ suy diễn mới kiểm tra các điều kiện kế tiếp, nếu sai thì bỏ qua luật này. Để có thể biết được tình huống hiện tại là gì, hệ thống sẽ truy cập vào bộ nhớ làm việc, nơi có lưu trữ các ký hiệu về tình huống hiện tại. Chính nhờ cấu trúc luật đặc biệt này mà mặc dù cơ sở luật có đến 10000 luật nhưng hệ thống thực hiện quá trình đối sánh để đưa ra các luật khả thi tại một thời điểm là rất nhanh. Với tính chất này, có thể nói công việc tạo cấu hình của R1/XCON được xem như là một hệ thống phân chia thành các công việc nhỏ hơn với sự phụ thuộc thời gian (temporal dependency) rất mạnh, công việc này phải thực hiện trước công việc kia.

Ngoài ký hiệu tình huống, *bộ nhớ làm việc* của hệ thống còn chứa các thông tin như ký hiệu của các thành phần đang có, ký hiệu các cấu hình đang thực hiện dở dang, chưa hoàn chỉnh. Vì các thành phần của máy thường được đặc trưng bởi nhiều thông số kỹ thuật, nên bộ nhớ làm việc chỉ chứa ký hiệu của nó, còn thông tin về các thông số này sẽ được lưu trong một cơ sở dữ liệu.

Động cơ suy diễn của XCON được phát triển bằng OPS5, sử dụng tiếp cận tìm kiếm hướng từ dữ liệu hay suy diễn tiến. Kết quả của hệ thống có thể nói là tương đương hoặc có khi tốt hơn giải pháp của các chuyên gia con người. Về tốc độ, thì HỆ CHUYÊN GIA cho ra giải pháp nhanh gấp mười lần chuyên gia con người và đã tiết kiệm được cho công ty DEC hàng năm khoảng 25 triệu đôla.

4.4 KẾT LUẬN VỀ HỆ CHUYÊN GIA DỰA TRÊN LUẬT

4.4.1 Ưu điểm của Hệ chuyên gia dựa trên luật:

1. Khả năng sử dụng trực tiếp các tri thức thực nghiệm của các chuyên gia.
2. Tính module của luật làm cho việc xây dựng và bảo trì luật dễ dàng.

3. Có thể thực hiện tốt trong các lĩnh vực hạn hẹp.
4. Có tiện ích giải thích tốt.
5. Các luật ánh xạ một cách tự nhiên vào không gian tìm kiếm trạng thái.
6. Dễ dàng theo dõi một chuỗi các luật và sửa lỗi.
7. Sự tách biệt giữa tri thức và điều khiển giúp đơn giản hóa quá trình phát triển Hệ chuyên gia

4.4.2 Khuyết điểm của Hệ chuyên gia dựa trên luật:

1. Các luật đạt được từ các chuyên gia mang tính heuristic rất cao. Chẳng hạn như trong lĩnh vực y học, luật "If sốt-cao Then bị-nhiễm-trùng" là sự kết hợp trực tiếp các triệu chứng quan sát được và các chẩn đoán, mà không thể hiện sự hiểu biết lý thuyết sâu hơn về lĩnh vực chuyên ngành (như cơ chế phản ứng của cơ thể để chống lại vi trùng) hoặc luật "If sốt-cao Then cho uống-Aspirin" cũng không thể hiện tri thức về giải quyết vấn đề tức là quá trình chữa bệnh như thế nào.
2. Các luật heuristic "dễ vỡ", không thể xử lý các trường hợp ngoài dự kiến. Vì các luật được tạo ra từ kinh nghiệm của các chuyên gia trên những tình huống đã biết, nên khi gặp phải một tình huống mới không đúng với các kinh nghiệm đó, thì các luật này không giải quyết được.
3. Có khả năng giải thích chứ không chứng minh. Hệ chuyên gia dựa trên luật chỉ có thể giải thích rằng kết luận này là do suy luận từ các luật như thế nào, chứ không chứng minh được kết luận đó là đúng.
4. Các tri thức thường rất phụ thuộc vào công việc. Quá trình thu thập tri thức rất phức tạp và khó khăn, tuy nhiên, tri thức có được không thể sử dụng lại cho một công việc khác.
5. Khó bảo trì các cơ sở luật lớn.

4.5 Tổng kết chương

Nội dung chính của chương này bao gồm:

1. Hệ chuyên gia là một hệ thống sử dụng các tri thức thu thập được từ kinh nghiệm của các chuyên gia để giải quyết các vấn đề ở mức độ chuyên gia.
2. Không phải vấn đề nào cần giải quyết cũng phù hợp với giải pháp hệ chuyên gia.
3. Một Hệ chuyên gia được phát triển theo phương pháp lập trình thăm dò.
4. Việc thu thập tri thức từ chuyên gia là một việc làm khó khăn và cần có sự hợp tác chặt chẽ giữa kỹ sư tri thức và các chuyên gia.
5. Mô hình khái niệm là một công cụ hỗ trợ cho quá trình thu thập tri thức.
6. Hệ chuyên gia dựa trên luật có kiến trúc và cơ chế hoạt động tương tự như một hệ sinh, đây là một trong những kỹ thuật đầu tiên và được sử dụng rộng rãi nhất dùng cho biểu diễn tri thức trong lĩnh vực Hệ chuyên gia.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

1. Trong mục 4.2 đã giới thiệu một tập hợp luật dùng cho chẩn đoán các trục trặc trong ô tô. Hãy cho biết ai có thể là những kỹ sư tri thức, những chuyên gia chuyên ngành và người dùng cuối cho một ứng dụng như vậy? Thảo luận các ý kiến này.
2. Chọn một lĩnh vực khác thích hợp đối với việc thiết kế một hệ chuyên gia. Xây dựng một tập hợp luật minh họa cho ứng dụng này.
3. Tìm hiểu, thảo luận và nêu ví dụ về suy luận dựa trên mô hình (MODEL – BASED REASONING) và suy luận dựa trên trường hợp (caSe– BASED REASONING)
4. Xây dựng hệ thống luật suy diễn lùi chẩn đoán hỏng hóc trong máy tính.
5. Tìm hiểu một số hệ hỗ trợ ra quyết định đã được ứng dụng thực tế, thảo luận về các hệ hỗ trợ ra quyết định này.

Chương 5 MÁY HỌC

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.

5.1 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 nhận biết một *sự khái quát* (generalization) từ một tập hợp các ví dụ cụ thể. *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.

5.2 CÁC TIẾP CẬN HỌC

Có ba tiếp cận học: *tiếp cận ký hiệu* (symbol-based learning), *tiếp cận mạng neuron* (neural networks) hay *mạng kết nối* (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 chúng ta chỉ khảo sát một giải thuật được sử dụng rộng rãi, đó là giải thuật quy nạp cây quyết định ID3.

Ngược lại với tiếp cận ký hiệu, tiếp cận mạng neuron (mạng 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.

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.

5.2.1 Tiếp cận ký hiệu - Giải thuật quy nạp cây quyết định ID3

5.2.1.1 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, cách tiếp cận nó trong việc quản lý tính phức tạp, heuristic dùng cho việc chọn lựa các khái niệm ứng viên và tiềm năng đối với việc xử lý dữ liệu nhiễu của nó.

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 sẽ 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:

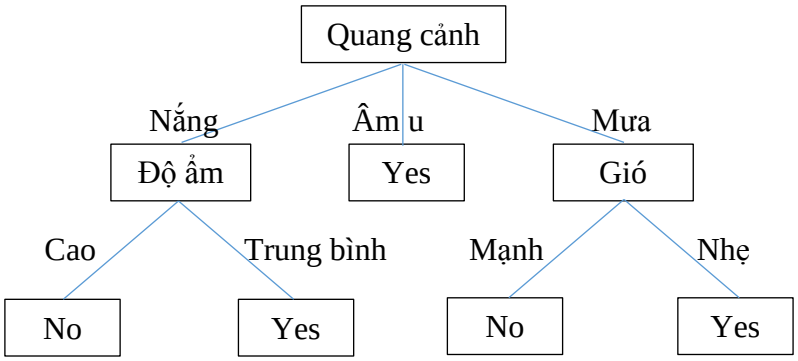
Ngày	Quang cảnh	Nhiệt độ	Độ ẩm	Gió	Chơi Tennis
D1	Nắng	Nóng	Cao	Nhẹ	Không
D2	Nắng	Nóng	Cao	Mạnh	Không
D3	Âm u	Nóng	Cao	Nhẹ	Có
D4	Mưa	Ấm áp	Cao	Nhẹ	Có
D5	Mưa	Mát	Trung bình	Nhẹ	Có
D6	Mưa	Mát	Trung bình	Mạnh	Không
D7	Âm u	Mát	Trung bình	Mạnh	Có
D8	Nắng	Ấm áp	Cao	Nhẹ	Không
D9	Nắng	Mát	Trung bình	Nhẹ	Có
D10	Mưa	Ấm áp	Trung bình	Nhẹ	Có
D11	Nắng	Ấm áp	Trung bình	Mạnh	Có
D12	Âm u	Ấm áp	Cao	Mạnh	Có
D13	Âm u	Nóng	Trung bình	Nhẹ	Có
D14	Mưa	Ấm áp	Cao	Mạnh	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, gió và đều có 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). Nói cách khác, phân loại 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 5.1 - 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. Cây quyết định sẽ không thay đổi cho đến khi 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. Vì vậy, ta nên học một cây quyết định như thế nào là tốt?

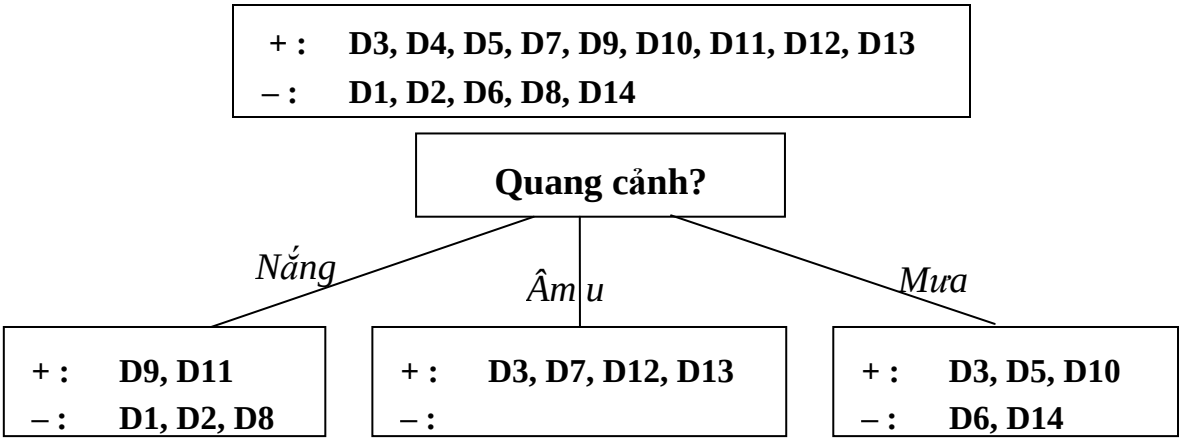
Occam's Razor và một số lập luận khác đều cho rằng 'giả thuyết có khả năng nhất là giả thuyết đơn giản nhất thống nhất với tất cả các quan sát', ta nên chấp nhận những câu trả lời đơn giản nhất đáp ứng một cách đúng đắn dữ liệu của chúng ta. Trong trường hợp này, các giải thuật học cố gắng tạo ra cây 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.

5.2.1.2 Giải thuật ID3

ID3 xây dựng cây quyết định 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; khi đó thuật toán được 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 quyết định 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 quyết định, với việc giả định đã có một hàm chọn trắc nghiệm thích hợp. Phần kế tiếp sẽ trình bày heuristic chọn lựa của ID3.

Từ tập ví dụ rèn luyện trên, cách xây dựng cây quyết định của ID3 như sau:



Hình 5.2 - *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 5.2 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***

kết quả là một nút lá được gán nhãn bởi lớp đó

else if tập_thuộc_tính là rỗng ***then***

kết quả là nút lá được gán nhãn bởi tuyển 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;

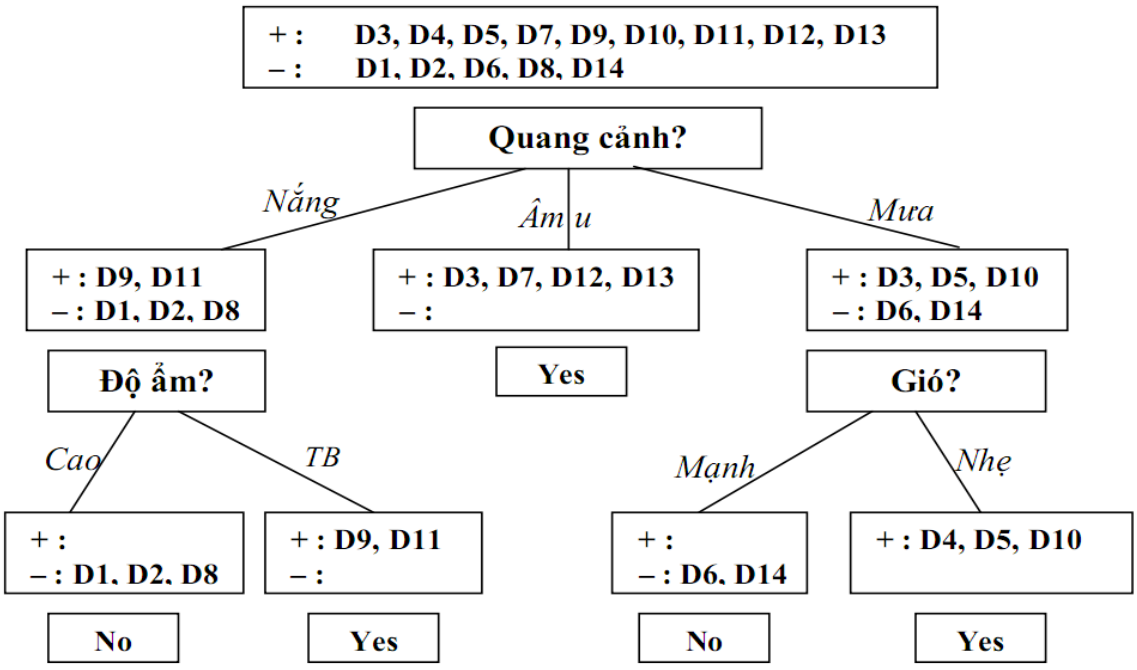
Đặt vào phân_vùng V các ví dụ trong tập_ví_dụ có giá trị V tại thuộc tính P;

Gọi induce_tree(phân_vùng, tập_thuộc_tính), gắn kết quả vào nhánh V

end

end

end



Hình 5.3 - Cây quyết định đã xây dựng xong.

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 5.3.

Lưu ý, để phân loại một ví dụ, có khi cây QĐ không cần sử dụng tất cả các thuộc tính đã cho và nó vẫn phân loại đúng tất cả các ví dụ.

5.2.1.3 Thuộc tính 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.

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 nghĩa là tập ví dụ S chỉ toàn ví dụ thuộc cùng một loại, hay S là thuần nhất.
- Entropy(S) = 1 nghĩa là tập ví dụ S có các ví dụ thuộc các loại khác nhau với độ pha trộn là cao nhất.
- 0 < Entropy(S) < 1 nghĩa là tập ví dụ S có số lượng ví dụ thuộc các loại khác nhau là không bằng nhau.

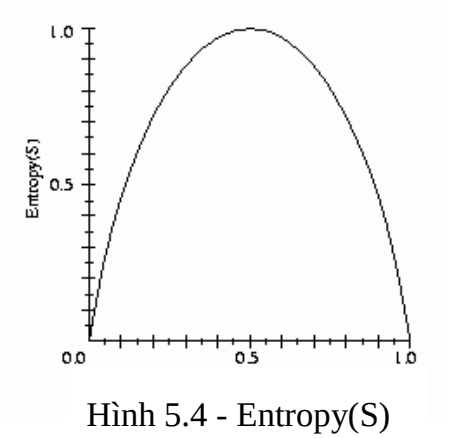
Để đơn giản ta xét trường hợp các ví dụ của S chỉ thuộc loại âm (-) hoặc dương (+).

Hình 5.4 minh họa sự phụ thuộc của giá trị entropy vào xác suất xuất hiện của ví dụ 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:



Hình 5.4 - Entropy(S)

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

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:

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

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:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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.

5.2.2 Tiếp cận mạng kết nối

5.2.2.1 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 đơn giản (neuron sinh học hay neuron nhân tạo), 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 sao cho 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 cách 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.

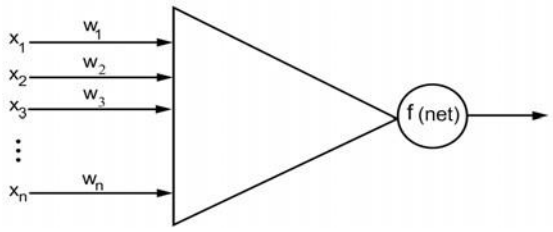
Các bài toán thích hợp với tiếp cận kết nối thường là:

- 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): phân biệt các tín hiệu với nền, tìm ra các thành phần không quan trọng trong một tín hiệu.

5.2.2.2 Cơ bản về mạng kết nối

Thành phần cơ bản của một mạng neuron là một neuron nhân tạo, như mô tả trong hình 5.4 sau đây.

5.2.2.3 Neuron nhân tạo



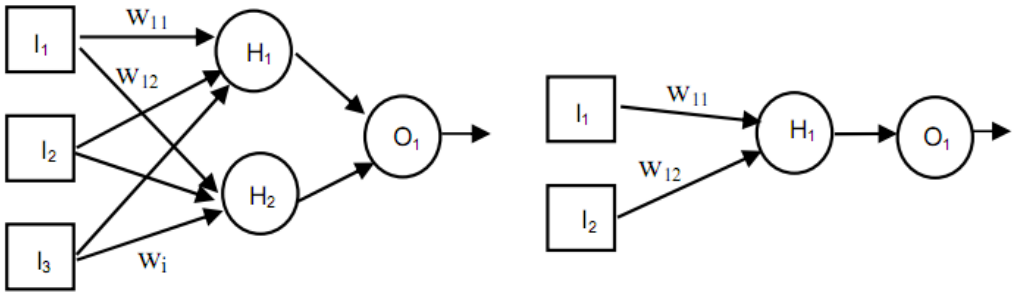
Hình 5.4 - Một neuron nhân tạo.

- **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.

5.2.2.4 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 5.5 - Các hình thái mạng neuron khác nhau.

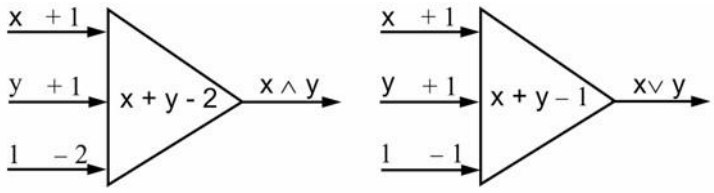
- **Giải thuật học (learning algorithm)**: là giải thuật dùng để điều chỉnh các trọng số ở các đầu vào của các neuron.
- **Sơ đồ mã hóa (encoding schema)**: bao gồm việc thông dịch dữ liệu thực tế thành các giá trị đầu vào của mạng và việc thông dịch giá trị đầu ra của mạng thành một kết quả có ý nghĩa.

5.2.2.5 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 5.6 minh họa các neuron McCulloch-Pitts dùng để tính hàm logic *and* và *or*.



Hình 5.6 - 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

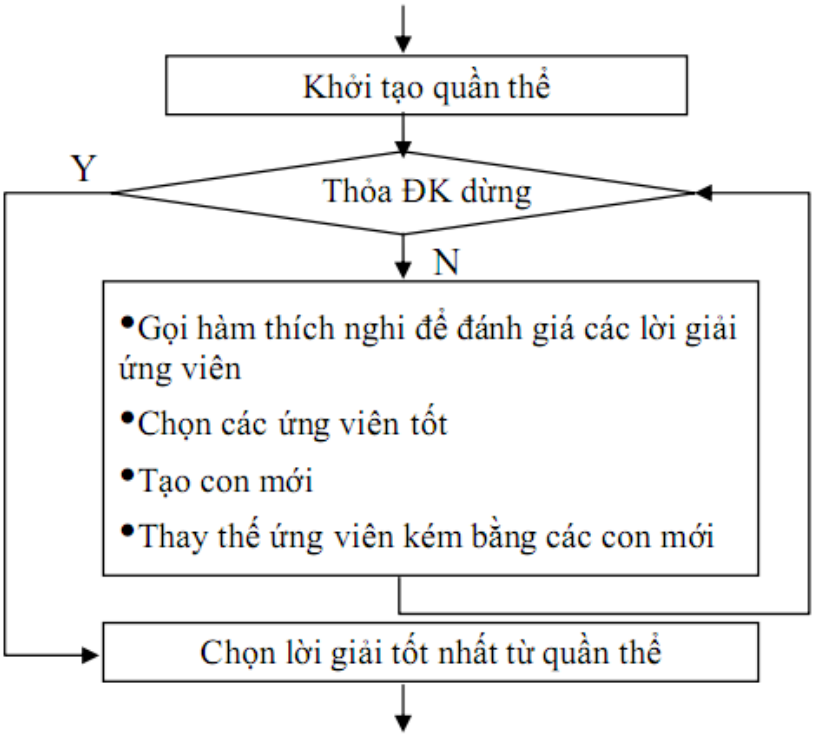
Mặc dù McCulloch-Pitts chứng minh cho sức mạnh của tính toán neural nhưng sức hấp dẫn của tiếp cận này chỉ thực sự bắt đầu khi các giải thuật học phát triển. Phiên bản đầu tiên của mạng neuron có kèm giải thuật học được Frank Rosenblatt đưa ra vào cuối thập niên 1950, có tên gọi là perceptron.

5.2.3 Tiếp cận xã hội và nổi trội

5.2.3.1 Giới thiệu

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.

5.2.3.2 Giải thuật

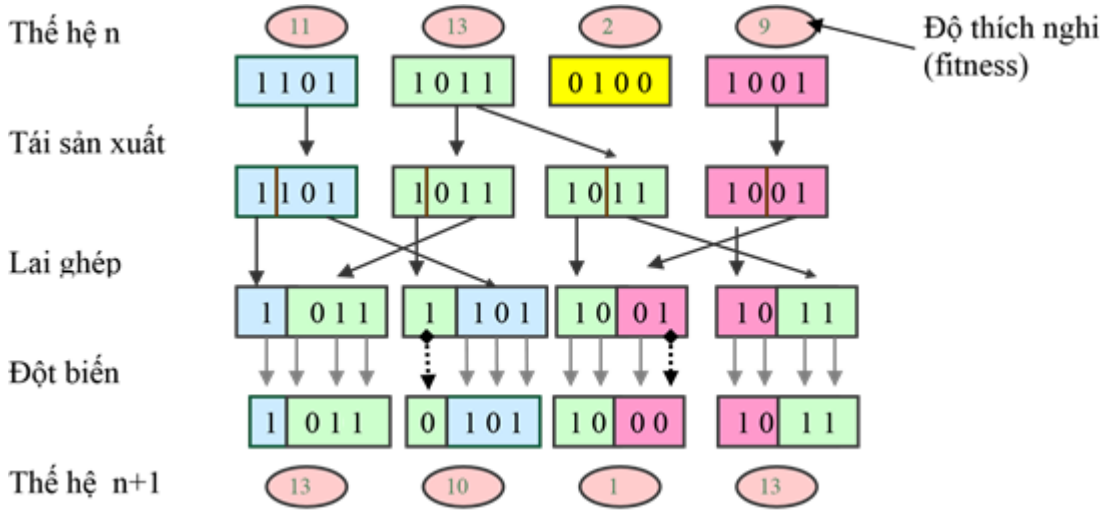


Hình 5.7 - Giải thuật di truyền.

Hình trên 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, nói khác hơn là hình thức biểu diễn một lời giải tiềm năng của bài toán.
- 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 (hình 5.8).
+ 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 (hình 5.8).

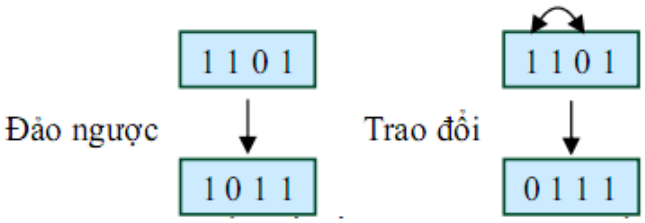


Hình 5.8 - Ví dụ minh họa giải thuật và toán tử di truyền.

Trong ví dụ minh họa bằng hình 5.8, ta thấy tại thế hệ thứ n ta có một lời giải có độ thích nghi rất thấp (2). 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.



Hình 5.9 - Ví dụ minh họa toán tử di truyền đảo ngược và trao đổi.

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?
- …

5.2.3.3 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:

(¬a ∨ c) ∧ (¬a ∨ c ∨ ¬e) ∧ (¬b ∨ c ∨ d ∨ ¬e) ∧ (a ∨ ¬b ∨ c) ∧ (¬e ∨ f)

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. Do đó khi thay các giá trị này vào biểu thức trên 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 trên 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.

5.2.3.4 Bài toán người bán hàng TSP

Bài toán người bán hàng (Traveling Saleman Problem – TSP) là một bài toán cổ điển đối với AI và khoa học máy tính. Như chúng đã biết, 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 9 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. 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, khi đó ta có 1110 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 "|", mỗi điểm cắt này được đưa một cách ngẫu nhiên vào cùng một vị trí trên các mẫu cha mẹ. Những điểm cắt này là ngẫu nhiên, nhưng được đưa vào những vị trí như nhau trên cả cha và mẹ. Ví dụ, có hai mẫu cho mẹ p1 và p2, với các điểm cắt sau thành phố thứ ba và thứ bảy:

p1 = (1 9 2 | 4 6 5 7 | 8 3)

p2 = (4 5 9 | 1 8 7 6 | 2 3)

Hai mẫu con c1 và c2 sẽ được sinh ra theo cách sau. Đầu tiên, các đoạn giữa hai điểm cắt sẽ được chép vào các mẫu con:

c1 = (x x x | 4 6 5 7 | x x)

c2 = (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 c1, thì ta sẽ bắt đầu từ điểm cắt thứ hai của mẫu p2, ta chép các thành phố từ điểm cắt này theo thứ tự vào các chỗ còn trống của c1, bỏ qua những thành phố mà c1 đã có (các ký số được in đậm và nghiêng trong sơ đồ bên dưới). Khi đến cuối mẫu p2, thì quay lại đầu mẫu p2 tiếp tục chép sang c1 cho đến khi c1 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 p1 sang một con c1, trong khi sắp xếp của các thành phố còn lại của con c1 được thừa kế từ cha mẹ kia p2. 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. 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.

5.3 Tổng kết chương

Nội dung chính của chương này bao gồm:

1. Giới thiệu tổng quát về một nhánh nghiên cứu mới của TTNT, đó là máy học. 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.
2. 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.
3. 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.
4. 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.
5. 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.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5

1. Cho một tập hợp các ví dụ rèn luyện như sau:

   a.

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

   b.

TT	MÃ VD	THUỘC TÍNH A1	THUỘC TÍNH A2	THUỘC TÍNH A3	THUỘC TÍNH A4	PHÂN LOẠI
1	VD01	T	T	T	T	+
2	VD02	T	T	T	F	+
3	VD03	T	T	F	T	-
4	VD04	T	F	F	F	-
5	VD05	T	F	T	T	-
6	VD06	T	F	T	F	+
7	VD07	F	T	F	T	-
8	VD08	F	T	F	F	+
9	VD09	F	T	T	T	+
10	VD10	F	F	T	F	-
11	VD11	F	F	F	T	-
12	VD12	F	F	F	F	-

Hãy áp dụng giải thuật ID3 để xây dựng cây quyết định với tập dữ liệu rèn luyện trên.

2. Tìm hiểu, thảo luận về giải thuật học Perceptron và bài toán phân loại.
3. Tìm hiểu, thảo luận về giải thuật học lan truyền ngược và mạng NetTalk.
4. Thảo luận về thuận lợi và khó khăn của giải thuật di truyền.

TÀI LIỆU THAM KHẢO

[1] Trần Ngân Bình, Võ Huỳnh Trâm (2006), *Giáo trình Trí tuệ nhân tạo*, Trường Đại Học Cần Thơ.

[2] Nguyễn Thanh Thủy (2008), *Giáo trình Trí tuệ nhân tạo*, Trường Đại học Bách khoa Hà Nội.

[3] Bùi Xuân Toại, Trương Gia Việt (2000), *Trí tuệ nhân tạo – Các cấu trúc và chiến lược giải quyết vấn đề*, NXB Thống kê.

[4] George F. Luger, William A. Stubblefield, Albuquerque (1997), *Artificial Intelligence*, Wesley Publishing Company.

[5] Larry Medsker (1995), *Hybrid Intelligent Systems*, Kluwer Academic Publishers.

[6] Stuart Russell, Peter Norvig (2009), *Artificial Intelligence: Amodern Approach*, Prentice- Hall.