HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

TỪ MINH PHƯƠNG

GIÁO TRÌNH

Nhập môn trí tuệ nhân tạo

Hà nội 2014

LỜI NÓI ĐẦU

Trí tuệ nhân tạo là một lĩnh vực của khoa học máy tính với mục tiêu nghiên cứu xây dựng và ứng dụng các hệ thống thông minh nhân tạo. Đây là một trong những lĩnh vực được quan tâm nghiên cứu nhiều nhất của khoa học máy tính hiện nay với nhiều kết quả ứng dụng rộng rãi.

Môn học Nhập môn trí tuệ nhân tạo là môn học mang tính chuyên ngành trong chương trình đào tạo công nghệ thông tin hệ đại học. Mục tiêu của môn học nhằm giúp sinh viên làm quen với khái niệm trí tuệ nhân tạo thông qua việc giới thiệu một số kỹ thuật và ứng dụng cụ thể. Với việc học về trí tuệ nhân tạo, một mặt, sinh viên sẽ được làm quen với những phương pháp, cách giải quyết vấn đề không thuộc lĩnh vực toán rời rạc hoặc giải thuật truyền thống, chẳng hạn các phương pháp dựa trên heuristics, các phương pháp dựa trên tri thức, dữ liệu. Mặt khác, sinh viên sẽ được làm quen với khả năng ứng dụng tiềm tàng các kỹ thuật trí tuệ nhân tạo trong nhiều bài toán thực tế.

Do trí tuệ nhân tạo hiện đã phát triển thành một lĩnh vực rộng với khá nhiều lĩnh vực chuyên sâu, việc lựa chọn các nội dung để giới thiệu cho sinh viên là vấn đề không đơn giản. Trong tài liệu này, các nội dung được lựa chọn hoặc là những nội dung có tính tiêu biểu, kinh điển của trí tuệ nhân tạo như biểu diễn tri thức bằng logic, các phương pháp tìm kiếm, hoặc là những kỹ thuật có nhiều ứng dụng và đang có tính thời sự hiện nay, tiêu biểu là phương pháp suy diễn xác suất và các kỹ thuật học máy.

Trong khuôn khổ có hạn của tài liệu với tính chất là giáo trình, phần giới thiệu về việc sử dụng kỹ thuật trí tuệ nhân tạo trong ứng dụng cụ thể không được trình bày nhiều. Chúng tôi dành phần lựa chọn ứng dụng cụ thể cho giảng viên trong quá trình lên lớp và hướng dẫn sinh viên. Tùy điều kiện, giảng viên có thể lựa chọn trong danh mục ứng dụng rất phong phú để giới thiệu và minh họa cho các nội dung của tài liệu.

Nội dung tài liệu được trình bày thành năm chương.

Chương 1 là phần giới thiệu tổng quan về trí tuệ nhân tạo bao gồm khái niệm, lịch sử hình thành, sơ lược về những kỹ thuật và ứng dụng tiêu biểu. Nội dung chương không đi quá sâu vào việc định nghĩa chính xác trí tuệ nhân tạo là gì, thay vào đó, người đọc được giới thiệu về những lĩnh vực nghiên cứu chuyên sâu và lịch sử phát triển, trước khi làm quen với nội dung cụ thể trong các chương sau.

Chương 2 trình bày cách giải quyết vấn đề bằng phương pháp tìm kiếm. Các phương pháp tìm kiếm bao gồm: tìm kiếm mù, tìm kiếm có thông tin, và tìm kiếm cục bộ. Khác với một số tài liệu khác về trí tuệ nhân tạo, nội dung về tìm kiếm có đối thủ không được đề cập đến trong tài liệu này. Một số nội dung như giải thuật di truyền có thể bỏ qua trong phần nhập môn và dùng để tham khảo do tương đối phức tạp so với các kỹ thuật khác.

Chương 3 tóm tắt về vấn đề sử dụng, biểu diễn tri thức và lập luận, trước khi đi sâu trình bày về biểu diễn tri thức và lập luận sử dụng logic. Trong hai hệ thống logic được trình bày là logic mệnh đề và logic vị từ, nội dung chương được dành nhiều hơn cho logic vị từ. Do nội dung về lập trình logic hiện không còn ứng dụng nhiều, chúng tôi không giới thiệu về vấn đề lập trình và xây dựng ứng dụng cụ thể ở đây.

Chương 4 là mở rộng của biểu diễn tri thức và lập luận với việc sử dụng nguyên tắc suy diễn xác suất và mạng Bayes. Sau khi trình bày về sự cần thiết của lập luận trong điều kiện không rõ ràng cùng với nguyên tắc lập luận xác suất, phần chính của chương tập trung vào khái niệm cùng với ứng dụng mạng Bayes trong biểu diễn tri thức và lập luận.

Chương 5 là chương nhập môn về học máy. Trong chương này, người đọc được làm quen với khái niệm, nguyên tắc và ứng dụng của học máy. Trong phạm vi chương cũng trình bày bốn kỹ thuật học máy dùng cho phân loại là cây quyết định, phân loại Bayes, phân loại dựa trên ví dụ và hồi quy logistic, cùng với một số kỹ thuật đánh giá mô hình và lựa chọn đặc trưng. Đây là những phương pháp đơn giản, dễ giới thiệu, thuận tiện để trình bày với tính chất nhập môn. Đồng thời, đây cũng là những phương pháp có tính đại diện trong học máy, cần thiết cho người nghiên cứu về lĩnh vực này. Do ưu điểm và sự phổ biến của Support Vector Machines, phương pháp phân loại này cũng được giới thiệu, nhưng ở mức tóm tắt, không đi vào chi tiết để phù hợp với trình độ nhập môn.

Tài liệu được biên soạn từ kinh nghiệm giảng dạy học phần Nhập môn trí tuệ nhân tạo của tác giả tại Học viện Công nghệ bưu chính viễn thông, trên cơ sở tiếp thu phản hồi từ sinh viên và đồng nghiệp. Tài liệu có thể sử dụng làm tài liệu học tập cho sinh viên đại học ngành công nghệ thông tin và các ngành liên quan, ngoài ra có thể sử dụng với mục đích tham khảo cho những người quan tâm tới trí tuệ nhân tạo.

Trong quá trình biên soạn tài liệu, mặc dù tác giả đã có nhiều cố gắng song không thể tránh khỏi những thiếu sót. Ngoài ra, trí tuệ nhân tạo một lĩnh vực rộng, đang tiến bộ rất nhanh của khoa học máy tính đòi hỏi tài liệu phải được cập nhật thường xuyên. Tác giả rất mong muốn nhận được ý kiến phản hồi, góp ý cho các thiếu sót cũng như ý kiến về việc cập nhật, hoàn thiện nội dung của tài liệu.

Hà nội 12/2014

Tác giả

Mục lục

CHƯƠNG 1: GIỚI THIỆU CHUNG

1.1. KHÁI NIỆM TRÍ TUỆ NHÂN TẠO

Trí tuệ nhân tạo (TTNT) là một lĩnh vực nghiên cứu của khoa học máy tính và khoa học tính toán nói chung. Có nhiều quan điểm khác nhau về trí tuệ nhân tạo và do vậy có nhiều định nghĩa khác nhau về lĩnh vực khoa học này.

Mục đích của trí tuệ nhân tạo là xây dựng các *thực thể thông minh*. Tuy nhiên, do rất khó định nghĩa thế nào là thực thể thông minh nên cũng khó thống nhất định nghĩa trí tuệ nhân tạo. Theo một tài liệu được sử dụng rộng rãi trong giảng dạy trí tuệ nhân tạo hiện nay, các định nghĩa có thể nhóm thành bốn nhóm khác nhau, theo đó, trí tuệ nhân tạo là lĩnh vực nghiên cứu việc xây dựng các hệ thống máy tính có đặc điểm sau:

1) Hệ thống hành động như người.
2) Hệ thống có thể suy nghĩ như người
3) Hệ thống có thể suy nghĩ hợp lý
4) Hệ thống hành động hợp lý

Trong số các định nghĩa trên, nhóm thứ hai và ba quan tâm tới quá trình suy nghĩ và tư duy, trong khi nhóm thứ nhất và thứ tư quan tâm chủ yếu tới hành vi. Ngoài ra, hai nhóm định nghĩa đầu xác định mức độ thông minh hay mức độ trí tuệ bằng cách so sánh với khả năng suy nghĩ và hành động của con người, trong khi hai nhóm định nghĩa sau dựa trên khái niệm suy nghĩ hợp lý và hành động hợp lý. Trong phần phân tích bên dưới ta sẽ thấy suy nghĩ và hành động hợp lý khác với suy nghĩ và hành động như người thế nào.

Sau đây ta sẽ xem xét cụ thể các nhóm định nghĩa trên.

1) Hành động như người

Do con người được coi là động vật có trí tuệ, nên một cách rất tự nhiên là lấy con người làm thước đo khi đánh giá mức độ thông minh của máy tính.

Theo cách định nghĩa này, trí tuệ nhân tạo nhằm tạo ra các hệ thống có hành vi hay hành động tương tự con người, đặc biệt trong những hoạt động có liên quan tới trí tuệ. Để xác định thế nào là hành động như người, có thể sử dụng phép thử Turing.

Phép thử Turing (Turing test): Vào năm 1950, Alan Turing – nhà toán học người Anh có nhiều đóng góp cho khoa học máy tính – đã xây dựng thủ tục cho phép định nghĩa trí tuệ. Thủ tục này sau đó được gọi là phép thử Turing (Turing test), và được thực hiện như sau. Hệ thống được gọi là thông minh, hay có trí tuệ nếu hệ thống có thể hành động tương tự con người trong các công việc đòi hỏi trí tuệ. Trong quá trình thử, một người kiểm tra sẽ đặt các câu hỏi (dưới dạng văn bản) và nhận câu trả lời cũng dưới dạng văn bản từ hệ thống, tương tự khi ta chat hay nhắn tin. Nếu người kiểm tra không phân biệt được câu trả lời là do người thật trả lời hay do máy sinh ra thì hệ thống qua được phép thử và được gọi là có trí tuệ.

Cần lưu ý rằng, phép thử Turing nguyên bản không đòi hỏi có sự tiếp xúc vật lý trực tiếp giữa người kiểm tra và hệ thống bị kiểm tra, do việc tạo ra hệ thống người nhân tạo một cách vật lý được coi là không liên quan tới trí tuệ.

Để qua được phép thử Turing, hệ thống cần có những khả năng sau:

- *Xử lý ngôn ngữ tự nhiên*: để có thể phân tích, hiểu câu hỏi và tổng hợp câu trả lời trên một ngôn ngữ giao tiếp thông thường như tiếng Việt hay tiếng Anh.
- *Biểu diễn tri thức*: phục vụ việc lưu tri thức và thông tin trong hệ thống.
- *Suy diễn*: sử dụng tri thức để trả lời câu hỏi.
- *Học máy*: để có thể thích nghi với hoàn cảnh và học những mẫu trả lời.

Trong lịch sử trí tuệ nhân tạo đã có những hệ thống như ELIZA được xây dựng nhằm mục đích vượt qua phép thử Turing mà không cần đầy đủ tới cả bốn khả năng trên.

Mặc dù không nhiều người coi mục đích chính của trí tuệ nhân tạo là vượt qua phép thử Turing, một số hệ thống đã xây dựng chuyên cho mục đích này. Gần đây nhất, vào tháng 6 năm 2014, một hệ thống chat tự động có tên là Eugene Goostman do một nhóm nghiên cứu người Nga xây dựng đã giành giải nhất trong cuộc thi phép thử Turing. Sau khi thực hiện một đoạn hội thoại dài 5 phút với hệ thống, 33% giám khảo cho rằng đó là người thực. Một số ý kiến cho rằng Eugene Goostman là hệ thống máy tính đầu tiên vượt qua phép thử Turing.

2) Suy nghĩ như người

Theo nhóm định nghĩa này, hành động thông minh chỉ đạt được nếu được dẫn dắt bởi quá trình suy nghĩ tương tự quá trình suy nghĩ của con người.

Những nghiên cứu theo hướng này dựa trên việc nghiên cứu quá trình nhận thức và tư duy của con người, từ đây mô hình hóa và tạo ra những hệ thống có mô hình nhận thức, tư duy tương tự. Việc tìm hiểu quá trình nhận thức, tư duy của người có thể thực hiện theo một số phương pháp như: 1) thực nghiệm về hành vi con người khi suy nghĩ hoặc giải quyết vấn đề; 2) chụp ảnh sóng não, đo tín hiệu điện từ hoặc các tín hiệu khác của não trong quá trình thực hiện các công việc khác nhau; 3) sử dụng các phương pháp nơ ron sinh học khác như kích thích não, giải phẫu não v.v.

Một hệ thống trí tuệ nhân tạo dạng này là hệ thống GPS, viết tắt của General Problem Solver do Newell và Simon trình diễn năm 1961. GPS là chương trình máy tính cho phép giải quyết các bài toán bằng cách mô phỏng chuỗi suy nghĩ của con người khi giải quyết những bài toán như vậy.

Hiện nay, hướng nghiên cứu này được thực hiện trong khuôn khổ *khoa học nhận thức* (cognitive science). Đây là lĩnh vực khoa học liên ngành, kết hợp các mô hình máy tính với phương pháp thực nghiệm tâm lý. Nhiều kết quả nghiên cứu về nhận thức đã được áp dụng trong các mô hình tính toán. Ví dụ, nhiều nghiên cứu về quá trình tiếp nhận tín hiệu ảnh và nhận dạng đối tượng đã được áp dụng trong lĩnh vực thị giác máy. Hay, gần đây, một số nghiên cứu về việc thiết kế các vi mạch có cấu trúc dựa trên hệ thần kinh của người (neuromorphic chips) đã cho kết quả tốt trong các bài toán học máy hoặc xử lý lượng khối lượng dữ liệu lớn.

3) Suy nghĩ hợp lý

Thực tế cho thấy con người bị chi phối bởi tâm lý, cảm xúc. Do vậy, không phải lúc nào con người cũng suy nghĩ và hành động theo hướng đạt tới kết quả tốt. Từ đây xuất hiện cách tiếp cận theo hướng xây dựng các hệ thống cho phép đạt tới kết quả tốt mà không cần học

theo con người. Cách tiếp cận này được gọi là suy nghĩ hợp lý và hành động hợp lý. Trước hết là suy nghĩ hợp lý.

Một cách tiếp cận tiêu biểu của suy nghĩ hợp lý là xây dựng những hệ thống có khả năng lập luận dựa trên việc sử dụng các hệ thống hình thức như logic. Tiền thân của cách tiếp cận này có gốc rễ từ triết học Hy lạp do Aristot khởi xướng. Cơ sở chủ yếu là sử dụng logic để biểu diễn bài toán và giải quyết bằng suy diễn logic. Một số hệ thống logic cho phép biểu diễn mọi loại đối tượng và quan hệ giữa các đối tượng đó. Sau khi đã biểu diễn dưới dạng logic, có thể xây dựng chương trình để giải quyết các bài toán về suy diễn và lập luận.

Khó khăn chủ yếu của cách tiếp cận này là việc mô tả hay biểu diện bài toán dưới dạng các cấu trúc logic để có thể giải quyết được. Trên thực tế, tri thức và thông tin về bài toán thường có yếu tố không đầy đủ, không chính xác. Ngoài ra, việc suy diễn logic đòi hỏi khối lượng tính toán lớn khi sử dụng trong thực tế và rất khó để triển khai cho các bài toán thực.

4) Hành động hợp lý

Cách tiếp cận này tập trung vào việc xây dựng các tác tử (agent) có khả năng hành động hợp lý, tức là hành động để đem lại kết quả tốt nhất hoặc kết quả kỳ vọng tốt nhất khi có yếu tố không chắc chắn. Cần lưu ý rằng, hành động hợp lý có thể khác với hành động giống con người: con người không phải lúc nào cũng hành động hợp lý do bị chi phối bởi các yếu tố chủ quan.

Một đặc điểm quan trọng của hành động hợp lý là hành động kiểu này có thể dựa trên việc suy nghĩ (suy luận) hợp lý hoặc không. Trong một số trường hợp, để quyết định hành động thế nào, cần dựa trên việc suy luận hợp lý. Tuy nhiên, trong nhiều tình huống, việc hành động theo phản xạ, chẳng hạn khi gặp nguy hiểm, không đòi hỏi suy diễn phức tạp, nhưng lại cho kết quả tốt hơn. Các hệ thống hành động hợp lý có thể sử dụng cả hai cách tiếp cận dựa trên suy diễn và dựa trên phản xạ để đạt được kết quả tốt.

Hệ thống có khả năng hành động hợp lý có thể bao gồm suy diễn hoặc không, có thể dựa trên cách suy nghĩ giống người hoặc không, có thể bao gồm cả các kỹ thuật dùng để vượt qua phép thử Turing. Do vậy, cách tiếp cận này được coi là tổng quát và bao gồm các cách tiếp cận khác. *Hiện có nhiều ý kiến coi hệ thống trí tuệ nhân tạo là các hệ thống dạng này*.

Tóm tắt

Các phân tích ở trên cho thấy một số cách tiếp cận chính trong định nghĩa trí tuệ nhân tạo:

- Lấy con người làm tiêu chuẩn, nghiên cứu tâm lý và thần kinh học để mô phỏng nhận thức con người, dựa trên đó xây dựng hệ thống trí tuệ nhân tạo.
- Lấy kết quả làm tiêu chuẩn, không nhất thiết phải xây dựng hệ thống mô phỏng người.
- Lấy hành vi và hành động làm mục đích, có thể có quá trình lập luận để hướng dẫn hành động hoặc không.

1.2. LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN

Lịch sử hình thành và phát triển trí tuệ nhân tạo có thể chia thành một số giai đoạn sau (các giai đoạn được chia theo mức độ phát triển và có thể giao nhau về thời gian):

a. Giai đoạn tiền khởi đầu (1943-1955)

Mặc dù chưa có khái niệm chính thức về trí tuệ nhân tạo, giai đoạn này ghi nhận một số kết quả có liên quan trực tiếp tới nghiên cứu về trí tuệ nhân tạo sau này:

- Năm 1943, Warren McCulloch và Walter Pitts mô tả mô hình mạng nơ ron nhân tạo đầu tiên, và cho thấy mạng nơ ron nhân tạo có khả năng biểu diễn nhiều hàm số toán học.
- Năm 1950, Alan Turing công bố bài báo nhắc tới trí tuệ máy, trong đó lần đầu tiên mô tả khái niệm phép thử Turing, học máy, thuật toán di truyền, và học tăng cường.
- Năm 1956 được coi là năm chính thức ra đời của khái niệm trí tuệ nhân tạo. Mười nhà nghiên cứu trẻ đã tổ chức một cuộc hội thảo kéo dài hai tháng tại trường đạt học Dartmouth với mục đích đặt nền móng đầu tiên cùng với tên gọi chính thức của trí tuệ nhân tạo: "artificial intelligence". Đa số những người tham gia hội thảo này, bao gồm John McCarthy, Marvin Minsky, Allen Newell, và Herbert Simon, sau đó đã trở thành những chuyên gia tiên phong trong các nghiên cứu về trí tuệ nhân tạo. Điểm quan trọng nhất của hội thảo này là đưa ra một số đề xuất và hình dung về trí tuệ nhân tạo. Các đề xuất này vượt ra ngoài khuôn khổ các lĩnh vực nghiên cứu đã hình thành trước đó như lý thuyết điều khiển, vận trù học, lý thuyết ra quyết định. Chính các đề xuất mới này đã đưa trí tuệ nhân tạo thành một lĩnh vực khoa học mới với đối tượng và phương pháp nghiên cứu riêng của mình.

b. Giai đoạn khởi đầu (1952-1969)

Đây là giai đoạn với nhiều thành tích nhất định của các nghiên cứu trí tuệ nhân tạo, được thể hiện qua một số ví dụ sau:

- Các chương trình Logic Theorist và sau đó là General Problem Solver (GPS) của Newell và Simon, có khả năng chứng minh định lý toán học theo cách tương tự tư duy của con người. Chẳng hạn, trong lớp bài toán mà GPS có thể giải quyết, việc chia bài toán thành các bài toán con và thứ tự các bước giải được tiến hành tương tự với con người khi giải quyết cùng bài toán. Chương trình Logic Theorist đã chứng minh được 38 trong số 52 định lý từ một sách giáo khoa toán, trong đó có định lý về tam giác cân được chứng minh theo cách ngắn hơn cách truyền thống.
- Năm 1952, Arthur Samuel xây dựng một số chương trình chơi cờ đam (checkers). Chương trình có khả năng học và đánh thắng các đối thủ là người chơi cờ đam nghiệp dư. Điểm đặc biệt của chương trình này là khả năng tự học từ kinh nghiệm. Nhờ khả năng học, chương trình có thể thắng cả người đã tạo ra nó.
- Năm 1958, John McCarthy đề xuất ngôn ngữ Lisp, sau này trở thành một trong hai ngôn ngữ thông dụng nhất của trí tuệ nhân tạo.
- Cũng trong những năm này, Minsky khởi xướng việc giải quyết những vấn đề có miền giới hạn hẹp và cần tới tri thức khi giải quyết. Các bài toán có miền hẹp như vậy được gọi là *thế giới nhỏ* (microworld). Chẳng hạn, trong lĩnh vực hẹp về giải tích, chương trình SAINT do James Slagle viết năm 1963 có thể giải các bài toán tích phân ở mức độ sinh viên năm thứ nhất đại học.

- Mạng nơ ron nhân tạo tiếp tục tiếp tục được phát triển với một số phát minh mới như mạng adalines của Bernie Widrow (1962), Perceptron của Rosenblatt (1962), cho phép giải quyết nhiều bài toán học máy. Trong năm 1962, các nghiên cứu đã chứng minh khả năng học của mạng nơ ron, theo đó có thể thay đổi trọng số kết nối của các nơ ron để phù hợp với bất cứ thong tin đầu vào nào.
- Năm 1965, John Alan Robinson phát minh ra cách chứng minh và suy diễn bằng cách sử dụng phép giải cho logic vị từ. Đây là phương pháp quan trọng, cho phép chương trình máy tính thực hiện lập luận và suy diễn tự động một cách hiệu quả trong trường hợp tri thức được biểu diễn bằng logic.

c. Một số khó khăn và giai đoạn trầm lắng

Sau một số thành công ban đầu, đã có những dự báo lạc quan về khả năng xây dựng các hệ thống thông minh trong tương lai gần. Một số nhà khoa học dự báo về khả năng tạo ra các hệ thống thông minh trong vòng 10 tới vài chục năm. Tuy nhiên, sau giai đoạn phát triển mạnh lúc đầu, vào đầu những năm bẩy mươi, các nhà khoa học bắt đầu nhận ra một số khó khăn, đòi hỏi cách tiếp cận thực tế hơn khi nghiên cứu trí tuệ nhân tạo.

Thứ nhất, cách tiếp cận thời kỳ đầu thường sử dụng các biến đổi cú pháp đơn giản và *không quan tâm tới tri thức* về bài toán cần giải quyết. Ví dụ, khi xây dựng các hệ thống dịch máy, nhiều người kỳ vọng có nếu có từ điển và biết cách lắp ghép các từ để tạo thành câu là có thể dịch tự động. Trong khi đó trên thực tế, để dịch được, người dịch cần có hiểu biết nhất định về lĩnh vực được đề cập đến để thể hiện lại nội dung cần dịch trên ngôn ngữ đích. Các dự án dịch tự động từ tiếng Nga sang tiếng Anh do Bộ quốc phòng Mỹ tài trợ đã thất bại do cách tiếp cận đơn giản lúc đầu.

Thứ hai, nhiều hệ thống trí tuệ nhân tạo thời kỳ đầu sử dụng việc tìm kiếm các hành động dẫn tới lời giải. Với bài toán kích thước nhỏ, các kỹ thuật tìm kiếm đơn giản cho kết quả tốt. Tuy nhiên, khi kích thước bài toán tăng lên, số tổ hợp cần xem xét tăng nhanh, vượt khả năng xử lý của máy tính. Hiệu ứng này được gọi là sự "*bùng nổ tổ hợp*" và chỉ được quan tâm đúng mức sau khi lý thuyết về độ phức tạp tính toán ra đời.

Do các khó khăn nói trên, một số báo cáo bi quan về triển vọng trí tuệ nhân tạo đã được trình lên chính phủ các nước như Mỹ, Anh, dẫn tới việc các chính phủ ngừng cấp kinh phí nghiên cứu cho lĩnh vực này. Đây cũng là giai đoạn khó khăn trong lịch sử phát triển trí tuệ nhân tạo, kéo dài trong khoảng 1974 – 1980. Giai đoạn này được gọi là *mùa đông trí tuệ nhân tạo* (AI winter), lấy nguyên mẫu từ khái niệm mùa đông hạt nhân, là kết quả mô phỏng khí hậu trái đất lạnh lẽo sau khi xảy ra chiến tranh hạt nhân.

Một giai đoạn thứ hai cũng được gọi là mùa đông trí tuệ nhân tạo là giai đoạn 1987-1993. Giai đoạn trì trệ này gắn với sự đi xuống của thị trường các hệ chuyên gia và thất bại của dự án máy tính thế hệ năm do chính phủ Nhật Bản tài trợ.

d. Hệ thống dựa trên tri thức (1969-1979)

Các chương trình trí tuệ nhân tạo xây dựng trong giai đoạn trước có một số hạn chế do không có tri thức về lĩnh vực liên quan, và do vậy không thể giải quyết những bài toán khó, đòi hỏi khối lượng tính toán lớn hoặc nhiều tri thức chuyên sâu. Để khắc phục, giai đoạn này chú trọng tới việc sử dụng nhiều tri thức, thông tin đặc thù cho lĩnh vực hẹp của vấn đề cần giải quyết. Điển hình của hệ thống dựa trên tri thức là các **hệ chuyên gia** (expert systems).

Hệ chuyên gia là các hệ thống có khả năng ra quyết định tương tự chuyên gia trong lĩnh vực hẹp của mình. Hệ thống loại này được xây dựng để giải quyết những vấn đề phức tạp bằng cách lập luận trên tri thức nhận được từ các chuyên gia. Chẳng hạn, một bác sĩ chuyên khoa giỏi có thể mô tả các quy tắc chẩn đoán bệnh trong chuyên khoa của mình. Các quy tắc đó chính là tri thức cần thiết khi chẩn đoán bệnh. Thông thường, tri thức được biểu diễn dưới dạng các luật "*Nếu…Thì…*". Hệ chuyên gia thường gồm *cơ sở tri thức* chứa các luật như vậy và *mô tơ suy diễn* giúp tìm ra lời giải từ tri thức và thông tin về trường hợp đang có.

Sau đây là ví dụ một số hệ thống như vậy:

- DENDRAL (năm 1967) là chương trình hệ chuyên gia xây dựng tại trường Stanford, cho phép dự đoán cấu trúc phân tử hữu cơ. Chương trình làm việc dựa trên các luật do chuyên gia trong lĩnh vực hóa học và vật lý cung cấp.
- Một trong các tác giả của DENDRAL, sau đó đã cùng với cộng sự xây dựng MYCIN (1974), hệ chuyên gia cho phép chẩn đoán bệnh nhiễm trùng máu. Với khoảng 450 luật do chuyên gia cung cấp, hệ thống có chất lượng chẩn đoán tương đương bác sĩ giỏi trong lĩnh vực này.
- Việc sử dụng tri thức cũng được sử dụng để giải quyết vấn đề hiểu ngôn ngữ tự nhiên, ví dụ trong hệ thống dịch tự động.

Hệ chuyên gia cho phép giải quyết một phần hạn chế của các hệ thống đơn giản không dựa trên tri thức trước đó. Một số hệ chuyên gia đã được thương mại hoá và đem lại doanh thu cho lĩnh vực trí tuệ nhân tạo.

Cũng trong giai đoạn này, vào năm 1972 Alain Colmerauer đã phát triển ngôn ngữ **Prolog** (viết tắt của logic programming tức là lập trình logic) phục vụ việc biểu diễn tri thức dưới dạng tương tự logic vị từ và lập luận trên tri thức đó. Prolog cùng với Lisp trở thành hai ngôn ngữ được dùng nhiều nhất trong trí tuệ nhân tạo.

e. Trí tuệ nhân tạo có sản phẩm thương mại (1980 đến nay)

Sau thành công của những hệ chuyên gia đầu tiên, việc xây dựng hệ chuyên gia được thương mại hóa từ năm 1980 và đặc biệt phát triển cho tới 1988. Sau giai đoạn này, do một số hạn chế của hệ chuyên gia, trí tuệ nhân tạo rơi vào một giai đoạn trì trệ, không có những bước tiến đáng kể (mùa đông trí tuệ nhân tạo thứ hai).

Giai đoạn này cũng đánh dấu sự trở lại của mạng nơ ron nhân tạo sau một thời gian không có các phát minh và ứng dụng đáng kể. Cho đến hiện nay, mạng nơ ron nhân tạo vẫn được sử dụng tương đối nhiều cho học máy và như các chương trình nhận dạng, phân loại tự động. Đặc biệt, trong khoảng gần 10 năm gần đây, các mạng nơ ron nhân tạo nhiều lớp được gọi là *mạng sâu* (deep network) đang được đặc biệt quan tâm do có độ chính xác rất tốt trong các ứng dụng nhận dạng âm thanh, hình ảnh, xử lý ngôn ngữ tự nhiên.

Vào năm 1981, chính phủ Nhật Bản khởi động chương trình xây dựng máy tính thế hệ 5. Mục đích của chương trình là xây dựng các máy tính thông minh chạy trên ngôn ngữ Prolog. Chính phủ Mỹ và Anh cũng có những dự án tương tự nhưng quy mô nhỏ hơn. Mặc dù các chương trình này đều không đạt được mục tiêu đề ra những đã giúp chấm dứt mùa đông trí tuệ nhân tạo lần thứ nhất.

f. Trí tuệ nhân tạo chính thức trở thành ngành khoa học (1987 đến nay)

Trong giai đoạn trước, chủ đề nghiên cứu chủ yếu có tính thăm dò và tìm kiếm định hướng. Các phương pháp nghiên cứu về trí tuệ nhân tạo cũng không bị giới hạn nhiều trong các lý thuyết có sẵn. Nhiều chủ đề nghiên cứu được đề xuất hoàn toàn mới và phương pháp giải quyết cũng tương đối tự do, không dựa trên các lý thuyết hay kết quả khoa học đã có. Mục tiêu của trí tuệ nhân tạo giai đoạn đầu là vượt ra ngoài khuôn khổ các lĩnh vực nghiên cứu đã có như lý thuyết điều khiển hay thống kê, do vậy nhiều nghiên cứu không đòi hỏi phải dựa trên cơ sở lý thuyết đã phát triển trong các lĩnh vực này.

Bắt đầu từ giai đoạn này (1987), trí tuệ nhân tạo đã có phương pháp nghiên cứu riêng của mình, tuân theo các yêu cầu chung đối với phương pháp nghiên cứu khoa học. Chẳng hạn, kết quả cần chứng minh bằng thực nghiệm, và được phân tích kỹ lưỡng bằng khoa học thống kê. Các vấn đề nghiên cứu cũng gắn nhiều với ứng dụng và đời sống, thay vì những ví dụ mang tính minh hoạ như trong các giai đoạn trước.

Nhiều phát minh trước đây của trí tuệ nhân tạo như mạng nơ ron nhân tạo được phân tích và so sánh kỹ càng với những kỹ thuật khác của thống kê, nhận dạng, và học máy. Nhờ vậy, các phương pháp không còn mang tính kinh nghiệm thuần túy mà đều dựa trên các cơ sở lý thuyết rõ ràng hơn. Tương tự như vậy, một ví dụ khác là cách tiếp cận với bài toán nhận dạng tiếng nói. Trong giai đoạn đầu, bài toán này được tiếp cận theo một số cách tương đối tự do, hướng vào việc giải quyết một số trường hợp riêng với phạm vi hạn chế, và không dựa trên các lý thuyết đã có. Hiện nay, đa số giải pháp nhận dạng tiếng nói được xây dựng trên các mô hình thống kê như mô hình Markov ẩn (Hidden Markov Models), hay các mạng nơ ron nhiều lớp. Hiệu quả của các giải pháp này có thể giải thích dựa trên lý thuyết thống kê, học máy và các phương pháp tối ưu đã tồn tại và được kiểm chứng từ lâu.

g. Cách tiếp cận dựa trên dữ liệu, sử dụng khối lượng dữ liệu lớn (2001 đến nay)

Trong các giai đoạn trước, việc phát triển trí tuệ nhân tạo chủ yếu tập trung vào xây dựng thuật toán và các hệ thống dựa trên tri thức chuyên gia. Gần đây, do sự xuất hiện của Internet, thương mại điện tử, và một số lĩnh vực khoa học như sinh học phân tử, lượng dữ liệu số hóa được tạo ra tăng rất nhanh. Nhiều nghiên cứu cũng cho thấy việc sử dụng dữ liệu hợp lý quan trọng hơn việc xây dựng các thuật toán phức tạp. Một trong những ví dụ là tiến bộ của hệ thống dịch tự động của Google, được xây dựng dựa trên việc thống kê số lượng lớn các văn bản đơn ngữ và song ngữ, thay vì sử dụng luật và quy tắc ngữ pháp như trước đây.

Trong vài năm gần đây, xu hướng sử dụng *dữ liệu lớn* (big data), tức là các kỹ thuật ra quyết định dựa trên việc phân tích lượng lớn dữ liệu với bản chất đa dạng và thay đổi nhanh theo thời gian đang được coi là một trong những xu hướng có ảnh hưởng quan trọng tới sự phát triển và ứng dụng của trí tuệ nhân tạo. Nhiều ứng dụng quan trọng dựa trên dữ liệu lớn như các hệ thống hỗ trợ khuyến nghị của Amazon, ứng dụng trợ giúp Siri của Apple, chương trình dịch tự động và nhận dạng giọng nó của Google, chương trình trò chơi Watson của IBM là những ví dụ thành công điển hình của xu hướng này.

Thành công của việc sử dụng dữ liệu lớn cho thấy có thể giải quyết một vấn đề quan trọng của trí tuệ nhân tạo: đó là việc thu thập đủ lượng tri thức cần thiết cho hệ thống. Thay vì thu thập bằng tay như trước đây, tri thức có thể được tổng hợp, hay "học" từ dữ liệu. Đây cũng là hứa hẹn cho một giai đoạn phát triển mạnh các ứng dụng của trí tuệ nhân tạo.

1.3. CÁC LĨNH VỰC NGHIÊN CỨU VÀ ỨNG DỤNG CHÍNH

1.3.1. Các lĩnh vực nghiên cứu

Trí tuệ nhân tạo được chia thành một số lĩnh vực nghiên cứu nhỏ hơn và chuyên sâu nhằm giải quyết những vấn đề khác nhau khi xây dựng một hệ thống trí tuệ nhân tạo. Một số lĩnh vực chuyên sâu được hình thành để giải quyết một lớp bài toán. Một số lĩnh vực chuyên sâu khác tập trung vào các hướng tiếp cận hay các kỹ thuật. Một số lĩnh vực nghiên cứu lại xoay quanh các ứng dụng cụ thể. Trong khi nhiều lĩnh vực nghiên cứu nhỏ có liên quan mật thiết đến nhau thì có nhiều lĩnh vực khác rất xa nhau, cả về mục tiêu, phương pháp và cộng đồng nghiên cứu.

Thông thường, một hệ thống trí tuệ nhân tạo hoàn chỉnh, làm việc trong việc một môi trường nào đó cần có khả năng: *cảm nhận* (perception), *lập luận* (reasoning), và *hành động* (action). Dưới đây là một số lĩnh vực nghiên cứu của trí tuệ nhân tạo được phân chia theo ba thành phần này.

a) Cảm nhận

Hệ thống cần có cơ chế thu nhận thông tin liên quan tới hoạt động từ môi trường bên ngoài. Đó có thể là camera, cảm biến âm thanh (microphone), cảm biến siêu âm, radar, cảm biến gia tốc, các cảm biến khác. Đó cũng có thể đơn giản hơn là thông tin do người dùng nhập vào chương trình bằng tay. Để biến đổi thông tin nhận được về dạng có thể hiểu được, thông tin cần được xử lý nhờ những kỹ thuật được nghiên cứu và trong khuôn khổ các lĩnh vực sau.

Thị giác máy (computer vision)

Đây là lĩnh vực thuộc trí tuệ nhân tạo có mục đích nghiên cứu về việc thu nhận, xử lý, phân tích, nhận dạng thông tin hình ảnh thu được từ các cảm biến hình ảnh như camera. Mục đích của thị giác máy là biến thông tin thu được thành biểu diễn mức cao hơn để máy tính sau đó có thể hiểu được, chẳng hạn từ ảnh chụp văn bản cần trả về mã UNICODE của các chữ in trên văn bản đó. Biểu diễn ở mức cao hơn của thông tin từ cảm biến hình ảnh sau đó có thể sử dụng để phục vụ quá trình ra quyết định. Thị giác máy tính bao gồm một số bài toán chính sau: *nhận dạng mẫu* (pattern recognition), *phân tích chuyển động* (motion analysis), *tạo lập khung cảnh 3D* (scene reconstruction), *nâng cao chất lượng ảnh* (image restoration).

Nhận dạng mẫu là lĩnh vực nghiên cứu lớn nhất trong phạm vi thị giác máy. Bản thân nhận dạng mẫu được chia thành nhiều bài toán nhỏ và đặc thù hơn như bài toán nhận dạng đối tượng nói chung, nhận dạng các lớp đối tượng cụ thể như *nhận dạng mặt người*, *nhận dạng vân tay*, *nhận dạng chữ viết* tay hoặc chữ in. Nhận dạng đối tượng là phát hiện ra đối tượng trong ảnh hoặc video và xác định đó là đối tượng nào. Trong khi con người có thể thực hiện việc này tương đối đơn giản thì việc nhận dạng tự động thường khó hơn nhiều. Hiện máy tính chỉ có khả năng nhận dạng một số lớp đối tượng nhất định như chữ in, mặt người nhìn thẳng, với độ chính xác gần với con người.

Xử lý ngôn ngữ tự nhiên (natural language processing)

Đây là lĩnh vực nghiên cứu với có mục đích phân tích thông tin, dữ liệu nhận được dưới dạng âm thanh hoặc văn bản và được trình bày dưới dạng ngôn ngữ tự nhiên của con người. Chẳng hạn, thay vì gõ các lệnh quy ước, ta có thể ra lệnh bằng cách nói với máy tính như với

người thường. Do đối tượng giao tiếp của hệ thống trị tuệ nhân tạo thường là con người, khả năng tiếp nhận thông tin và phản hồi dưới dạng lời nói hoặc văn bản theo cách diễn đạt của người sẽ rất có ích trong những trường hợp như vậy.

Xử lý ngôn ngữ tự nhiên bao gồm ba giai đoạn chính: **nhận dạng tiếng nói** (speech recognition), xử lý thông tin đã được biểu diễn dưới dạng văn bản, và biến đổi từ **văn bản thành tiếng nói** (text to speech).

Nhận dạng tiếng nói là quá trình biến đổi từ tín hiệu âm thanh của lời nói thành văn bản. Nhận dạng tiếng nói còn có các tên gọi như nhận dạng tiếng nói tự động (automatic speech recognition) hay nhận dạng tiếng nói bằng máy tính, hay biến đổi *tiếng nói thành văn bản* (speech to text – STT). Nhận dạng tiếng nói được thực hiện bằng cách kết hợp kỹ thuật xử lý tín hiệu âm thanh với kỹ thuật nhận dạng mẫu, chẳng hạn bằng cách sử dụng các mô hình thống kê và học máy.

Xử lý thông tin văn bản được diễn đạt bằng ngôn ngữ tự nhiên như tiếng Việt hay tiếng Anh bao gồm một số bài toán và ứng dụng chính sau:

- *Phân tích từ loại, ngữ pháp*. Nhận đầu vào là một câu, trả về từ loại (động từ, tính từ v.v.) của các từ trong câu; xây dựng cây cú pháp của câu đó tức là xác định các thành phần như chủ ngữ, vị ngữ và quan hệ giữa các thành phần. Do ngôn ngữ tự nhiên thường có tính nhập nhằng nên từ một câu có thể có nhiều cách phân tích. Đây là bài toán cơ sở cho các bài toán xử lý ngô ngữ tự nhiên khác.
- *Hiểu ngôn ngữ tự nhiên hay phân tích ngữ nghĩa*. Biến đổi các câu trên ngôn ngữ tự nhiên thành các biểu diễn như biểu thức trên logic vị từ sao cho máy tính có thể thực hiện biến đổi hoặc lập luận trên đó. Các biểu diễn này cần tương ứng với ngữ nghĩa trong thế giới thực của bài toán.
- *Dịch tự động hay dịch máy*. Tự động biến đổi các văn bản trên ngôn ngữ tự nhiên này sang ngôn ngữ tự nhiên khác, ví dụ từ tiếng Việt sang tiếng Anh và ngược lại. Đây là bài toán có tính ứng dụng cao nhưng là bài toán khó do đòi hỏi khả năng thực hiện các bài toán xử lý ngôn ngữ tự nhiên khác cộng với khả năng sử dụng tri thức về các lĩnh vực liên quan tới nội dung văn bản cần dịch.
- *Trả lời tự động* (question answering). Tự động sinh ra câu trả lời cho các câu hỏi của con người. Ví dụ hệ thống dạng này là chương trình Watson của IBM cho phép trả lời các câu hỏi trong trò chơi Jeopardy tương tự trò Ai là triệu phú, hay trợ lý ảo Siri của iPhone. Hệ thống dạng này thường đòi hỏi khả năng hiểu câu hỏi, cơ sở tri thức liên quan, và tổng hợp câu trả lời.
- *Tách thông tin* (information extraction). Tách thông tin có ngữ nghĩa từ văn bản, chẳng hạn tên riêng, thời gian, quan hệ giữa các thực thể, các thông tin có ý nghĩa khác và lưu các thông tin này dưới dạng thuận tiện cho việc xử lý bằng máy tính. Bài toán xác định và tách tên riêng được gọi là nhận dạng thực thể có tên (Named Entity Recognition). Kết quả tách thông tin được sử dụng trong nhiều bài toán khác, chẳng hạn để xây dựng cơ sở tri thức cho các hệ thống trả lời tự động.
- *Tổng hợp ngôn ngữ tự nhiên*. Biến đổi từ các thông tin tiện dùng cho máy tính, chẳng hạn thông tin trong cơ sở dữ liệu, thành các câu trên ngôn ngữ tự nhiên của người với mục đích giúp việc giao tiếp của máy tính với người được tự nhiên và thuận lợi hơn.

Đây là bài toán ngược với bài toán hiểu nhận dạng tiếng nói và hiểu ngôn ngữ tự nhiên.

b) Lập luận và suy diễn

Sau khi cảm nhận được thông tin về môi trường xung quanh, hệ thống cần có cơ chế để đưa ra được quyết định phù hợp. Quá trình ra quyết định thường dựa trên việc kết hợp thông tin cảm nhận được với tri thức có sẵn về thế giới xung quanh. Việc ra quyết định dựa trên tri thức được thực hiện nhờ lập luận hay suy diễn. Cũng có những trường hợp hệ thống không thực hiện suy diễn mà dựa trên những kỹ thuật khác như tìm kiếm hay tập hợp các phản xạ hoặc hành vi đơn giản.

Thành phần lập luận và ra quyết định được xây dựng dựa trên kỹ thuật từ những lĩnh vực nghiên cứu sau:

Biểu diễn tri thức (knowledge representation)

Nhiều bài toán của trí tuệ nhân tạo đòi hỏi lập luận dựa trên hình dung về thế giới xung quanh. Để lập luận được, sự kiện, thông tin, tri thức về thế giới xung quanh cần được biểu diễn dưới dạng máy tính có thể "hiểu" được, chẳng hạn dưới dạng logic hoặc ngôn ngữ trí tuệ nhân tạo nào đó. Thông thường, hệ thống cần có tri thức về: đối tượng hoặc thực thể, tính chất của chúng, phân loại và quan hệ giữa các đối tượng, tình huống, sự kiện, trạng thái, thời gian, nguyên nhân và hiệu quả, tri thức về tri thức (chúng ta biết về tri thức mà người khác có) v.v. Trong phạm vi nghiên cứu về biểu diễn tri thức, một số phương pháp biểu diễn đã được phát triển và được áp dụng như: *logic*, *mạng ngữ nghĩa*, *Frame*, *các luật* (chẳng hạn luật Nếu…Thì …), *bản thể học* (ontology).

Biểu diễn tri thức dùng trong máy tính thường gặp một số khó khăn sau. Thứ nhất, lượng tri thức mà mỗi người bình thường có về thế giới xung quanh là rất lớn. Việc xây dựng và biểu diễn lượng tri thức lớn như vậy đòi hỏi nhiều công sức. Hiện nay đang xuất hiện hướng tự động thu thập và xây dựng cơ sở tri thức tự động từ lượng dữ liệu lớn, thay vì thu thập bằng tay. Cách xây dựng tri thức như vậy được nghiên cứu nhiều trong khuôn khổ **khai phá dữ liệu** (data mining) hay hiện nay được gọi là **dữ liệu lớn** (big data) Điển hình của cách tiếp cận này là hệ thống Watson của IBM (sẽ được nhắc tới ở bên dưới). Thứ hai, tri thức trong thế giới thực ít khi đầy đủ, chính xác và nhất quán. Con người có thể sử dụng hiệu quả các tri thức như vậy, trong khi các hệ thống biểu diễn tri thức như logic gặp nhiều khó khăn. Thứ ba, một số tri thức khó biểu diễn dưới dạng biểu tượng mà tồn tại như các trực giác của con người.

Tìm kiếm (search)

Nhiều bài toán hoặc vấn đề có thể phát biểu và giải quyết như bài toán tìm kiếm trong không gian trạng thái. Chẳng hạn các bài toán tìm đường đi, bài toán tìm trạng thái thoả mãn ràng buộc. Nhiều bài toán khác của trí tuệ nhân tạo cũng có thể giải quyết bằng tìm kiếm. Chẳng hạn, lập luận logic có thể tiến hành bằng cách tìm các đường đi cho phép dẫn từ các tiền đề tới các kết luận.

Trí tuệ nhân tạo nghiên cứu cách tìm kiếm khi số trạng thái trong không gian quá lớn và không thể thực hiện tìm kiếm bằng cách vét cạn. Trong khuôn khổ trí tuệ nhân tạo đã phát triển một số phương pháp tìm kiếm riêng như tìm kiếm heuristic, tìm kiếm cục bộ, bao gồm các thuật toán tìm kiếm tiến hoá.

Lập luận, suy diễn (reasoning hay inference)

Lập luận là quá trình sinh ra kết luận hoặc tri thức mới từ những tri thức, sự kiện và thông tin đã có. Trong giai đoạn đầu, nhiều kỹ thuật lập luận tự động dựa trên việc mô phỏng hoặc học tập quá trình lập luận của con người. Các nghiên cứu về sau đã phát triển nhiều kỹ thuật suy diễn hiệu quả, không dựa trên cách lập luận của người. Điển hình là các kỹ thuật *chứng minh định lý* và *suy diễn logic*. Lập luận tự động thường dựa trên tìm kiếm cho phép tìm ra các liên kết giữa tiên đề và kết quả.

Việc suy diễn tự động thường gặp phải một số khó khăn sau. Thứ nhất, với bài toán kích thước lớn, số tổ hợp cần tìm kiếm khi lập luận rất lớn. Vấn đề này được gọi là "bùng nổ tổ hợp" và đòi hỏi có phát triển các kỹ thuật với độ phức tạp chấp nhận được. Thứ hai, lập luận và biểu diễn tri thức thường gặp vấn đề thông tin và tri thức không rõ ràng, không chắc chắn. Hiện nay, một trong những cách giải quyết vấn đề này là sử dụng lập luận xác suất, sẽ được trình bầy trong chương 4 của giáo trình. Thứ ba, trong nhiều tình huống, con người có thể ra quyết định rất nhanh và hiệu quả thay vì lập luận từng bước, chẳng hạn co tay lại khi chạm phải nước sôi. Hệ thống trí tuệ nhân tạo cần có cách tiếp cận khác với lập luận truyền thống cho những trường hợp như vậy.

Học máy (machine learning)

Học máy hay học tự động là khả năng của hệ thống máy tính tự cải thiện mình nhờ sử dụng dữ liệu và kinh nghiệm thu thập được. Học là khả năng quan trọng trong việc tạo ra tri thức của người. Do vậy, đây là vấn đề được quan tâm nghiên cứu ngay từ khi hình thành trí tuệ nhân tạo. Hiện nay, đây là một trong những lĩnh vực được quan tâm nghiên cứu nhiều nhất với rất nhiều kết quả và ứng dụng.

Học máy bao gồm các dạng chính là *học có giám sát*, *học không giám sát*, và *học tăng cường*. Trong học có giám sát, hệ thống được cung cấp đầu vào, đầu ra và cần tìm quy tắc để ánh xạ đầu vào thành đầu ra. Trong học không giám sát, hệ thống không được cung cấp đầu ra và cần tìm các mẫu hay quy luật từ thông tin đầu vào. Trong học tăng cường, hệ thống chỉ biết đầu ra cuối cùng của cả quá trình thay vì đầu ra cho từng bước cụ thể. Chi tiết về ba dạng học này được trình bầy trong chương 5.

Học máy được phát triển trong khuôn khổ cả khoa học máy tính và thống kê. Rất nhiều kỹ thuật học máy có nguồn gốc từ thống kê nhưng được thay đổi để trở thành các thuật toán có thể thực hiện hiệu quả trên máy tính.

Học máy hiện là kỹ thuật chính được sử dụng trong thị giác máy, xử lý ngôn ngữ tự nhiên, khai phá dữ liệu và phân tích dữ liệu lớn.

Lập kế hoạch (planning)

Lập kế hoạch và thời khoá biểu tự động, hay đơn giản là lập kế hoạch, là quá trình sinh ra các bước hành động cần thực hiện để thực hiện một mục tiêu nào đó dựa trên thông tin về môi trường, về hiệu quả từng hành động, về tình huống hiện thời và mục tiêu cần đạt. Lấy ví dụ một rô bốt nhận nhiệm vụ di chuyển một vật tới một vị trí khác. Kế hoạch thực hiện bao gồm xác định các bước để tiếp cận vật cần di chuyển, nhấc vật lên, xác định quỹ đạo và các bước di chuyển theo quỹ đạo, đặt vật xuống.

Khác với lý thuyết điều khiển truyền thống, bài toán lập kế hoạch thường phức tạp, lời giải phải tối ưu theo nhiều tiêu chí. Việc lập kế hoạch có thể thực hiện trước đối với môi trường không thay đổi, hoặc thực hiện theo thời gian với môi trường động.

Lập kế hoạch sử dụng các kỹ thuật tìm kiếm và tối ưu, các phương pháp quy hoạch động để tìm ra lời giải. Một số dạng biểu diễn và ngôn ngữ riêng cũng được phát triển để thuận lợi cho việc mô tả yêu cầu và lời giải.

c) Hành động

Cho phép hệ thống tác động vào môi trường xung quanh hoặc đơn giản là đưa ra thông tin về kết luận của mình. Thành phần này được xây dựng dựa trên những kỹ thuật sau.

Tổng hợp ngôn ngữ tự nhiên và tiếng nói.

Các kỹ thuật này đã được nhắc tới trong phần xử lý ngôn ngữ tự nhiên ở trên.

Kỹ thuật rô bốt (robotics)

Là kỹ thuật xây dựng các cơ quan chấp hành như cánh tay người máy, tổng hợp tiếng nói, tổng hợp ngôn ngữ tự nhiên. Đây là lĩnh vực nghiên cứu giao thoa giữa cơ khí, điện tử, và trí tuệ nhân tạo. Bên cạnh kỹ thuật cơ khí để tạo ra các cơ chế vật lý, chuyển động, cần có thuật toán và chương trình điều khiển hoạt động và chuyển động cho các cơ chế đó. Chẳng hạn, với cánh tay máy, cần tính toán quỹ đạo và điều khiển cụ thể các khớp nối cơ khí khi muốn di chuyển tay tới vị trí xác định và thực hiện hành động nào đó. Đây là những thành phần của kỹ thuật rô bốt mà trí tuệ nhân tạo có đóng góp chính. Ngoài ra, việc xây dựng những rô bốt thông minh chính là xây dựng các hệ thống trí tuệ nhân tạo hoàn chỉnh.

1.3.2. Một số ứng dụng và thành tựu

a. Các chương trình trò chơi

Xây dựng chương trình có khả năng chơi những trò chơi trí tuệ là lĩnh vực có nhiều thành tựu của trí tuệ nhân tạo. Với những trò chơi tương đối đơn giản như cờ ca rô hay cờ thỏ cáo, máy tính đã thắng người từ cách đây vài thập kỷ.

Đối với những trò chơi phức tạp hơn, các hệ thống trí tuệ nhân tạo cũng dần đuổi kịp và vượt qua con người. Sự kiện quan trọng thường được nhắc tới là vào tháng 5 năm 1997 chương trình cờ vua *Deep Blue* của IBM đã thắng vô địch cờ vua thế giới lúc đó là Gary Kasparov. Trong vòng đấu kéo dài 6 ván, Deep Blue thắng Kasparov với điểm số 3.5 : 2.5. Đây là lần đầu tiên máy tính thắng đương kim vô địch cờ vua thế giới.

Một trường hợp tiêu biểu khác là hệ thống *trả lời tự động Watson* cũng của IBM đã chiến thắng hai quán quân của Jeopardy trong trò chơi này vào năm 2011. Jeopardy là trò chơi hỏi đáp trên truyền hình Mỹ, tương tự "Ai là triệu phú" trên truyền hình Việt Nam nhưng trong đó ba người chơi phải thi với nhau không những trả lời đúng mà còn phải nhanh. Watson là hệ thống hỏi đáp do IBM xây dựng dựa trên việc thu thập và phân tích thông tin từ khoảng 200 triệu trang Web, trong đó có toàn bộ Wikipedia. Trong một cuộc đấu với hai cựu quán quân Jeopardy, Watson đã giành thắng lợi và phần thưởng 1 triệu USD. Các kỹ thuật sử dụng trong Watson như thu thập thông tin, phát hiện tri thức, hiểu ngôn ngữ tự nhiên, tìm kiếm, đã được IBM thương mại hóa và có thể sử dụng trong nhiều ứng dụng.

b. Nhận dạng tiếng nói

Nhận dạng tiếng nói là biến đổi từ âm thanh tiếng nói thành các văn bản. Hiện người dùng công cụ tìm kiếm Google có thể đọc vào câu truy vấn thay cho việc gõ từ khóa như trước. Các điện thoại di động thông minh cũng có khả năng nhận dạng giọng nói và trả lời các câu hỏi. Ví dụ điển hình là *chương trình trợ giúp Siri* trên điện thoại thông minh của Apple (sử dụng công nghệ nhận dạng tiếng nói của hãng Nuance) hay hệ thống *Google Now*.

Chất lượng nhận dạng giọng nói đang được cải thiện và tiến bộ rất nhanh trong vài năm gần đây. Các hệ thống nhận dạng tiếng nói hiện tại cho phép nhận dạng tới vài chục ngôn ngữ khác nhau và không phụ thuộc vào người nói (ở một mức độ nhất định).

c. Thị giác máy tính

Mặc dù nhiều ứng dụng của thị giác máy tính vẫn chưa đạt tới độ chính xác như người, nhưng trong một số bài toán, thị giác máy tính cho độ chính xác tương đương hoặc gần với khả năng của người. Tiêu biểu phải kể đến các hệ thống *nhận dạng chữ in* với độ chính xác gần như tuyệt đối, hệ thống *nhận dạng tròng mắt*, *vân tay*, *mặt người*. Những hệ thống dạng này được sử dụng rộng rãi trong sản xuất để kiểm tra sản phẩm, trong hệ thống camera an ninh. Ứng dụng *nhận dạng mặt người* trên Facebook được dùng để xác định những người quen xuất hiện trong ảnh và gán nhãn tên cho người đó.

Các ứng dụng nhận dạng hiện nay đang được cải thiện nhiều nhờ sử dụng kỹ thuật học sâu (deep learning), trong đó các mạng nơ ron có nhiều lớp được kết nối với nhau được sử dụng để phát hiện các đặc trưng của đối tượng ở mức từ đơn giản tới phức tạp.

d. Các thiết bị tự lái

Các thiết bị tự lái bao gồm máy bay, ô tô, tầu thủy, thiết bị thám hiểm vũ trụ có thể tự di chuyển mà không có sự điều khiển của người (cả điều khiển trực tiếp và điều khiển từ xa). Hiện ô tô tự lái đang được một số hãng công nghệ và các tổ chức khác nghiên cứu và phát triển, trong đó có những dự án nổi tiếng như xe tự lái của Google. Mặc dù tại thời điểm viết sách này mới chỉ có một mẫu xe duy nhất được thương mại hóa dùng cho các khu đi bộ và chỉ có thể chạy với tốc độ khoảng 20 km/giờ nhưng các dự báo cho thấy xe tự lái sẽ được thương mại hóa thành công trong vòng vài năm tới. Các thiết bị tự lái khác bao gồm cả các xem thám hiểm vũ trụ và hành tinh khác như xe thám hiểm sao Hỏa của NASA.

e. Hệ chuyên gia

Là các hệ thống làm việc dựa trên kinh nghiệm và tri thức của chuyên gia trong một lĩnh vực tương đối hẹp nào đó để đưa ra khuyến cáo, kết luận, chuẩn đoán một cách tự động. Các ví dụ gồm:

- MYCIN: hệ chuyên gian đầu tiên chẩn đoán bệnh về nhiễm trùng máu và cách điều trị với khả năng tương đương một bác sĩ giỏi trong lĩnh vực này.
- XCON của DEC: hỗ trợ chọn cấu hình máy tính tự động.

f. Xử lý, hiểu ngôn ngữ tự nhiên

Tiêu biểu là các hệ thống dịch tự động như hệ thống dịch của Google, các hệ thống tóm tắt nội dung văn bản tự động. Hệ thống dịch tự động của Google sử dụng các mô hình thống kê xây dựng từ các văn bản song ngữ và các văn bản đơn ngữ. Hệ thống này có khả năng dịch qua lại giữa vài chục ngôn ngữ.

Các hệ thống hỏi đáp được đề cập tới trong phần về trò chơi và nhận dạng tiếng nói cũng thuộc loại ứng dụng xử lý ngôn ngữ tự nhiên. Những hệ thống này sử dụng những thành phần đơn giản hơn như các phân hệ phân tích hình thái, cú pháp, ngữ nghĩa.

Nhiều kỹ thuật xử lý ngôn ngữ tự nhiên đã được ứng dụng trong các ứng dụng rất thiết thực như các bộ lọc thư rác. Dịch vụ thư điện tử của Google, Microsoft, Yahoo đều có các bộ lọc thư rác với cơ chế học tự động và thích nghi với thay đổi của người phát tán. Khả năng phát hiện thư rác của các hệ thống này là rất cao, gần như tuyệt đối trong một số trường hợp.

g. Lập kế hoạch, lập thời khóa biểu

Kỹ thuật trí tuệ nhân tạo được sử dụng nhiều trong bài toán lập thời khóa biểu cho trường học, xí nghiệp, các bài toán lập kế hoạch khác. Một ví dụ lập kế hoạch thành công với quy mô lớn là kế hoạch đảm bảo hậu cần cho quân đội Mỹ trong chiến dịch Cơn bão sa mạc tại Iraq đã được thực hiện gần như hoàn toàn dựa trên kỹ thuật trí tuệ nhân tạo. Đây là một kế hoạch lớn, liên quan tới khoảng 50000 thiết bị vận tải và người tại cùng một thời điểm. Kế hoạch bao gồm điểm xuất phát, điểm tới, thời gian, phương tiện và người tham gia sao cho không mâu thuẫn và tối ưu theo các tiêu chí.

h. Rô bốt

Một số rô bốt được xây dựng sao cho có hình dạng tương tự con người và khả năng toàn diện như thị giác máy, giao tiếp bằng ngôn ngữ tự nhiên, khả năng lập luận nhất định, khả năng di chuyển và thực hiện các hành động như nhảy múa. Các rô bốt này chủ yếu được tạo ra để chứng minh khả năng của kỹ thuật rô bốt thay vì hướng vào ứng dụng cụ thể. Trong số này có thể kể tới rô bốt Asimo, rô bốt Nao.

Bên cạnh đó, một số rô bốt không mô phỏng người nhưng được sử dụng trong đời sống hàng ngày hoặc các ứng dụng thực tế. Ví dụ, rô bốt Roomba của hãng iRobot có khả năng tự động di chuyển trong phòng, tránh vật cản, chui vào các ngóc ngách để lau sạch toàn bộ sàn. Số lượng rô bốt Roomba đã bán lên tới vài triệu bản.

1.3.3. Những vấn đề chưa được giải quyết

Mặc dù đạt được nhiều thành tựu và có nhiều ứng dụng đáng kể, các hệ thống trí tuệ nhân tạo hiện nay chưa đạt được mức độ *trí tuệ nhân tạo mạnh* (strong AI) hay trí tuệ nhân tạo tổng quát (Artificial General Intelligence). Đây cũng được coi là vấn đề khó nhất và chưa được giải quyết. Trí tuệ nhân tạo mạnh là khái niệm để chỉ khả năng của máy tính thực hiện bất cứ công việc trí tuệ nào mà con người có thể thực hiện. Khái niệm trí tuệ mạnh được sử dụng để phân biệt với *trí tuệ nhân tạo yếu* (weak AI) hay *trí tuệ nhân tạo ứng dụng* (applied AI), tức là dùng máy tính để giải quyết từng bài toán ra quyết định hay lập luận đơn lẻ. Như vậy, trí tuệ nhân tạo mạnh đòi hỏi giải quyết đầy đủ các công việc trí tuệ như người trong khi trí tuệ nhân tạo yếu giải quyết bài toán cụ thể.

Các khó khăn để đạt được trí tuệ nhân tạo tổng quát bao gồm khả năng thị giác máy, xử lý ngôn ngữ tự nhiên, khả năng xử lý các tình hướng mới, tình huống không ngờ tới khi giải quyết các bài toán thực tế. Đây là những lĩnh vực mà máy tính còn thua kém con người. Các hệ thống trí tuệ nhân tạo hiện nay có thể giải quyết tốt bài toán đặt ra trong một phạm vi hẹp. Tuy nhiên, khi gặp vấn đề thực tế ở phạm vi rộng hơn, hệ thống trí tuệ nhân tạo thường không thể xử lý được các tình huống mới, vượt ra ngoài ngữ cảnh ban đầu của bài toán. Ngược lại,

con người có khả năng xử lý tốt hơn nhiều những trường hợp như vậy do có hiểu biết rộng về thế giới xung quanh. Việc trang bị cho máy tính lượng tri thức như con người hiện vẫn là vấn đề chưa được giải quyết.

Để đánh giá hệ thống trí tuệ nhân tạo, có thể so sánh các hệ thống này với con người khi thực hiện từng bài toán trí tuệ cụ thể. Kết quả so sánh được chia thành các mức sau:

- *Tối ưu*: hệ thống cho kết quả tối ưu, không thể tốt hơn nữa.
- *Tốt hơn người*: hệ thống cho kết quả tốt hơn bất cứ người nào.
- *Tương đương người giỏi*: hệ thống cho kết quả tương đương những người giỏi (nhất) và hơn đa số người còn lại.
- *Tương đương người*: hệ thống cho kết quả tương đương đa số người.
- *Kém hơn người*.

Những bài toán trí tuệ trong đó hệ thống máy tính thực hiện kém hơn người là những bài toán cần tiếp tục giải quyết. Dưới đây là một số bài toán và mức độ so sánh giữa máy tính và người.

- Rubik, cờ ca rô 3 ô: tối ưu.
- Cờ vua: gần đạt mức tốt hơn người. Hệ thống Deep Blue đã thắng đương kim vô địch cờ vua thế giới Gary Kasparov.
- Trả lời câu hỏi tự động: gần đạt mức tốt hơn người. Hệ thống IBM Watson đã thắng người trong trò chơi truyền hình Jeopardy.
- Lái xe tự động: tương đương người. Một ví dụ là hệ thống lái xem tự động của Google có kết quả lái an toàn và êm ái hơn so với người. Tuy vậy, kết quả này mới chỉ trong các thử nghiệm tại Mỹ, nơi có hệ thống giao thông tốt. Chưa rõ hệ thống cho kết quả thế nào trong các điều kiện khác, ví dụ trong điều kiện giao thông tại Việt Nam.
- Nhận dạng chữ in (theo chuẩn ISO 1073-1:1976, tức là các phông chữ đơn nét và không quá phức tạp): tương đương người. Các hệ thống OCR hiện nay có độ chính xác 99% hoặc hơn trên văn bản in bằng máy in la de.
- Nhận dạng chữ viết tay: kém người. Mặc dù có nhiều tiến bộ, các hệ thống nhận dạng chữ viết tay tự động vẫn có độ chính xác kém hơn người với khoảng cách tương đối lớn.
- Nhận dạng đối tượng: kém người. Trừ một số trường hợp đặc biệt như nhận dạng vân tay, tròng mắt, mặt người, khả năng nhận dạng đối tượng nói chung của máy tính hiện vẫn kém khá xa khả năng của người.
- Dịch tự động: kém người. Mặc dù hiện nay, khả năng dịch tự động của máy kém con người trên từng cặp ngôn ngữ cụ thể nhưng những hệ thống dịch tự động như Google translate lại vượt từng người cụ thể về số lượng ngôn ngữ có thể dịch.
- Nhận dạng tiếng nói: kém người. Trong vài năm gần đây, khả năng nhận dạng tiếng nói bằng máy tính đã có rất nhiều tiến bộ và được ứng dụng rộng rãi như các hệ thống Google Voice, hay Siri (dùng phần nhận dạng của hãng Nuance). Tuy nhiên, khả năng nhận dạng tự động vẫn kém khả năng con người.
- Phân biệt nhập nhằng trong nghĩa từ và một số bài toán xử lý ngôn ngữ tự nhiên khá

CHƯƠNG 2: GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Chương này sẽ trình bầy về kỹ thuật giải quyết vấn đề bằng tìm kiếm, còn được gọi là tìm kiếm trong không gian trạng thái. Đây là các phương pháp được dùng phổ biến trong một lớp lớn các bài toán và là lớp kỹ thuật giải quyết vấn đề quan trọng, được nghiên cứu và ứng dụng nhiều của trí tuệ nhân tạo. Trong phạm vi chương, ta sẽ xem xét cách phát biểu một vấn đề dưới dạng bài toán tìm kiếm, trước khi chuyển sang các giải thuật đã được phát triển để giải quyết bài toán này. Các giải thuật tìm kiếm được chia thành ba nhóm lớn: tìm kiếm mù (không có thông tin), tìm kiếm heuristics (có thông tin), và tìm kiếm cục bộ. Giải thuật thuộc nhóm hai và nhóm ba, đặc biệt là nhóm ba, được sử dụng cho các bài toán thực tế với kích thước lớn.

2.1. GIẢI QUYẾT VẤN ĐỀ VÀ KHOA HỌC TRÍ TUỆ NHÂN TẠO

Tại sao phải tìm kiếm

Khi quan sát các bài toán trên thực tế, có thể nhận thấy một lớp lớn bài toán có thể phát biểu và giải quyết dưới dạng tìm kiếm, trong đó yêu cầu có thể là tìm những trạng thái, tính chất thỏa mãn một số điều kiện nào đó, hoặc tìm chuỗi hành động cho phép đạt tới trạng thái mong muốn. Sau đây là một số ví dụ các bài toán như vậy.

- Trò chơi: nhiều trò chơi, ví dụ cờ vua, thực chất là quá trình tìm kiếm nước đi của các bên trong số những nước mà luật chơi cho phép, để giành lấy ưu thế cho bên mình.
- Lập lịch, hay thời khóa biểu: lập lịch là lựa chọn thứ tự, thời gian, tài nguyên (máy móc, địa điểm, con người) thỏa mãn một số tiêu chí nào đó. Như vậy, lập lịch có thể coi như quá trình tìm kiếm trong số tổ hợp phương án sắp xếp phương án đáp ứng yêu cầu đề ra.
- Tìm đường đi: cần tìm đường đi từ điểm xuất phát tới đích, có thể thỏa mãn thêm một số tiêu chí nào đó như tiêu chí tối ưu về độ dài, thời gian, giá thành v.v.
- Lập kế hoạch: là lựa chọn chuỗi hành động cơ sở cho phép đạt mục tiêu đề ra đồng thời thỏa mãn các yêu cầu phụ.

Sự phổ biến của các vấn đề có tính chất tìm kiếm dẫn tới yêu cầu phát biểu bài toán tìm kiếm một cách tổng quát, đồng thời xây dựng phương pháp giải bài toán tìm kiếm sao cho hiệu quả, thuận lợi. Các bài toán tìm kiếm mới có thể đưa về dạng bài toán tìm kiếm tổng quát và áp dụng thuật giải đã được xây dựng.

Do tính quan trọng của lớp bài toán này, tìm kiếm đã được nghiên cứu từ rất sớm trong khuôn khổ toán rời rạc và lý thuyết giải thuật. Trong khuôn khổ trí tuệ nhân tạo, tìm kiếm được đặc biệt quan tâm từ khía cạnh xây dựng phương pháp cho phép tìm ra kết quả trong trường hợp không gian tìm kiếm có kích thước lớn khiến cho những phương pháp truyền thống gặp khó khăn. Rất nhiều thuật toán tìm kiếm được phát triển trong khuôn khổ trí tuệ nhân tạo được phát triển dựa trên việc tìm hiểu quá trình giải quyết vấn đề của con người, hay

dựa trên sự tương tự với một số quá trình xảy ra trong tự nhiên như quá trình tiến hóa, quá trình hình thành mạng tinh thể, cách thức di chuyển của một số loại côn trùng.

Ngoài việc đứng độc lập như chủ đề nghiên cứu riêng, tìm kiếm còn là cơ sở cho rất nhiều nhánh nghiên cứu khác của trí tuệ nhân tạo như lập kế hoạch, học máy, xử lý ngôn ngữ tự nhiên, suy diễn. Chẳng hạn, bài toán suy diễn có thể phát biểu như quá trình tìm ra chuỗi các luật suy diễn cho phép biến đổi từ quan sát ban đầu tới đích của quá trình suy diễn. Nhiều phương pháp học máy cũng coi quá trình học như quá trình tìm kiếm mô hình cho phép biểu diễn dữ liệu huấn luyện được dùng trong quá trình học.

Lưu ý: cần phân biệt bài toán tìm kiếm được trình bầy trong chương này, còn được gọi là tìm kiếm trong không gian trạng thái, với bài toán tìm kiếm và thu hồi thông tin (information retrieval) được giải quyết trong các máy tìm kiếm (search engine) như Google, Yahoo, hay Bing. Trong bài toán thu hồi thông tin, các máy tìm kiếm cần tìm các văn bản, hình ảnh, tài liệu đa phương tiện thỏa mãn nhu cầu thông tin của người dùng. Việc này được thực hiện bằng các kỹ thuật khác với các kỹ thuật sẽ được trình bầy trong các phần dưới đây.

Ngoài ra, cũng cần phân biệt bài toán tìm kiếm trong không gian trạng thái với các bài toán như tìm kiếm xâu tương tự, tìm kiếm k láng giềng gần nhất. Mặc dù cùng có tên là tìm kiếm nhưng những bài toán này có mục tiêu, tính chất và cách giải quyết khác với tìm kiếm trong không gian trạng thái được trình bầy ở chương này.

2.2. BÀI TOÁN TÌM KIẾM TRONG KHÔNG GIAN TRẠNG THÁI

2.2.1. Phát biểu bài toán tìm kiếm

Một cách tổng quát, một vấn đề có thể giải quyết thông qua tìm kiếm bằng cách xác định tập hợp các phương án, đối tượng, hay trạng thái liên quan, gọi chung là *không gian trạng thái*. Thủ tục tìm kiếm sau đó sẽ khảo sát không gian trạng thái theo một cách nào đó để tìm ra lời giải cho vấn đề. Trong một số trường hợp, thủ tục tìm kiếm sẽ khảo sát và tìm ra chuỗi các hành động cho phép đạt tới trạng thái mong muốn và bản thân chuỗi hành động này là lời giải của bài toán, như trong bài toán tìm đường. Tùy vào cách thức khảo sát không gian trạng thái cụ thể, ta sẽ có những phương pháp tìm kiếm khác nhau.

Để có thể khảo sát không gian trạng thái, thuật toán tìm kiếm bắt đầu từ một trạng thái xuất phát nào đó, sau đó sử dụng những phép biến đổi trạng thái để nhận biết và chuyển sang trạng thái khác. Quá trình tìm kiếm kết thúc khi tìm ra lời giải, tức là khi đạt tới *trạng thái đích*.

Bài toán tìm kiếm cơ bản có thể phát biểu thông qua năm thành phần chính sau:

- Tập các trạng thái Q. Đây chính là không gian trạng thái của bài toán.
- Tập (không rỗng) các trạng thái xuất phát S ($S \subseteq Q$). Thuật toán tìm kiếm sẽ xuất phát từ một trong những trạng thái này để khảo sát không gian tìm kiếm.
- Tập (không rỗng) các trạng thái đích G ($G \subseteq Q$). Trạng thái đích có thể được cho một cách tường minh, tức là chỉ ra cụ thể đó là trạng thái nào, hoặc không tường minh. Trong trường hợp sau, thay vì trạng thái cụ thể, bài toán sẽ quy định một số điều kiện

mà trạng thái đích cần thỏa mãn. Ví dụ, khi chơi cờ vua, thay vì chỉ ra vị trí cụ thể của quân cờ, ta chỉ có quy tắc cho biết trạng thái chiếu hết.

- Các *toán tử*, còn gọi là *hành động* hay *chuyển động* hay *hàm chuyển tiếp*. Thực chất đây là một ánh xạ P: $Q \rightarrow Q$, cho phép chuyển từ trạng thái hiện thời sang các trạng thái khác. Với mỗi trạng thái n, P (n) là tập các trạng thái được sinh ra khi áp dụng toán tử hay chuyển động P cho trạng thái đó. Các trạng thái này được gọi là trạng thái hàng xóm hay láng giềng của n.

- Giá thành c: Q x $Q \rightarrow R$. Trong một số trường hợp, quá trình tìm kiếm cần quan tâm tới giá thành đường đi. Giá thành để di chuyển từ nút x tới nút hàng xóm y được cho dưới dạng số không âm c (x, y). Giá thành cụ thể được chọn tùy vào từng trường hợp và thể hiện mối quan tâm chính trong bài toán. Ví dụ, với cùng bài toán tìm đường đi, giá thành có thể tính bằng độ dài quãng đường, hay thời gian cần để di chuyển, hay giá thành nhiên liệu tiêu thụ. Giá trị c (x, y) được gọi là *giá thành bước*, từ giá thành bước có thể tính ra giá thành toàn bộ đường đi bằng cách lấy tổng các bước đi đã thực hiện.

Với bài toán phát biểu như trên, *lời giải* là chuỗi chuyển động cho phép di chuyển từ trạng thái xuất phát tới trạng thái đích. Chất lượng lời giải được tính bằng giá thành đường đi, tức là tổng giá thành cần để thực hiện chuỗi chuyển động này, giá thành càng thấp thì lời giải càng tốt. Cũng có những trường hợp ta không quan tâm tới chuỗi chuyển động mà chỉ quan tâm tới trạng thái đích, ví dụ trong bài toán lập lịch. Trong trường hợp đó, thuật toán tìm kiếm cần trả về lời giải là trạng thái đích, thay vì trả về chuỗi chuyển động.

Cần lưu ý rằng, không gian trạng thái có thể cho một cách tường minh, bằng cách liệt kê các trạng thái như trong trường hợp bài toán tìm đường với mỗi địa điểm là một trạng thái. Tuy nhiên, trong nhiều trường hợp, không gian trạng thái được cho một cách không tường minh thông qua trạng thái xuất phát và các chuyển động: khi đó không gian trạng thái là tập hợp tất cả các trạng thái có thể đạt tới từ trạng thái xuất phát bằng cách áp dụng mọi tổ hợp chuyển động. Trong trường hợp này, không gian trạng thái có thể là hữu hạn hoặc vô hạn.

2.2.2. Một số ví dụ

Các thành phần của bài toán tìm kiếm được minh họa trên những ví dụ sau. Một số ví dụ không có ứng dụng thực tế nhưng đơn giản, dễ sử dụng cho mục đích minh họa. Một số khác là những bài toán thực tế, có nhiều ứng dụng.

Bài toán đố tám ô

Cho hình chữ nhật được chia thành chín ô như trên hình dưới, trong đó tám ô được đánh số từ 1 đến 8 và một ô trống. Có thể thay đổi vị trí các số bằng cách di chuyển ô trống. Mỗi lần di chuyển, ô trống có thể đổi chỗ với một ô số ở ngay phía trên, phía dưới, bên phải hoặc bên trái.

Yêu cầu của bài toán là thực hiện các di chuyển để chuyển từ một sắp xếp các ô (ví dụ trên hình bên trái) sang một cách sắp xếp khác (hình bên phải). Ngoài ra có thể có yêu cầu phụ, ví dụ cần di chuyển sao cho số lần đổi chỗ ô trống là tối thiểu.

3	1	6
5		8
2	7	4

Trạng thái xuất phát

	1	2
3	4	5
6	7	8

Trạng thái đích

Hình 2.1. Trò đố 8 ô

Trò đố 8 ô có thể phát biểu như bài toán tìm kiếm với các thành phần sau.

- Trạng thái: mỗi trạng thái là một sắp xếp cụ thể vị trí các ô
- Hành động: mỗi hành động tương ứng với một di chuyển ô trống trái, phải, lên, xuống
- Trạng thái xuất phát: được cho trước (trạng thái bên trái trong hình 2.1)
- Trạng thái đích: được cho một cách tường minh (trạng thái bên phải trong hình 2.1)
- Giá thành: bằng tổng số lần dịch chuyển ô trống. Nói cách khác, mỗi chuyển động có giá thành bằng 1.

Lời giải trong trường hợp này là chuỗi các hành động cho phép di chuyển từ trạng thái xuất phát tới đích. Lời giải cần ít hành động hơn là lời giải tốt hơn.

Bài toán n con hậu

Cho một bàn cờ vua kích thước n x n. Cần xếp n con hậu lên bàn cờ sao cho không có đôi hậu nào đe dọa nhau. Trường hợp riêng của bài toán này là trường hợp $n = 8$, khi đó bàn cờ là bàn cờ vua truyền thống kích thước 8 x 8.

Đây cũng là bài toán tìm kiếm với các thành phần cụ thể như sau:

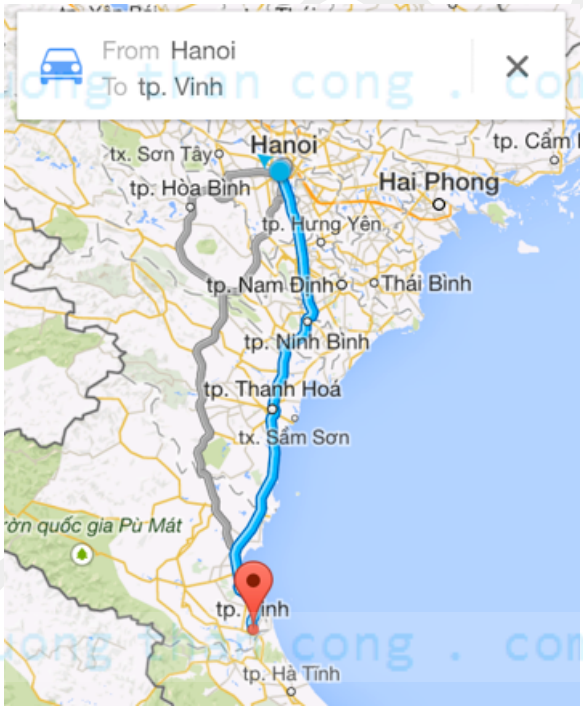
- Trạng thái: ở đây có hai cách xác định trạng thái. Theo cách *thứ nhất*, trên bàn cờ luôn có n con hậu, khi đó mỗi trạng thái được xác định bởi vị trí cụ thể của n con hậu. Theo cách *thứ hai*, trên bàn cờ có thể có từ 0 tới n con hậu, khi đó mỗi trạng thái được xác định bởi sắp xếp của từ 0 tới n con hậu trên bàn cờ và mỗi hành động được thực hiện bằng cách thêm một con hậu vào trạng thái cũ.
- Trạng thái xuất phát: theo cách biểu diễn trạng thái thứ nhất, trạng thái xuất phát có thể được chọn bằng cách sắp xếp ngẫu nhiên n con hậu. Theo cách thứ hai, trạng thái xuất phát là bàn cờ trống không có con hậu nào.
- Trạng thái đích: được cho bằng quy tắc kiểm tra, theo đó trạng thái là đích nếu gồm đủ n con hậu và không có quá một con trên cùng một cột, hoặc cùng một dòng, hoặc cùng một đường chéo.

- Chuyển động: Có nhiều cách khác nhau. Ví dụ, với cách xác định trạng thái thứ nhất, mỗi chuyển động được thực hiện bằng cách đổi chỗ 2 con hậu, di chuyển một con hậu sang ô khác (cùng cột, khác cột). Với cách xác định trạng thái thứ hai và trạng thái đích là bàn cờ trống, ta có thể quy định chuyển động như sau: tại mỗi chuyển động, tìm cách đặt thêm một con hậu và cột trái nhất còn trống sao cho con hậu mới đặt vào không đe dọa các con trước (cách chuyển động này cho không gian trạng thái nhỏ hơn nhiều so với các kiểu chuyển động khác và mỗi trạng thái chỉ cho phép không quá một con hậu trong một cột).

Trong bài toán này, khác với bài toán tám ô, ta không quan tâm tới đường đi mà chỉ quan tâm tới trạng thái đích tìm được.

Bài toán tìm đường đi

Đây là bài toán có rất nhiều ứng dụng. Dạng đơn giản và thường gặp nhất là tìm đường đi giữa hai điểm trên bản đồ, có thể là đường đi theo đường bộ như trên hình 2.2, đường không, hoặc đường thủy. Các công cụ tìm đường dạng này có thể gặp trong ứng dụng bản đồ như của Google, bán kèm máy định vị GPS, công cụ hỗ trợ lái xe, dịch vụ đặt chuyến bay v.v. Bài toán tìm đường đi cũng gặp trong định tuyến cho mạng, chẳng hạn mạng Internet, trong đó cần tìm đường đi cho các gói tin giữa hai nút mạng.



Hình 2.2. Ứng dụng tìm kiếm đường đi trên bản đồ

Xét ví dụ tìm kiếm đường bộ trên bản đồ như trong ví dụ hình 2.2. Ta có các thành phần bài toán tìm kiếm như sau:

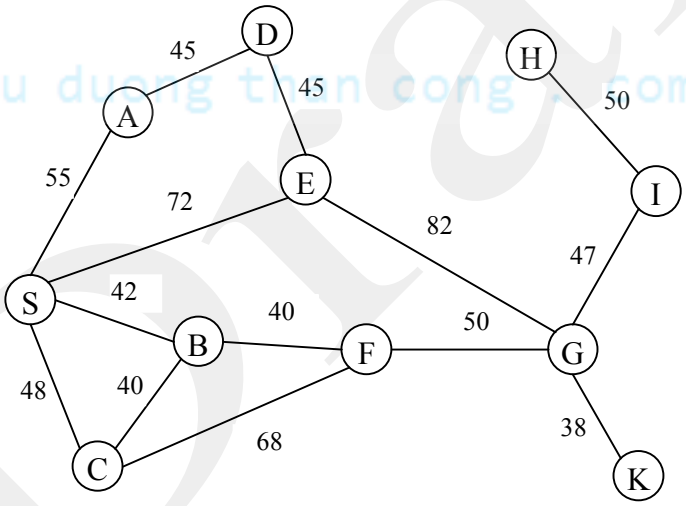
- Trạng thái: mỗi trạng thái là một địa danh trên đường đi, ví dụ một thành phố, thị xã, thị trấn, hay một nút giao thông. Để cho đơn giản, ta sẽ quy định mỗi trạng thái là một nút giao thông từ mức thị trấn trở lên. Ví dụ, trạng thái hiện thời có thể là ở

nút giao thông Ninh Bình, hay nút giao thông Hòa Bình với bản đồ trên hình 2.2.

- Trạng thái xuất phát: do người dùng xác định.
- Trạng thái đích: do người dùng xác định.
- Chuyển động: được xác định bằng tập các nút giao thông láng giềng của trạng thái hiện thời, tức là các nút giao thông có thể di chuyển tới từ nút giao thông hiện thời mà không phải đi qua một nút khác.
- Giá thành: có thể tính bằng khoảng cách (số km) để di chuyển từ nút giao thông này sang nút giao thông khác. Giá thành cũng có thể tính bằng thời gian để di chuyển, số tiền phải bỏ ra cho tiền xăng và tiền phí giao thông (nếu có) v.v.

Các hệ thống tìm đường sẽ trả về đường đi tức là chuỗi các chuyển động ngắn nhất theo hàm giá thành được chọn.

Để tiện cho việc tập trung vào các thành phần chính, khi phát biểu bài toán tìm đường đi, nhiều chi tiết thường được bỏ qua như quỹ đạo thực tế, điều kiện thời tiết, chất lượng đường v.v. Khi đó, bài toán tìm đường đi thường được biểu diễn dưới dạng tìm đường đi giữa hai nút trên đồi thị như minh họa trên hình 2.3. Tại mỗi nút, các chuyển động được phép là chuyển động sang nút liền kề. Giá thành đường đi giữa hai nút liền kề được thể hiện trên cung nối hai nút.



Hình 2.3. Ví dụ bài toán tìm đường đi trên đồ thị. Các số trên cung thể hiện giá thành.

2.2.3.Thuật toán tìm kiếm tổng quát và cây tìm kiếm

Một cách tổng quát, các thuật toán tìm kiếm dựa trên nguyên lý chung như sau:

Nguyên lý chung: bắt đầu từ trạng thái xuất phát, sử dụng các hàm chuyển động để di chuyển trong không gian trạng thái cho tới khi đạt đến trạng thái mong muốn. Trả về chuỗi chuyển động hoặc trạng thái đích tìm được tùy vào yêu cầu bài toán.

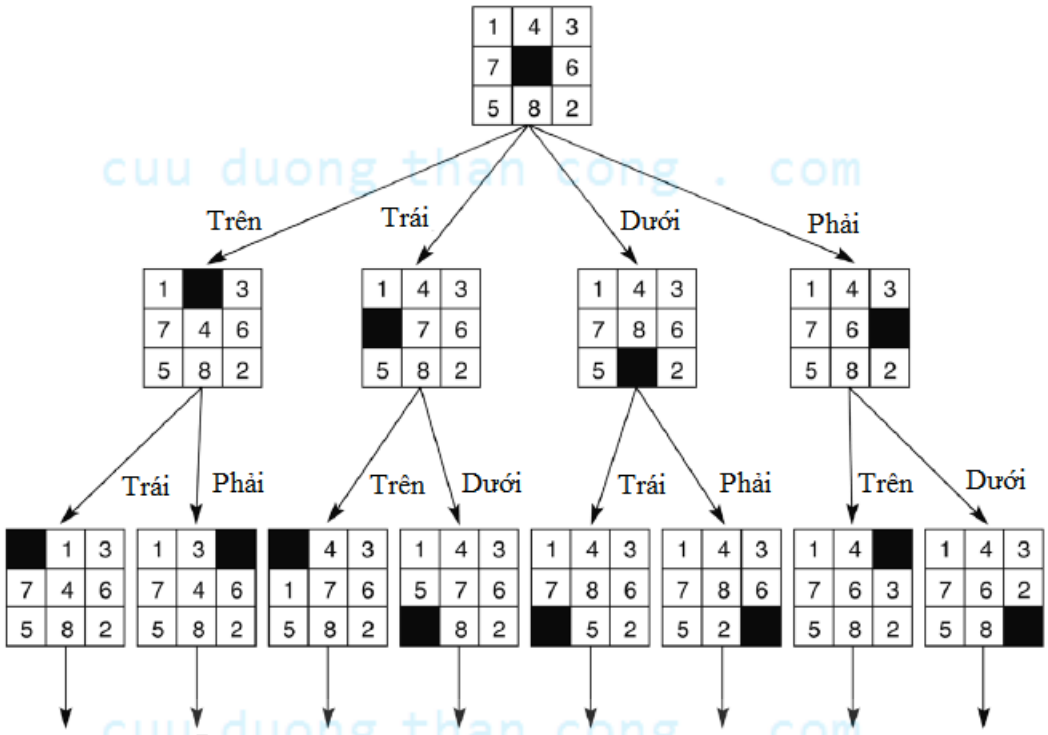
Để minh họa nguyên lý tìm kiếm này, ta xét ví dụ trò đố 8 ô với trạng thái xuất phát là trạng thái trên cùng trong hình 2.4. Khởi đầu từ trạng thái xuất phát, thuật toán kiểm tra xem đây có phải đích chưa. Nếu chưa, ta áp dụng các chuyển động để sinh ra các trạng thái láng giềng. Tiếp theo, ta xét một trạng thái vừa được sinh ra, gọi là trạng thái hiện thời bằng cách

kiểm tra xem đây đã phải đích chưa. Nếu chưa, ta *mở rộng* trạng thái hiện thời bằng cách áp dụng các chuyển động được phép đối với trạng thái đó để sinh ra các trạng thái khác. Quá trình tìm kiếm như vậy kết thúc khi tìm được trạng thái đích hoặc khi không còn trạng thái nào để mở rộng nữa.

Thuật toán tìm kiếm tổng quát như vậy sinh ra một *cây tìm kiếm*, trong đó mỗi trạng thái tương ứng với một *nút* trên cây, mỗi nhánh tương ứng với một chuyển động tại nút đang xét. Trạng thái xuất phát tương ứng với gốc cây, những trạng thái được mở rộng tạo thành các nút thế hệ tiếp theo. Trên hình 2.4 là ví dụ một phần cây tìm kiếm sinh ra cho bài toán đố 8 ô.

Sau đây là một số thuật ngữ sử dụng khi trình bày về thuật toán tìm kiếm:

- *Mở rộng* nút là áp dụng các chuyển động lên trạng thái tương ứng để sinh ra các nút con.
- *Nút lá* là các nút không có nút con tại thời điểm đang xét.
- Các *nút biên* (còn gọi là nút mở): là tập các nút lá có thể mở rộng tiếp.
- Tập các nút đã được mở rộng được gọi là tập các *nút đóng*, hay đơn giản là *tập đóng*.



Hình 2.4. Cây tìm kiếm cho bài toán 8 ô

Nguyên lý tìm kiếm vừa trình bàyđược thể hiện qua thuật toán tìm kiếm tổng quát trên hình 2.5. Thuật toán duy trì tập các nút biên *O* được khởi tạo bằng tập trạng thái xuất phát. Qua mỗi vòng lặp, thuật toán lấy ra một nút từ tập biện *O*, kiểm tra xem nút này có phải đích không. Nếu nút được lấy ra là đích, thuật toán trả về kết quả. Trong trường hợp ngược lại, nút này được mở rộng, tức là dùng hàm chuyển động để sinh ra các nút con. Các nút mới sinh ra lại được thêm vào tập *O*. Thuật toán kết thúc khi tìm thấy trạng thái đích hoặc khi *O* rỗng.

Search(Q, S, G, P) (Q: không gian trạng thái, S: trạng thái bắt đầu, G: đích, P: hành động)

Đầu vào: bài toán tìm kiếm với 4 thành phần như trên

Đầu ra: trạng thái đích

Khởi tạo: $O \leftarrow S$ (*O: tập các nút biên, bước này khởi tạo giá trị ban đầu cho O bằng S*)

While($O \neq Ø$) do

1. Chọn nút $n \in O$ và xóa n khỏi O
2. If $n \in G$, return (đường đi tới n)
3. Thêm $P(n)$ vào O

Return: Không có lời giải

Hình 2.5. Thuật toán tìm kiếm tổng quát

Cần lưu ý là trong thuật toán tìm kiếm tổng quát ở trên không quy định rõ nút nào trong số các nút biên (nằm trong tập O) được chọn để mở rộng. Việc lựa chọn nút cụ thể phụ thuộc vào từng phương pháp tìm kiếm và được trình bày trong những phần tiếp theo.

Tránh vòng lặp

Trong cây tìm kiếm trên hình 2.4 và thuật toán trên hình 2.5, ta đã giả sử rằng mỗi nút đã được duyệt sẽ không được duyệt lại lần nữa, do vậy không cần lưu trữ danh sách những nút đã duyệt. Trên thực tế, trong nhiều trường hợp, việc di chuyển trong không gian trạng thái sẽ dẫn tới những nút đã duyệt qua và tạo thành vòng lặp. Chẳng hạn, trên hình 2.4, từ trạng thái ngoài cùng bên trái thuộc lớp giữa có thể quay về trạng thái xuất phát nếu sử dụng chuyển động "Dưới". Tương tự, với đồ thị trên hình 2.3, thuật toán có thể dẫn tới vòng lặp G → A → S → A →… Việc chuyển động theo vòng lặp ảnh hưởng không tốt tới thuật toán tìm kiếm. Một số thuật toán tìm kiếm như tìm kiếm theo chiều sâu (trình bầy trong một phần sau) không thể tìm ra lời giải nếu rơi vào vòng lặp. Một số thuật toán khác như tìm theo chiều rộng, mặc dù vẫn tìm ra lời giải, nhưng sẽ phải tính toán lâu hơn, xem xét nhiều trạng thái hơn nếu gặp phải vòng lặp.

Để tránh không rời vào vòng lặp cần có cách kiểm tra để không xem xét lại nút đã duyệt. Cách chung nhất để tránh duyệt lại các nút là duy trì *tập các nút đóng*, tức là các nút đã được mở rộng, và nhớ tất cả các nút đã được mở rộng vào tập đóng này. Nếu một nút mới sinh ra đã có mặt trong tập các nút đóng hoặc tập nút biên thì sẽ không được thêm vào tập các nút biên nữa.

Trên hình 2.6 là phiên bản của thuật toán tìm kiếm tổng quát trình bầy trong hình 2.5, trong đó tập các nút đóng được sử dụng để lưu các nút đã mở rộng và cho phép tránh vòng lặp. Phần chữ đậm là các phần được bổ sung so với thuật toán cũ. Thuật toán này được đặt tên là Graph_Search để phân biệt với thuật toán không lưu các nút đóng.

Graph_Search(Q, S, G, P) (Q: tập trạng thái, S: trạng thái bắt đầu, G: đích, P: hành động)

Đầu vào: bài toán tìm kiếm với 4 thành phần như trên

Đầu ra: trạng thái đích

Khởi tạo: $O \leftarrow S$ // *O: tập các nút biên, bước này khởi tạo giá trị ban đầu cho O bằng S*

$\boldsymbol{D \leftarrow \emptyset}$ // D là tập nút đóng, được khởi tạo bằng rỗng

---

While($O \neq \emptyset$) do

1. chọn nút $n \in O$ và xóa n khỏi O
2. If $n \in G$, return (đường đi tới n)
3. **Thêm *n* vào *D***
4. Thêm các nút thuộc $P(n)$ vào O **nếu nút đó không thuộc *D* và *O***

Return: Không có lời giải

Hình 2.6. Thuật toán Graph_Search với danh sách các nút đóng cho phép tránh vòng lặp. Phần bôi đậm là phần khác so với thuật toán tìm kiếm không tránh vòng lặp.

Lưu ý: cần phân biệt khái niệm nút và trạng thái khi triển khai thuật toán cụ thể. Mỗi nút được gắn với một đường đi cụ thể trên cây tìm kiếm còn trạng thái là đặc trưng của thế giới bài toán. Hai nút khác nhau có thể chứa cùng một trạng thái nếu trạng thái đó được sinh ra từ hai đường đi khác nhau. Như vậy, trong nhiều thuật toán, ta chỉ không duyệt lại các nút đã được mở rộng, trong khi vẫn có thể xem xét các trạng thái đã được xem xét trước đó nhưng được sinh ra theo những đường khác trong cây tìm kiếm.

2.2.4. Các tiêu chuẩn đánh giá thuật toán tìm kiếm

Với bài toán tìm kiếm được phát biểu như trên, nhiều thuật toán tìm kiếm có thể sử dụng để khảo sát không gian và tìm ra lời giải. Để có thể so sánh với nhau, thuật toán tìm kiếm được đánh giá theo bốn tiêu chuẩn sau:

- *Tính đầy đủ*: nếu bài toán có lời giải thì thuật toán có đảm bảo tìm ra lời giải đó không? Nếu có, ta nói rằng thuật toán có tính đầy đủ, trong trường hợp ngược lại ta nói thuật toán không đầy đủ.
- *Tính tối ưu*: nếu bài toán có nhiều lời giải thì thuật toán có cho phép tìm ra lời giải tốt nhất không? Tiêu chuẩn tối ưu thường được dùng là giá thành đường đi. Lời giải tối ưu là lời giải có giá thành đường đi nhỏ nhất. Thuật toán luôn đảm bảo tìm ra lời giải tối ưu được gọi là thuật toán có tính tối ưu.

- *Độ phức tạp tính toán*: được xác định bằng khối lượng tính toán cần thực hiện để tìm ra lời giải. Thông thường, khối lượng tính toán được xác định bằng số lượng nút tối đa cần sinh ra trước khi tìm ra lời giải.
- *Yêu cầu bộ nhớ*: được xác định bằng số lượng nút tối đa cần lưu trữ đồng thời trong bộ nhớ khi thực hiện thuật toán.

Các tiêu chuẩn trên được đánh giá tùy theo độ khó (kích thước) của bài toán. Rõ ràng, bài toán có không gian trạng thái lớn hơn sẽ đòi hỏi tính toán nhiều hơn. Trong trường hợp không gian trạng thái được cho tường minh và hữu hạn, độ khó của bài toán được xác định bằng tổng số nút và số liên kết giữa các nút trong đồ thị tìm kiếm. Tuy nhiên, do không gian trạng thái có thể là vô hạn và được cho một cách không tường minh, độ khó của bài toán được xác định qua ba tham số sau:

- Mức độ rẽ nhánh *b*: là số lượng tối đa nút con có thể sinh ra từ một nút cha.
- Độ sâu *d* của lời giải: là độ sâu của lời giải nông nhất, trong đó độ sâu được tính bằng số nút theo đường đi từ gốc tới lời giải.
- Độ sâu *m* của cây tìm kiếm: là độ sâu lớn nhất của mọi nhánh trên cây tìm kiếm.

2.3. TÌM KIẾM KHÔNG CÓ THÔNG TIN (TÌM KIẾM MÙ)

Định nghĩa: Tìm kiếm không có thông tin, còn gọi là tìm kiếm mù (blind, uninformed search) là phương pháp duyệt không gian trạng thái chỉ sử dụng các thông tin theo phát biểu của bài toán tìm kiếm tổng quát trong quá trình tìm kiếm, ngoài ra không sử dụng thêm thông tin nào khác.

Tìm kiếm không có thông tin bao gồm một số thuật toán khác nhau. Điểm khác nhau căn bản của các thuật toán là ở thứ tự mở rộng các nút biên. Sau đây ta sẽ xem xét các thuật toán tìm theo chiều rộng, tìm theo chiều sâu, tìm kiếm sâu dần và một số biến thể của những thuật toán này.

2.3.1. Tìm kiếm theo chiều rộng

Thuật toán tìm kiếm không có thông tin được xem xét trước tiên là *tìm kiếm theo chiều rộng* (Breadth-first search, viết tắt là **BFS**), một dạng tìm kiếm vét cạn.

Nguyên tắc của tìm kiếm theo chiều rộng là trong số những nút biên lựa chọn nút nông nhất (gần nút gốc nhất) để mở rộng. Như vậy, trước hết tất cả các nút có độ sâu bằng 0 (nút gốc) được mở rộng, sau đó tới các nút có độ sâu bằng 1 được mở rộng, rồi tới các nút có độ sâu bằng 2, và tiếp tục như vậy. Ở đây, độ sâu được tính bằng số nút nằm trên đường đi từ nút gốc tới nút đang xét.

Có thể nhận thấy, để thực hiện nguyên tắc tìm kiếm theo chiều rộng, ta cần lựa chọn nút được thêm vào sớm hơn trong danh sách nút biên *O* để mở rộng. Điều này có thể thực hiện dễ dàng bằng cách dùng một hàng đợi FIFO để lưu các nút biên.

Thuật toán tìm theo chiều rộng được thể hiện trên hình 2.7. Khác với thuật toán tìm kiếm tổng quát ở trên, tập nút biên *O* được tổ chức dưới dạng hàng đợi FIFO: các nút mới sinh ra được thêm vào cuối của *O* tại bước 3 của mỗi vòng lặp; nút đầu tiên của *O* sẽ được lấy

ra để mở rộng như tại bước 1 của vòng lặp của thuật toán trên hình vẽ. Bước 2 của vòng lặp kiểm tra điều kiện đích và trả về kết quả trong trường hợp nút lấy ra từ O là nút đích. Thuật toán kết thúc trong hai trường hợp: 1) khi lấy được nút đích từ O; và 2) khi tập O rỗng. Hai trường hợp này tương ứng với hai lệnh return ở trong và ngoài vòng lặp.

Con trỏ ngược: khi mở rộng một nút ta cần sử dụng *con trỏ ngược* để ghi lại nút cha của nút vừa được mở ra. Con trỏ này được sử dụng để tìm ngược lại đường đi về trạng thái xuất phát khi tìm được trạng thái đích. Khi cài đặt thuật toán, mỗi nút được biểu diễn bằng một cấu trúc dữ liệu có chứa một con trỏ ngược trỏ tới nút cha.

Sau khi tìm được nút đích, có thể khôi phục đường đi tới nút đó bằng cách lần theo các con trỏ ngược, bắt đầu từ nút đích.

BFS (Q, S, G, P)

Đầu vào: bài toán tìm kiếm

Đầu ra: trạng thái đích

Khởi tạo: $O \leftarrow S$ // trong thuật toán này, O là hàng đợi FIFO

While (O không rỗng) do

1. Chọn nút đầu tiên n từ O và xóa n khỏi O
2. If $n \in G$, return (đường đi tới n)
3. Thêm $P(n)$ vào cuối O

Return: Không có lời giải

Hình 2.7. Thuật toán tìm kiếm theo chiều rộng

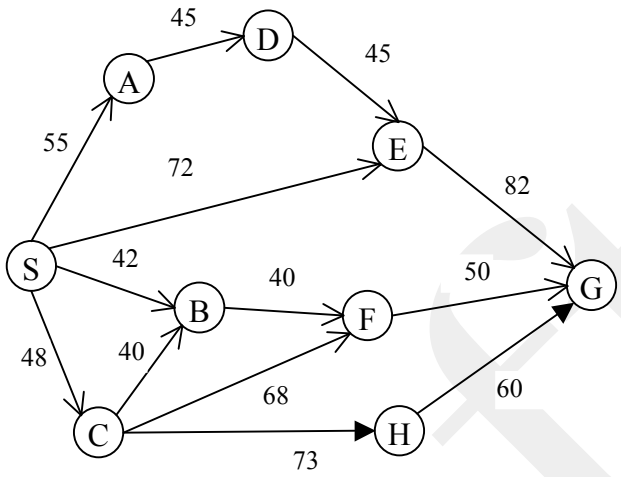
Các cải tiến. Một số cải tiến sau đây có thể sử dụng kết hợp với thuật toán tìm theo chiều rộng vừa trình bầy.

Tránh xem xét lại các nút đã mở rộng. Việc xem xét lại các nút đã mở rộng có thể dẫn tới vòng lặp. Mặc dù vòng lặp không ảnh hưởng tới khả năng tìm ra lời giải của tìm theo chiều rộng, xong việc có vòng lặp và xem xét lại các nút làm tăng độ phức tạp tính toán do phải khảo sát nhiều nút hơn. Vấn đề này có thể giải quyết bằng cách sử dụng tập đóng tương tự trong thuật toán Graph_Search trên hình 2.6. Một trạng thái đã có mặt trong tập đóng hoặc tập biên sẽ không được thêm vào tập biên nữa. Người đọc có thể tự kiểm tra kết quả của thuật toán trong trường hợp có sử dụng kiểm tra tập đóng và tập biên với trường hợp không kiểm tra bằng cách thực hiện thuật toán với ví dụ trên hình 2.8.

Kiểm tra đích trước khi thêm vào tập biên. Trong thuật toán tổng quát, việc kiểm tra điều kiện đích được thực hiện khi nút được lấy ra khỏi O để mở rộng. Thay vào đó, ta có thể kiểm tra trước khi thêm nút vào O. Ưu điểm của cách làm này là giảm bớt số lượng nút cần

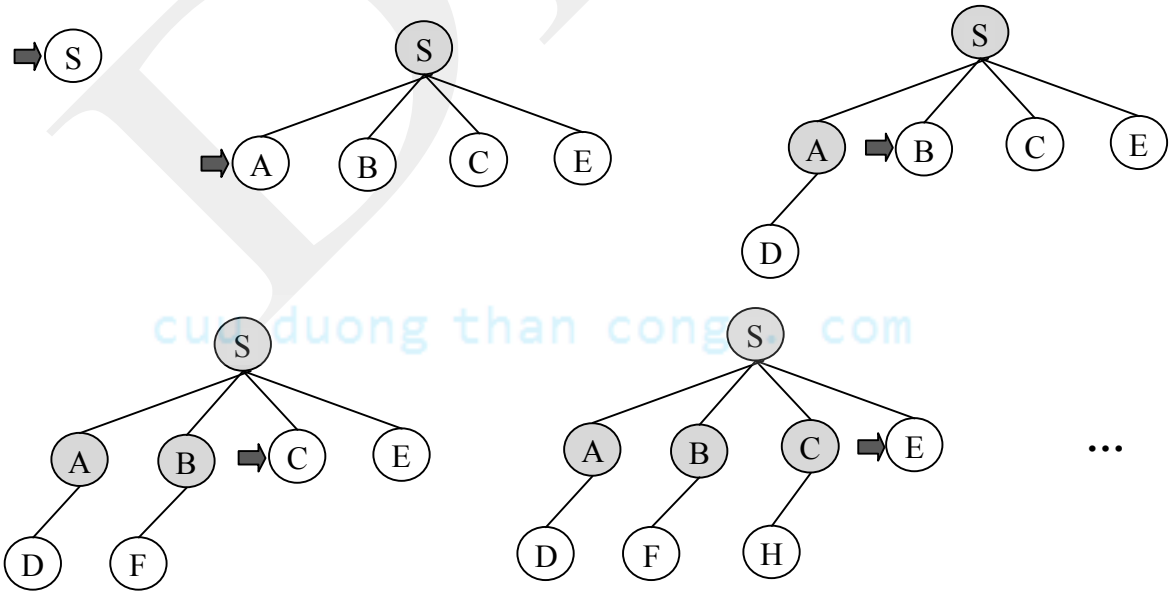
lưu trong nút biên cũng như số nút được mở rộng. Cụ thể về số lượng nút cần mở rộng khi thực hiện kiểm tra đích trước sẽ được trình bầy trong phần tính chất của thuật toán ở bên dưới.

Ví dụ: Xét ví dụ tìm đường đi từ nút S tới nút G trên đồ thị ở hình 2.8. (để đơn giản, ví dụ này sử dụng đồ thị có hướng; quá trình tìm đường đi trên đồ thị vô hướng được thực hiện tương tự, trừ việc từ một nút có thể di chuyển sang nút cha của nút đó)



Hình 2.8. Ví dụ đồ thị cho bài toán tìm đường đi

Một số bước đầu tiên của thuật toán, có sử dụng việc lưu và kiểm tra tập nút đóng, được thể hiện dưới dạng các cây tìm kiếm như trên hình 2.9. *Lưu ý*: để thống nhất trong việc trình bầy, trong số các nút có vai trò giống nhau, tức là có cùng độ sâu, nút đứng trước trong bảng chữ cái sẽ được mở rộng trước. Quy tắc này không được quy định trong thuật toán và chỉ để tiện cho trình bầy.



Hình 2.9. Một số cây tìm kiếm sinh ra khi tìm kiếm theo chiều rộng. Các cây được thể hiện theo thứ tự từ trái sang phải, từ trên xuống dưới. Nút mở rộng tiếp theo được đánh dấu bằng mũi tên

Kết quả thực hiện các bước của thuật toán cũng có thể theo dõi qua thứ tự mở rộng các nút và nội dung danh sách tập nút biên. Dưới đây là minh họa cho thứ tự mở rộng nút và tập biên cho ví dụ trên hình 2.8 (có sử dụng tập nút đóng). Để tiện cho trình bầy, ta sẽ quy định phía bên trái là đầu của hàng đợi *O* và phía bên phải là cuối của hàng đợi *O*. Như vậy các nút được thêm vào từ phía bên phải và lấy ra từ bên trái. Con trỏ tới nút cha sẽ được viết dưới dạng chỉ số bên cạnh mỗi nút

Nút được mở rộng	Tập biên O (hàng đợi FIFO trong trường hợp này)
	S
S	A_S , B_S , C_S , E_S
A_S	B_S , C_S , E_S, D_A
B_S	C_S , E_S, D_A, F_B
C_S	E_S , D_A, F_B , H_C
E_S	D_A , F_B, H_C , G_E
D_A	F_B, H_C , G_E
F_B	H_C , G_E
H_C	G_E
G_E	Đích

Sau đó, sử dụng con trỏ ngược, theo đó E là nút cha của G, S là nút cha của E, ta tìm được đường đi G ← E ← S.

Tính chất của tìm theo chiều rộng:

Đối chiếu với các tiêu chuẩn ở trên, tìm kiếm theo chiều rộng có những tính chất sau:

- Thuật toán có tính đầy đủ, tức là nếu bài toán có lời giải, tìm kiếm theo chiều rộng đảm bảo tìm ra lời giải. Thật vậy, nếu lời giải nằm ở độ sâu hữu hạn *d* thì thuật toán sẽ đạt tới lời giải đó sau khi đã khảo sát hết các nút nông hơn, trừ khi yếu tố rẽ nhánh *b* là vô hạn. Tìm theo chiều rộng là tìm kiếm vét cạn, trong đó các nút có độ sâu nhỏ hơn được xem xét trước.
- Tính tối ưu: thuật toán đảm bảo tìm ra lời giải có độ sâu nhỏ nhất. Tuy nhiên, trong trường hợp giá thành đường đi giữa các nút không bằng nhau thì điều này chưa đảm bảo tìm ra đường đi ngắn nhất. Cụ thể, trong ví dụ trên, thuật toán tìm ra đường đi SEG có độ sâu bằng 2 (nhỏ nhất) nhưng có độ dài 154, trong khi đường đi ngắn nhất SBFG có độ dài 132.

- Độ phức tạp tính toán: với mức độ rẽ nhánh là b và độ sâu lời giải d, thuật toán sinh ra $O(b^{d+1})$ nút trước khi tìm ra lời giải hay $O(b^d)$ nút nếu kiểm tra đích trước khi thêm nút vào tập biên. Độ phức tạp này là lớn và tăng rất nhanh khi b và d tăng.

Giả sử rằng, mỗi trạng thái khi được phát triển sẽ sinh ra b trạng thái kề. Như vậy từ nút gốc sẽ sinh ra b nút với độ sâu 1, các nút này lại sinh ra b^2 nút với độ sâu 2, và tiếp tục như vậy. Giả sử nút đích của bài toán nằm ở độ sâu d. Trong trường hợp xấu nhất, nút đích nằm cuối cùng trong số các nút ở độ sâu này và do vậy ta cần mở rộng tất cả nút ở độ sâu d trước khi tìm ra đích, tức là sinh ra b^{d+1} nút ở độ sâu $d+1$. Như vậy, tổng số nút cần mở rộng để tìm ra nút đích là (tính cả nút gốc):

$$1 + b + b^2 + \ldots + b^{d+1} = O(b^{d+1})$$

Nếu tiến hành kiểm tra điều kiện đích trước khi thêm vào tập biên như đề cập ở trên, ta sẽ không phải sinh ra các nút ở độ sâu $d + 1$ và do vậy số nút cần sinh ra chỉ còn là $O(b^d)$.

- Yêu cầu bộ nhớ: thuật toán cần lưu $O(b^{d+1})$ nút trong tập biên sau khi đã mở rộng tất cả các nút ở độ sâu d. Nếu sử dụng tập các nút đóng thì tập này cần lưu $O(b^d)$ nút. Như vậy độ phức tạp bộ nhớ của tìm kiếm rộng là $O(b^{d+1})$.

Như vậy, ưu điểm của tìm theo chiều rộng là tính đầy đủ và tối ưu nếu giá thành đường đi như nhau. Nhược điểm của thuật toán này là độ phức tạp tính toán lớn và yêu cầu về bộ nhớ lớn. Trong hai nhược điểm sau, độ phức tạp về bộ nhớ lớn là nghiêm trọng hơn do không thể kiếm được máy tính có bộ nhớ đủ lớn để chạy thuật toán, trong khi ta có thể đợi thêm thời gian để chờ thuật toán chạy xong nếu thời gian chạy không quá lâu. Trên thực thế, thuật toán tìm theo chiều rộng chỉ có thể sử dụng cho các bài toán có kích thước rất nhỏ (b và d không quá 10).

2.3.2. Tìm kiếm theo giá thành thống nhất

Trong trường hợp giá thành di chuyển giữa hai nút là không bằng nhau giữa các cặp nút, tìm theo chiều rộng không cho tìm ra lời giải có giá thành nhỏ nhất và do vậy không tối ưu. Để tìm ra đường đi ngắn nhất trong trường hợp này cần sử dụng một biến thể của tìm theo chiều rộng có tên gọi là tìm *kiếm theo giá thành thống nhất* (Uniform-Cost-Search).

Phương pháp: Thuật toán tìm theo giá thành thống nhất chọn nút n có giá thành đường đi $g(n)$ nhỏ nhất để mở rộng trước thay vì chọn nút nông nhất như trong tìm theo chiều rộng, trong đó $g(n)$ là giá thành đường đi từ nút xuất phát tới nút n.

Thuật toán: được biến đổi từ tìm kiếm theo chiều rộng bằng cách thay ba bước trong vòng lặp While như sau:

1. Chọn nút n có giá thành $g(n)$ nhỏ nhất thuộc O và xóa n khỏi O
2. If $n \in G$, return (đường đi tới n)
3. Thêm $P(n)$ và giá thành đường đi tới n $(g(n))$ vào O

Cách tránh vòng lặp được thực hiện như trong thuật toán Graph-search, tuy nhiên nếu nút đã có trong tập biên thì ta lưu lại bản có giá thành nhỏ hơn.

Ví dụ: với ví dụ trên hình 2.8, kết quả các bước tìm kiếm theo giá thành thống nhất được thể hiện dưới đây, với giá trị $g(n)$ được cho trong ngoặc sau mỗi nút.

Nút được mở rộng	Tập biên O
	S(0)
S	A_S (55) , B_S (42), C_S (48), E_S (72)
B_S	A_S (55) , C_S (48), E_S (72), F_B (82)
C_S	A_S (55) , E_S (72), F_B (82), H_C (121)
A_S	E_S (72), F_B (82), H_C (121), D_A (100)
E_S	F_B (82), H_C (121), D_A (100), G_E (154)
F_B	H_C (121), D_A (100), G_F (132)
D_A	H_C (121), G_F (132)
H_C	G_F (132)
G_F	Đích

The con trỏ ngược, ta tìm được đường đi G ← F ← B← S với giá thành 132.

Thuật toán tìm kiếm theo giá thành thống nhất có một trường hợp riêng là **thuật toán Dijkstra**. Một trong những điểm khác nhau lớn nhất với thuật toán Dijkstra là thuật toán Dijkstra tìm đường đi ngắn nhất từ nút gốc tới tất cả các nút còn lại thay vì chỉ tìm đường tới nút đích như trong trường hợp tìm theo giá thống nhất.

Thuật toán tìm kiếm này luôn cho lời giải tối ưu, tức là lời giải với đường đi có giá nhỏ nhất. Do thuật toán không lựa chọn nút để mở rộng dựa trên độ sâu mà dựa trên giá thành nên không thể phân tích độ phức tạp tính toán cũng như yêu cầu bộ nhớ của thuật toán dựa trên tham số *b* và *d*. Trong trường hợp giá thành mọi chuyển động bằng nhau, tìm kiếm theo giá thống nhất sẽ giống với tìm theo chiều rộng.

2.3.3. Tìm kiếm theo chiều sâu

Thuật toán tìm kiếm mù được biết đến nhiều tiếp theo là *tìm theo chiều sâu* (Depth-First-Search, viết tắt là **DFS**).

Nguyên tắc của tìm kiếm theo chiều sâu là lựa chọn trong số những nút biên nút sâu nhất (xa nút gốc nhất) để mở rộng trước. Như vậy, thay thì khảo sát tất cả các nhánh của cây tìm kiếm một lúc như trong tìm theo chiều rộng, thuật toán tìm sâu sẽ di chuyển theo một nhánh trong cây tìm kiếm cho tới nút sâu nhất, tức là nút không có nút con, trước khi chuyển sang nhánh tiếp theo.

Để thực hiện nguyên tắc trên, ta cần lựa chọn nút được thêm vào sau cùng trong tập nút biên *O* để mở rộng. Điều này có thể thực hiện dễ dàng bằng cách dùng một ngăn xếp để lưu các nút biên, các nút được thêm vào và lấy ra theo nguyên lý LIFO (vào sau ra trước). Thuật

toán tìm theo chiều sâu cũng có thể thực hiện bằng cách đệ quy và quay lui. Tuy nhiên, trong tài liệu này, ta không sẽ không xem xét phương án sử dụng đệ quy và quay lui.

Thuật toán tìm kiếm theo chiều sâu được thể hiện trên hình 2.10, trong đó tập biên *O* được triển khai dưới dạng ngăn xếp LIFO. Các nút mới sinh ra được thêm vào đầu ngăn xếp *O* ở bước 3 của vòng lặp chính. Nút để mở rộng cũng được lấy ra từ đầu của *O* ở bước 1 của vòng lặp chính. Tương tự tìm theo chiều rộng, mỗi nút cần có con trỏ ngược về nút cha. Con trỏ ngược được sử dụng để khôi phục lại đường đi khi thuật toán đã tìm ra nút đích. Thuật toán kết thúc và trả về kết quả tại bước 2 của vòng lặp nếu nút lấy ra khỏi O là nút đích; hoặc thuật toán kết thúc (bằng lệnh return ở dưới cùng) mà không tìm được kết quả nếu tập O rỗng.

DFS(Q, S, G, P)

Đầu vào: bài toán tìm kiếm

Đầu ra: (đường đi tới) trạng thái đíchC

Khởi tạo: O←S // O được tổ chức như ngăn xếp LIFO

While(O ≠ Ø) do

1. Chọn nút n đầu tiên của O và xóa n khỏi O
2. If n ∈ G, return (đường đi tới n)
3. Thêm P(n) vào đầu O

Return: Không có lời giải

Hình 2.10. Thuật toán tìm kiếm theo chiều sâu

Tránh các nút lặp:

Thuật toán trên hình 2.10 có thể dẫn tới vòng lặp. Ví dụ, trong bài toán tìm đường trên bản đồ, thuật toán có thể quay về vị trí trước đó và lặp đi lặp lại chuyển động giữa hai trạng thái liền nhau. Khác với tìm theo chiều rộng, tìm theo chiều sâu sẽ lặp vô hạn mà không tìm được lời giải.

Để tránh vòng lặp khi tìm kiếm theo chiều sâu, có thể sử dụng một trong hai phương pháp. *Phương pháp thứ nhất* giống như phương pháp sử dụng trong thuật toán Graph_Search, tức là duy trì danh sách nút đóng và nhớ tất cả các nút đã mở rộng vào đây. Một nút mới sinh ra chỉ được thêm vào nút biên nếu nó chưa xuất hiện trong tập đóng và tập biên. Phương pháp này đòi hỏi nhiều bộ nhớ để duy trì tập nút đóng.

Phương pháp thứ hai kiểm tra xem nút mới có thuộc đường đi từ gốc tới nút hiện thời không, nếu có, nút mới sẽ không được thêm vào tập nút biên. Cách này không đòi hỏi bộ nhớ để lưu tập đóng, tuy nhiên chỉ cho phép tránh vòng lặp vô hạn và không cho phép giảm khối lượng tính toán trong trường hợp nhiều nhánh của đồ thị tìm kiếm có những phần trùng nhau.

Ví dụ: kết quả sau các bước lặp của thuật toán tìm theo chiều sâu cho bài toán tìm đường từ S tới G trên đồ thị hình 2.8 được thể hiện dưới đây dưới dạng các nút được mở rộng và nội dung tập biên O sau mỗi vòng lặp. Cũng như trong các ví dụ trước, nếu hai nút cùng độ sâu thì nút đứng trước trong bảng chữ cái được mở rộng trước. Việc tránh vòng lặp được thực hiện theo phương pháp thứ hai. Đỉnh ngăn xếp nằm phía trái và đáy ngăn xếp nằm bên phải trong mỗi dòng.

Nút được mở rộng	Tập biên O (ngăn xếp LIFO trong trường hợp này)
	S
S	A_S , B_S , C_S , E_S
A_S	D_A , B_S , C_S , E_S
D_A	E_D, B_S , C_S , E_S
E_D	B_S , C_S , E_S
G_E	Đích

Theo con trỏ ngược, ta tìm được đường đi G ← E ← D ← A ← S với độ sâu bằng 4 và độ dài 227.

Tính chất thuật toán tìm theo chiều sâu:

- Nếu không gian trạng thái là hữu hạn thì thuật toán là đầy đủ. Ngược lại, nếu không gian trạng thái là vô hạn thì thuật toán là không đầy đủ do có thể di chuyển theo một đường đi không chứa nút đích và có độ sâu vô hạn (cứ đi theo nhánh không đúng mãi mà không chuyển sang nhánh khác được).
- Thuật toán không tối ưu: thuật toán có thể mở rộng những nhánh dẫn tới lời giải không tối ưu trước, đặc biệt trong trường hợp có nhiều lời giải.
- Độ phức tạp của thuật toán ở trường hợp xấu nhất là O(b^m) với m là độ sâu tối đa của cây tìm kiếm. Trong trường hợp không gian trạng thái là vô hạn thì *m* là vô hạn. Trên thực tế DFS tìm ra lời giải nhanh hơn BFS, đặc biệt nếu tồn tại nhiều lời giải.
- Bộ nhớ cần nhớ tối đa *b*m* (mỗi mức chỉ nhớ *b* nút, với tối đa *m* mức), như vậy độ phức tạp về bộ nhớ của thuật toán này chỉ là O(*bm*). Để đánh giá độ phức tạp không gian của tìm kiếm theo độ sâu ta có nhận xét rằng, khi ta phát triển một đỉnh u trên cây tìm kiếm theo độ sâu, ta chỉ cần lưu các đỉnh chưa được phát triển mà chúng là các đỉnh con của các đỉnh nằm trên đường đi từ gốc tới đỉnh u. Như vậy đối với cây tìm kiếm có nhân tố nhánh *b* và độ sâu lớn nhất là *m*, ta chỉ cần lưu ít hơn *b*m* đỉnh. Ưu cầu bộ nhớ so với tìm theo chiều rộng là ưu điểm nổi bật nhất của tìm theo chiều sâu.

2.3.4. Tìm kiếm sâu dần

Tìm kiếm sâu dần (Iterative Deepening Search, viết tắt là IDS) là một phương pháp tìm

kiếm dựa trên tìm theo chiều sâu nhưng có tính đầy đủ và cho phép tìm ra lời giải tối ưu.

Như đã nói ở trên, mặc dù có ưu điểm rất lớn là không yêu cầu nhiều bộ nhớ như tìm theo chiều rộng, tìm theo chiều sâu có thể rất chậm hoặc bế tắc nếu mở rộng những nhánh sâu (vô tận) không chứa lời giải. Để khắc phục, có thể sử dụng kỹ thuật *tìm kiếm với độ sâu hữu hạn*: tìm kiếm theo chiều sâu nhưng không tiếp tục phát triển một nhánh khi đã đạt tới một độ sâu nào đó, thay vào đó, thuật toán chuyển sang phát triển nhánh khác. Nói cách khác, các nút ở độ sâu giới hạn sẽ không được mở rộng tiếp. Thuật toán này có yêu cầu bộ nhớ nhỏ tương tự tìm theo chiều sâu, trong khi chắc chắn tìm được lời giải.

Kỹ thuật này có thể sử dụng trong trường hợp có thể dự đoán được độ sâu của lời giải bằng cách dựa trên đặc điểm bài toán cụ thể. Chẳng hạn, nếu ta biết rằng ở miền bắc và bắc trung bộ không có nhiều hơn 15 thành phố thì khi tìm đường từ Hà nội vào Vinh có thể giới hạn chiều sâu tìm kiếm bằng 15. Một số bài toán khác cũng có thể dự đoán trước giới hạn độ sâu như vậy. Trong trường hợp ta biết chính xác độ sâu của lời giải, thuật toán sẽ cho lời giải tối ưu (lời giải nông nhất).

Tuy nhiên, trong trường hợp chung, ta thường không có trước thông tin về độ sâu của lời giải. Trong trường hợp như vậy có thể sử dụng phương pháp tìm kiếm sâu dần. Thực chất tìm kiếm sâu dần là tìm kiếm với độ sâu hữu hạn, trong đó giới hạn độ sâu được khởi đầu bằng một giá trị nhỏ, sau đó tăng dần cho tới khi tìm được lời giải.

Phương pháp: Tìm theo DFS những không bao giờ mở rộng các nút có độ sâu quá một giới hạn nào đó. Giới hạn độ sâu được bắt đầu từ 0, sau đó tăng lên 1, 2, 3 v.v. cho đến khi tìm được lời giải.

Thuật toán tìm kiếm sâu dần thể hiện trên hình 2.11, trong đó tìm kiếm sâu được lặp lại, tại mỗi bước lặp, độ sâu được giới hạn bởi biến C. Sau mỗi vòng lặp, giá trị của C được tăng thêm một đơn vị và thuật toán xây dựng lại cây tìm kiếm từ đầu. Với mỗi giá trị của C, thuật toán tiến hành tìm kiếm theo chiều sâu nhưng không thêm vào ngăn xếp *O* các nút có độ sâu lớn hơn *C*. Việc kiểm tra này được thực hiện ở bước c) tại vòng lặp trong của thuật toán. Lưu ý rằng trong trường hợp này khó xác định điều kiện kết thúc của thuật toán trong trường hợp không tìm được lời giải.


```
IDS(Q, S, G, P)
Đầu vào: thuật toán tìm kiếm
Đầu ra: trạng thái đích
Khởi tạo:  O←S (O: ngăn xếp LIFO như trong DFS)
           C ← 0 (C là giới hạn độ sâu tìm kiếm)
---------------------------------
While (điều kiện kết thúc chưa thoả mãn) do
   While(O ≠ Ø) do
          //Thực hiện 3 bước tương tự tìm kiếm sâu
```

a) Lấy nút đầu tiên n từ đầu của O

b) If (n ∈ G) then return (đường đi tới n)

c) If (độ sâu của n nhỏ hơn hoặc bằng C) then

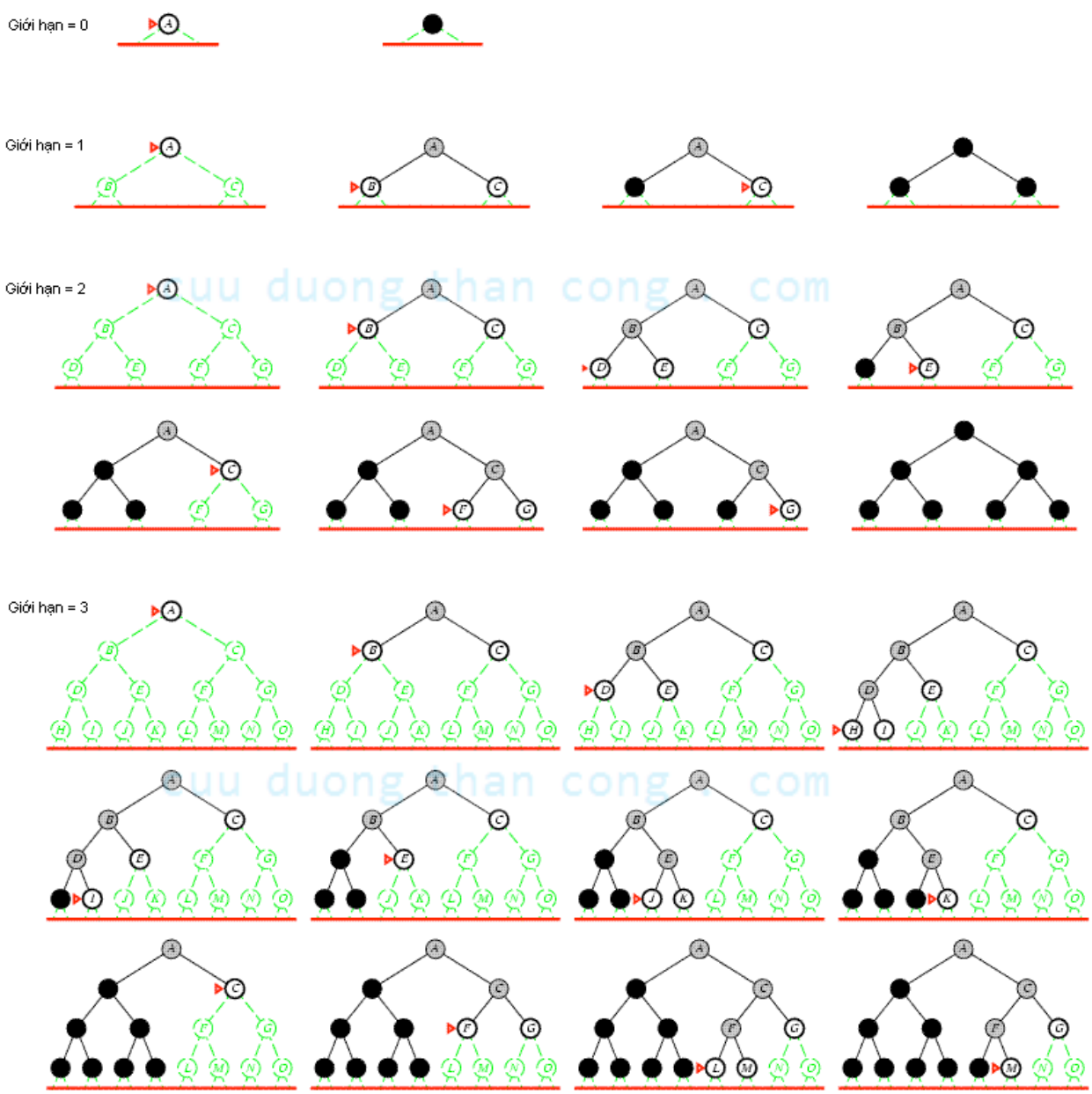
- Thêm P(n) vào đầu O

C++, O←S

Return: không tìm được lời

Hình 2.11. Thuật toán tìm kiếm sâu dần

Cây tìm kiếm trong trường hợp tìm sâu dần được minh họa trên hình 2.12.



Hình 2.12. Cây tìm kiếm theo thuật toán tìm kiếm sâu dần

Ví dụ. Kết quả sau các bước lặp của thuật toán tìm kiếm sâu dần cho bài toán tìm đường từ S tới G trên đồ thị hình 2.8 được thể hiện dưới đây. Các thông tin sau mỗi bước lặp bao gồm giới hạn độ sâu C, nút được mở rộng, và nội dung tập biên O. Cũng như trong các ví dụ trước, nếu hai nút cùng độ sâu thì nút đứng trước trong bảng chữ cái được mở rộng trước. Việc tránh vòng lặp được thực hiện theo phương pháp thứ hai. Đỉnh ngăn xếp nằm phía trái và đáy ngăn xếp nằm bên phải trong mỗi dòng.

Nút được mở rộng	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	A_S , B_S , C_S , E_S
A_S	B_S , C_S , E_S
B_S	C_S , E_S
C_S	E_S
E_S	
C = 2	
	S
S	A_S , B_S , C_S , E_S
A_S	B_S , C_S , E_S, D_A
B_S	C_S , E_S, D_A, F_B
C_S	E_S, D_A, F_B, B_C, F_C, H_C
E_S	D_A, F_B, B_C, F_C, H_C, G_C
D_A	F_B, B_C, F_C, H_C, G_C
F_B	B_C, F_C, H_C, G_C
B_C	F_C, H_C, G_C
F_C	H_C, G_C
H_C	G_C
G_C	Đích

Theo con trỏ ngược, ta tìm được đường đi G ← E ← S với độ sâu bằng 2 và độ dài 154. Đây là đường đi nông nhất, mặc dù không phải là đường đi ngắn nhất.

Tính chất của tìm sâu dần:

d) Thuật toán đầy đủ, tức là đảm bảo tìm ra lời giải nếu có. Thật vậy, tương tự tìm theo chiều rộng, tìm sâu dần xem xét toàn bộ các nút ở một độ sâu, bắt đầu từ độ sâu 0, trước khi chuyển sang xem xét toàn bộ nút ở độ sâu tiếp theo. Do vậy, thuật toán sẽ tìm được lời giải khi xem xét hết các nút ở độ sâu *d* (*d* là độ sâu lời giải).

e) Tương tự tìm theo chiều rộng, *thuật toán tối ưu nếu giá thành đường đi giữa hai nút giống nhau*, tức là nếu có nhiều lời giải, có thể tìm ra được lời giải nông nhất. Thật vậy, thuật toán sẽ xem xét toàn bộ các nút có độ sâu bằng 0, trước khi xem xét các nút có độ sâu bằng 1 v.v. Do vậy, trong các lời giải có độ sâu khác nhau, thuật toán sẽ xem xét lời giải nông hơn trước.

f) Yêu cầu bộ nhớ nhỏ. Cụ thể, yêu cầu bộ nhớ là O(*bd*). Thực chất, tại mỗi bước lặp, thuật toán thực hiện tìm kiếm theo chiều sâu nên yêu cầu bộ nhớ tương tự tìm theo chiều sâu, tức là O(*bd*).

g) Độ phức tạp tính toán O(b^d).Trong tìm kiếm sâu lặp, ta phải phát triển lặp lại nhiều lần cùng một trạng thái. Điều đó làm cho ta có cảm giác rằng, tìm kiếm sâu lặp lãng phí nhiều thời gian. Thực ra thời gian tiêu tốn cho phát triển lặp lại các trạng thái là không đáng kể so với thời gian tìm kiếm theo bề rộng. Thật vậy, mỗi lần gọi thủ tục tìm kiếm sâu hạn chế tới mức d, nếu cây tìm kiếm có nhân tố nhánh là b, thì số đỉnh cần phát triển là:

$$1 + b + b^2 + ... + b^d$$

Nếu lời giải ở độ sâu *d*, thì trong tìm kiếm sâu lặp, ta phải gọi thủ tục tìm kiếm sâu hạn chế với độ sâu lần lượt là 0, 1, 2, ..., d. Do đó các đỉnh ở mức 1 phải phát triển lặp d lần, các đỉnh ở mức 2 lặp d-1 lần, ..., các đỉnh ở mức d lặp 1 lần. Như vậy tổng số đỉnh cần phát triển trong tìm kiếm sâu lặp là:

$$(d + 1) + db + (d - 1)b^2 + ... + 2b^{d-1} + b^d$$

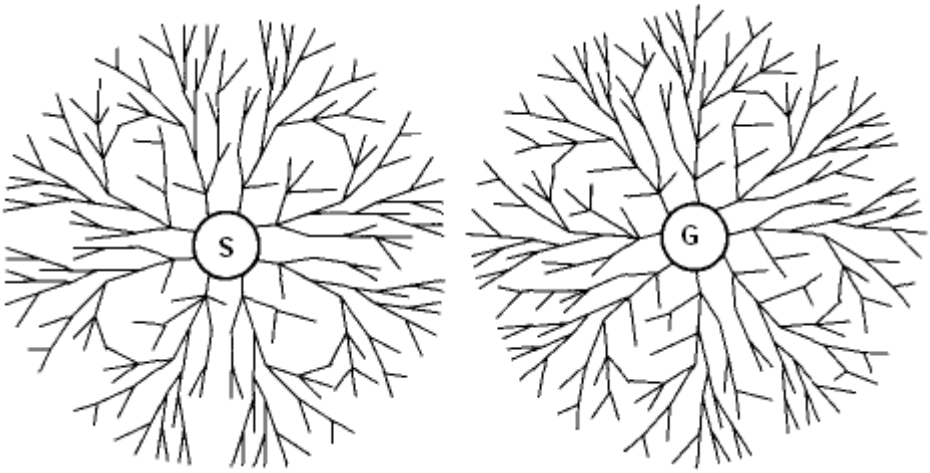
Do đó thời gian tìm kiếm sâu dần là O(b^d) tương tự tìm kiếm theo chiều rộng. Trên thực tế, do phải xem xét nhiều lần các nút ở độ sâu nhỏ, nên độ phức tạp tính toán của tìm sâu dần lớn hơn tìm kiếm theo chiều rộng nhưng không lớn hơn quá nhiều.

2.3.5.Tìm theo hai hướng

Trong các phương pháp tìm kiếm ở trên, quá trình tìm kiếm bắt đầu từ nút xuất phát và kết thúc khi đạt tới nút đích. Do tính chất đối xứng của đường đi, quá trình tìm kiếm cũng có thể bắt đầu từ nút đích và tìm tới nút xuất phát. Ngoài ra, quá trình tìm kiếm có thể xuất phát đồng thời từ cả nút xuất phát và nút đích, xây dựng đồng thời hai cây tìm kiếm. Quá trình tìm kiếm kết thúc khi hai cây tìm kiếm có một nút chung (hình 2.13).

Tìm theo hai hướng (Bidirectional Search) là phương pháp tìm kiếm trong đó ta đồng thời xây dựng hai cây tìm kiếm có nút gốc là trạng thái xuất phát và trạng thái đích.

Phương pháp. Tìm kiếm bắt nguồn từ nút xuất phát và nút đích. Như vậy, sẽ tồn tại hai cây tìm kiếm, một cây có gốc là nút xuất phát và một cây có gốc là nút đích. Tìm kiếm kết thúc khi có nút lá của cây này trùng với nút lá của cây kia.



Hình 2.13: Cây tìm kiếm trong trường hợp tìm theo hai hướng

Chú ý:

- Khi tìm theo hai hướng cần sử dụng tìm theo chiều rộng. Việc tìm theo chiều sâu có thể không cho phép tìm ra lời giải nếu hai cây tìm kiếm phát triển theo hai nhánh không gặp nhau.
- Để tìm theo hai hướng cần viết thêm một hàm chuyển động ngược là P(x): tập các nút con của x và D(x): tập các nút cha của x. Nút càng gần nút xuất phát càng được coi là tổ tiên

Tính chất của tìm theo hai hướng:

- Việc kiểm tra xem nút lá này có trùng với nút lá kia đòi hỏi tương đối nhiều thời gian. Thật vậy, ở độ sâu d, mỗi cây tìm kiếm sẽ sinh ra b^d nút lá. Như vậy cần so sánh mỗi nút lá của cây theo hướng này với b^d nút lá của hướng ngược lại.
- Độ phức tạp tính toán: do gặp nhau ở giữa nên chiều sâu mỗi cây là d/2. Theo tính toán đối với tìm theo chiều rộng, độ phức tạp tính toán khi đó là $O(b^{d/2})$. Như vậy mặc dù việc kiểm tra các nút trùng nhau gây tốn thời gian nhưng số lượng nút cần mở rộng của cả hai cây giảm đáng kể so với tìm theo một chiều.

2.4. TÌM KIẾM CÓ THÔNG TIN

Đối với tìm kiếm mù, các nút biên lần lượt được mở rộng theo một thứ tự nhất định mà không tính tới việc ưu tiên các nút có khả năng dẫn tới lời giải nhan hơn. Kết quả của việc tìm kiếm như vậy là việc di chuyển trong không gian tìm kiếm không có định hướng, dẫn tới phải

xem xét nhiều trạng thái. Đối với những bài toán thực tế có không gian trạng thái lớn, tìm kiếm mù thường không thực tế do có độ phức tạp tính toán và yêu cầu bộ nhớ lớn.

Để giải quyết vấn đề trên, *chiến lược tìm kiếm có thông tin (Informed search) hay còn được gọi là tìm kiếm heuristic sử dụng thêm thông tin từ bài toán để định hướng tìm kiếm, cụ thể là lựa chọn thứ tự mở rộng nút theo hướng mau dẫn tới đích hơn*.

Nguyên tắc chung của tìm kiếm có thông tin là sử dụng một hàm $f(n)$ để đánh giá độ "tốt" tiềm năng của nút n, từ đó chọn nút n có hàm f tốt nhất để mở rộng trước. Thông thường, độ tốt được đo bằng giá thành đường đi tới đích, do vậy nút có hàm $f(n)$ nhỏ được ưu tiên mở rộng trước.

Trên thực tế, việc xây dựng hàm $f(n)$ phản ánh chính xác độ tốt của nút thường không thực hiện được, thay vào đó ta chỉ có thể ước lượng hàm $f(n)$ dựa vào thông tin có được từ bài toán. Như sẽ thấy trong các phần sau, hàm $f(n)$ thường chứa một thành phần là hàm heuristic $h(n)$, là hàm ước lượng khoảng cách từ nút n tới đích.

Trong phần này ta sẽ xem xét hai chiến lược tìm kiếm có thông tin, đó là *tìm kiếm tham lam* và *tìm kiếm A**.

2.4.1. Tìm kiếm tham lam

Ý tưởng của *tìm kiếm tham lam* (Greedy Search) là chọn trong tập nút biên nút có khoảng cách tới đích nhỏ nhất để mở rộng. Lý do làm như vậy là việc mở rộng nút gần đích có xu hướng dẫn tới lời giải nhanh hơn.

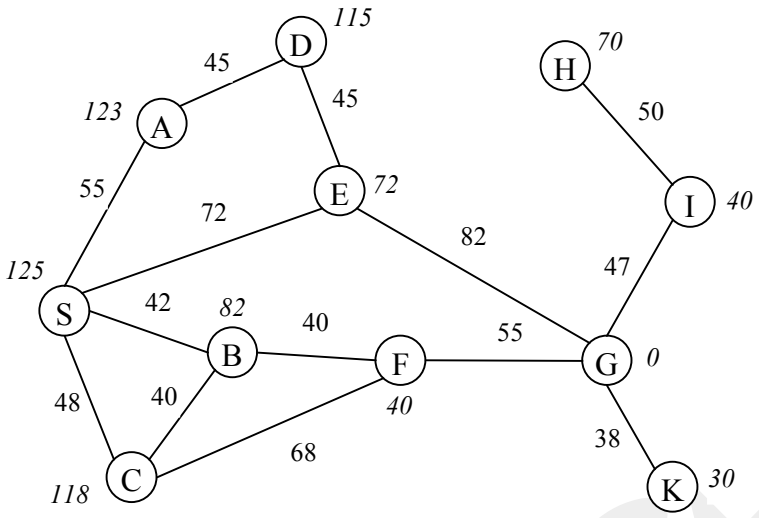
Trong phương pháp này, để đánh giá độ tốt của một nút, ta sử dụng hàm đo giá thành đường đi từ nút đó tới đích. Tuy nhiên, do không biết được chính xác giá thành đường đi từ một nút tới đích, ta chỉ có thể ước lượng giá trị này. Hàm ước lượng độ tốt, hay giá thành đường đi từ một nút n tới địch gọi là *hàm heuristic* và ký hiệu $h(n)$. Như vậy, đối với thuật toán tham lam, ta có $f(n) = h(n)$.

Do hàm $h(n)$ chỉ là hàm ước lượng giá thành đường đi tới đích nên có thể nói rằng tìm kiếm tham lam mở rộng nút trông có vẻ gần đích nhất trước các nút khác. Thuật toán được gọi là "tham lam" do tại mỗi bước, thuật toán cố gắng tiến về đích nhiều nhất có thể. Như ta sẽ thấy dưới đây, do thuật toán chỉ cố gắng làm tốt nhất tại mỗi bước mà không tính tới tổ hợp các bước, đường đi tìm được có thể không phải là đường đi ngắn nhất.

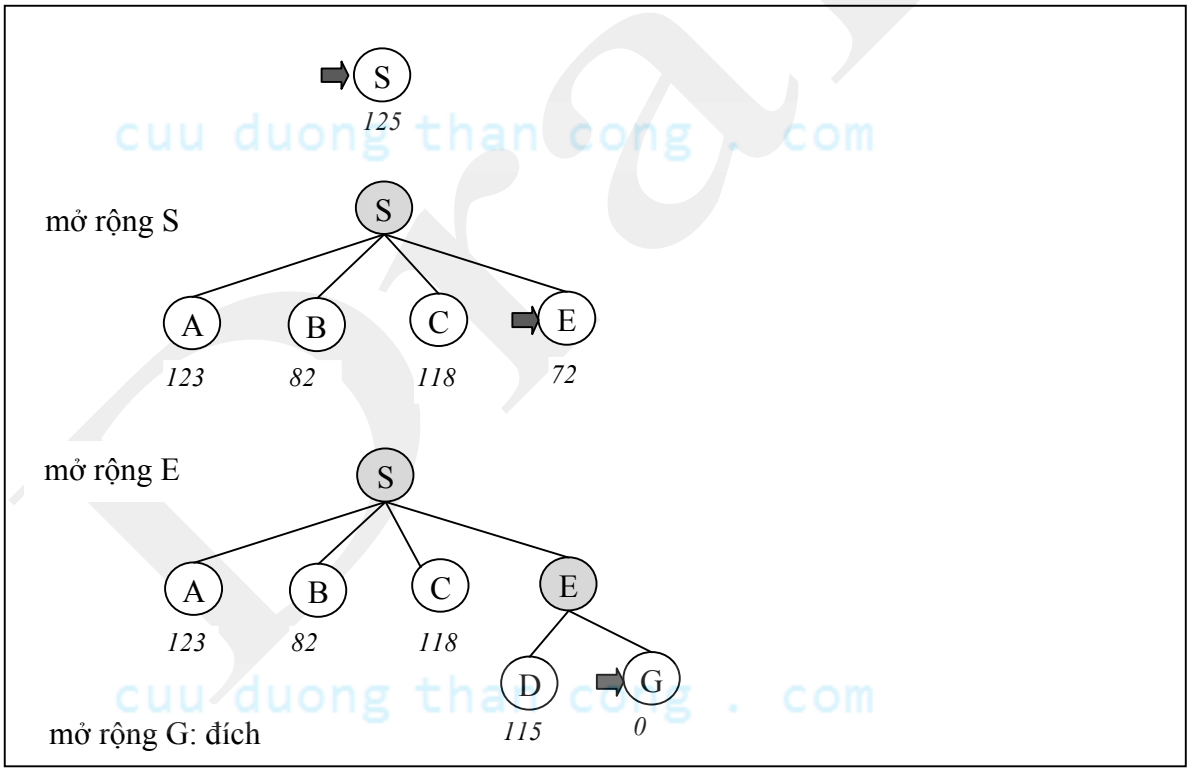
Hàm heuristic được xây dựng dựa trên thông tin có được về bài toán. Hàm này phải thỏa mãn hai điều kiện: thứ nhất, đây là hàm không âm ($h(n) \geq 0$), và thứ hai, nếu n là nút đích thì $h(n) = 0$.

Như vậy, *tìm kiếm tham lam sử dụng hàm heuristic h(n) để ước lượng khoảng cách từ các nút tới đích và thuật toán luôn mở rộng nút n có hàm h(n) nhỏ nhất trong số các nút biên.*

Ví dụ. Khi tìm đường đi giữa hai nút bản đồ, hàm heuristic cho một nút có thể tính bằng khoảng cách theo đường chim bay giữa thành phố đó với nút đích cần đến. Để minh họa, ta xét bài toán tìm đường đi từ nút S tới nút G trên đồ thị ở hình 2.14. Trên đồ thị này, khoảng cách thực giữa hai nút bất kỳ được cho dưới dạng số bên cạnh cung nối hai nút. Khoảng cách tính theo đường chim bay giữa một nút bất kỳ và nút đích G được cho bằng số in nghiêng bên cạnh nút đó.



Hình 2.14. Ví dụ tìm đường đi trên đồ thị từ S tới G. Khoảng cách thực được cho trên các cung. Khoảng cách tính theo đường chim bay từ mỗi nút tới nút đích G được cho bằng số in nghiêng bên cạnh các nút.



Hình 2.15. Minh họa hoạt động của thuật toán tìm kiếm tham lam

Xuất phát từ S, thứ tự mở rộng các nút của thuật toán tham lam được thể hiện trên hình 2.15. sau khi mở rộng nút S, nút được chọn tiếp theo là E do nút này có giá trị hàm heuristic (khoảng cách theo đường chim bay) bằng 72, nhỏ nhất trong số 4 nút A, B, C, E. Sau khi mở rộng nút E, nút được chọn tiếp theo là G do nút này có giá trị hàm heuristic nhỏ nhất (bằng 0)

trong số 5 nút biên A, B, C, D, G. Kiểm tra G cho thấy đây là đích. Đường đi tìm được là S→E→G.

Đặc điểm của tìm kiếm tham lam:

- Không có tính đầy đủ do có khả năng tạo thành vòng lặp vô hạn ở một số nút. Chẳng hạn, nếu trong ví dụ ở trên cần tìm đường đi từ nút I tới nút D. Thuật toán sẽ chọn nút tiếp theo là H do nút này gần D hơn G theo đường chim bay. Mở rộng H, thuật toán lại thêm I vào tập biên và do I gần D hơn G, thuật toán lại chọn I, và như vậy bị lặp vô hạn giữa I và H.
- Độ phức tạp về thời gian: trong trường hợp xấu nhất là O(b^m). Độ phức tạp này có thể giảm rất nhiều nếu tồn tại heuristic tốt. Tuy nhiên trong trường hợp heuristic không tốt thì thuật toán sẽ đi sai hướng và do vậy gần giống với tìm sâu.
- Độ phức tạp về không gian lưu trữ: trong trường hợp xấu nhất thuật toán lưu O(b^m) nút với m là độ sâu cây tìm kiếm, tuy nhiên nếu heuristic tốt thì số nút cần lưu giảm đi rất nhiều như trong ví dụ trên.
- Thuật toán không tối ưu: trong ví dụ ở trên, đường đi S→E→G tìm được có độ dài là 154 trong khi đường đi ngắn nhất S→B→F→G có độ dài bằng 97.

2.4.2. Thuật toán A*

Một trong những nhược điểm của tìm kiếm tham lam là không cho lời giải tối ưu, tức là lời giải với đường đi ngắn nhất. Lý do tìm kiếm tham lam không đảm bảo tìm ra đường đi ngắn nhất là do thuật toán chỉ quan tâm tới khoảng cách ước lượng từ một nút tới đích mà không quan tâm tới quãng đường đã đi từ nút xuất phát tới nút đó. Trong trường hợp khoảng cách từ nút xuất phát tới nút đang xét lớn sẽ làm tổng độ dài đường đi từ xuất phát tới đích qua nút hiện thời lớn lên.

Để khắc phục nhược điểm này, thuật toán A* sử dụng hàm đánh giá $f(n)$ với hai thành phần, thành phần thứ nhất là đường đi từ nút đang xét tới nút xuất phát, thành phần thứ hai là khoảng cách ước lượng tới đích.

Phương pháp: khắc phục nhược điểm của thuật toán tham lam, thuật toán A* sẽ sử dụng hàm $f(n) = g(n) + h(n)$. Trong đó:

- $g(n)$ là giá thành đường đi từ nút xuất phát đến nút n
- $h(n)$ là giá thành ước lượng đường đi từ nút n đến nút đích; $h(n)$ là hàm heuristic.

Trong phần tìm kiếm mù, ta đã xem xét thuật toán tìm kiếm với giá thành thống nhất, trong đó nút biên có hàm $g(n)$ nhỏ nhất được chọn để mở rộng. Như vậy, thuật toán A* khác tìm kiếm với giá thành thống nhất ở chỗ sử dụng thêm hàm $h(n)$ để ước lượng khoảng cách tới đích.

Thuật toán A* yêu cầu hàm $h(n)$ là hàm *chấp nhận được* (admissible) theo định nghĩa sau.

Định nghĩa: *Hàm h(n) được gọi là chấp nhận được nếu h(n) không lớn hơn độ dài đường đi thực ngắn nhất từ n tới nút đích.*

Thuật toán A* được thể hiện trên hình 2.16.

Thuật toán: *A**(*Q*, *S*, *G*, *P*, *c*, *h*)

- Đầu vào: bài toán tìm kiếm, hàm heuristic *h*
- Đầu ra: đường đi ngắn nhất từ nút xuất phát đến nút đích
- Khởi tạo: tập các nút biên (nút mở) $O \leftarrow S$

While(*O* không rỗng) do

1. Lấy nút *n* có *f*(*n*) nhỏ nhất ra khỏi *O*
2. Nếu *n* thuộc *G*, return(đường đi tới *n*)
3. Với mọi $m \in P(n)$
   i. $g(m) = g(n) + c(m, n)$
   ii. $f(m) = g(m) + h(m)$
   iii. Thêm *m* vào *O* cùng giá trị *f*(*m*)

Return: không tìm được đường đi

Hình 2.16. Thuật toán A*

Với thuật toán A*, có thể sử dụng phiên bản Graph-search (tức là có danh sách nút đóng) hoặc không. Trong trường hợp không sử dụng nút đóng, tất cả các nút mới sinh ra đều được thêm vào tập biên. Trong trường hợp sử dụng phiên bản Graph-search, một nút đã mở rộng sẽ không được thêm vào danh sách biên. Nếu một nút đã có một bản sao trong tập biên thì bản sao với giá trị hàm *f*(*n*) nhỏ hơn sẽ được giữ lại.

Ví dụ. Kết quả các bước thực hiện thuật toán A* cho bài toán tìm đường từ S tới G trên đồ thị hình 2.14 được thể hiện dưới đây. Các thông tin bao gồm nút được mở rộng, danh sách các nút trong tập biên cùng với giá trị hàm *f*(*n*). Phiên bản được trình bầy ở đây là phiên bản không sử dụng danh sách nút đóng và không kiểm tra tập biên. Như vậy, tập biên có thể chứa nhiều phiên bản của cùng một nút với giá trị hàm *f*(*n*) khác nhau.

Nút được mở rộng	Tập biên O. Giá trị hàm *f*(*n*) được cho trong ngoặc
	S (125)
S	A_S (123+55), B_S (82+42), C_S (118+48), E_S (72+72)
B_S	A_S (123+55), C_S (118+48), E_S (72+72), S_B (125+84), C_B (118+82), F_B (40+82)
F_B	A_S (123+55), C_S (118+48), E_S (72+72), S_B (125+84),

	C_B (118+82), G_F (0+137), C_F (118+150)
G_F	đích

Sử dụng con trỏ ngược, ta có đường đi tìm được là G←F←B←S với độ dài 137. Đây cũng là đường đi ngắn nhất giữa S và G.

Đặc điểm của thuật toán A*:

- Thuật toán cho *kết quả tối ưu* nếu hàm heuristic *h* là hàm *chấp nhận được*.
- *Thuật toán đầy đủ*, trừ trường hợp có vô số các nút với hàm *f* có giá trị rất nhỏ nằm giữa node xuất phát và node đích.
- Độ phức tạp: trong trường hợp xấu nhất, khi hàm heuristic không có nhiều thông tin, độ phức tạp tính toán và yêu cầu bộ nhớ của A* đều là O(b^m).
- Trong tất cả các thuật toán tìm kiếm tối ưu sử dụng cùng hàm heuristics thì thuật toán A* có độ phức tạp tính toán nhỏ nhất, tức là yêu cầu sinh ra ít nút nhất trước khi tìm ra lời giải.

Yêu cầu bộ nhớ lớn là nhược điểm lớn nhất của A*. Do nhược điểm này, A* thường không thể sử dụng để giải các bài toán có kích thước lớn. Trong phần sau, ta sẽ xem xét một thuật toán với yêu cầu bộ nhớ nhỏ, trong khi vẫn cho phép tìm ra lời giải tối ưu và có độ phức tạp tính toán gần tốt như A*.

2.4.3. Các hàm heuristic

Trong tìm kiếm có thông tin, các hàm heuristic đóng vai trò rất quan trọng. Hàm heuristic tốt đảm bảo tính tối ưu cho những thuật toán như A*. Bên cạnh đó, hàm heuristic tốt cho phép hướng thuật toán theo phía lời giải, nhờ vậy giảm số trạng thái cần khảo sát và độ phức tạp của thuật toán.

Các hàm heuristic $h(n)$ được xây dựng tùy thuộc vào bài toán cụ thể. Cùng một loại bài toán chúng ta có thể có rất nhiều hàm heuristic khác nhau. Chất lượng hàm heuristic ảnh hưởng rất nhiều đến quá trình tìm kiếm.

Hàm heuristic $h(n)$ được gọi là chấp nhận được khi:

$$h(n) \le h^*(n)$$

trong đó $h^*(n)$ là giá thành đường đi thực tế từ n đến node đích. Lưu ý rằng hàm $h(n)=0$ với mọi n, là hàm chấp nhận được. Để minh hoạ cho hàm heuristic, ta xét một số ví dụ sau.

Ví dụ:

Trong bài toán tìm đường, đường chim bay như nhắc tới ở trên là một ví dụ của hàm heuristic chấp nhận được.

Trong bài toán 8 ô, có thể xây dựng một số hàm heuristic. Dưới đây, ta sẽ xem xét hai trong số các hàm như vậy.

3	1	6
5		4
2	7	8

Trạng thái hiện thời

	1	2
3	4	5
6	7	8

Trạng thái đích

Ta có thể sử dụng hai hàm heuristic sau.

- $h_1(n)$: số ô đặt sai chỗ. Chẳng hạn nếu hình bên phải là trạng thái đích và hình bên trái là trạng thái u thì trạng thái bên trái có $h_1(u) = 5$ do có 5 ô là các ô 3, 6, 4, 5, 2 nằm sai vị trí. Có thể nhận thấy h_1 là hàm chấp nhận được do muốn di chuyển từ trạng thái bên trái sang trạng thái đích ta phải chuyển vị trí tất cả những ô đứng sai, trong khi để di chuyển mỗi ô sai, ta cần ít nhất một nước đi. Như vậy độ dài đường đi luôn lớn hơn hoặc bằng h1.
- $h_2(n)$: tổng khoảng cách Manhattan giữa vị trí hiện thời của mỗi ô tới vị trí đúng của ô đó. Khoảng cách Manhattan được tính bằng số ít nhất các dịch chuyển theo hàng hoặc cột để đưa một quân tới vị trí của nó trong trạng thái đích. Ví dụ, để đưa quân 2 tới vị trí đích ta cần 4 dịch chuyển và do vậy khoảng cách Manhattan của 2 tới đích là 4. Giá trị h_2 của trạng thái u trên hình bên trái sẽ bằng $h_2(u) = 1 + 4 + 1 + 2 + 1$. Hàm h_2 cũng là hàm chấp nhận được. Thật vậy, để di chuyển một ô tới vị trí đích, ta cần ít nhất số nước đi bằng khoảng cách Manhattan từ ô đó tới đích. Như vậy, số nước để di chuyển toàn bộ các ô đứng sai sẽ lớn hơn hoặc bằng tổng khoảng cách Manhattan như cách tính h_2.

Hàm heuristic trội

Ví dụ trên cho thấy, với cùng một bài toán, ta có thể xây dựng đồng thời nhiều hàm heuristic chấp nhận được. Vấn đề đặt ra khi đó là nên dùng hàm nào trong thuật toán tìm kiếm, hàm được chọn phải là hàm tốt hơn, tức là hàm nhanh dẫn tới kết quả hơn.

Giả sử có hai hàm heuristic chấp nhận được $h_1(n)$ và $h_2(n)$. Nếu $h_1(n) \le h_2(n)$ với mọi n thì ta nói rằng $h_2(n)$ trội hơn so với $h_1(n)$. Rõ ràng hàm $h_2(n)$ mang nhiều thông tin hơn và do vậy là hàm tốt hơn do dẫn tới kết quả nhanh hơn.

Trong trường hợp trong hai hàm $h_1(n)$ và $h_2(n)$ không có hàm trội hơn thì ta có thể tạo ra hàm $h'(n) = \max(h_1(n), h_2(n))$ với mọi n. Rõ ràng, $h'(n)$ là hàm trội hơn hai hàm ban đầu.

Cách xây dựng hàm heuristic

Việc lựa chọn hàm heuristic phụ thuộc vào bài toán cụ thể. Nguyên tắc chung để xây dựng hàm heuristic cho một bài toán là nới lỏng các ràng buộc của bài toán đó khi ước lượng đường đi tới đích. Ví dụ, đối với hàm $h_1(n)$ trong trò đố 8 ô, ta đã bỏ ràng buộc về việc các ô phải di chuyển từng bước để đến được vị trí đích. Hay đối với heuristic đường chim bay, ta đã nới rộng ràng buộc về việc phải di chuyển trên đường bộ (thường không phải là đường thẳng) và giả sử có thể di chuyển thẳng từ một điểm tới đích.

2.4.4. Thuật toán IDA* (thuật toán A* sâu dần)

Thuật toán A* có những ưu điểm quan trọng như tính tối ưu và tính đầy đủ. Tuy nhiên, nếu hàm heuristic không tốt, thuật toán sẽ phải xem xét nhiều trạng thái và có yêu cầu bộ nhớ trong trường hợp xấu nhất là $O(b^m)$. Yêu cầu bộ nhớ theo hàm mũ như vậy làm giảm khả năng sử dụng A*. Để giải quyết vấn đề này có thể sử dụng phiên bản của A* được gọi là A* sâu dần (Iterative Deepening A*) có yêu cầu bộ nhớ tỷ lệ tuyến tính với độ sâu lời giải.

Phương pháp: Nguyên tắc của A* sâu dần là lặp lại việc tìm kiếm theo chiều sâu trên các cây tìm kiếm con có giá trị hàm $f(n)$ không lớn hơn các ngưỡng 0, α, 2α, 3α, .v.v..

Cụ thể:

1. Tìm kiếm sâu dần (DFS), không mở rộng nút có hàm f > 0, nếu tìm được đích thì dừng lại.

2. Tìm kiếm sâu dần (DFS), không mở rộng nút có hàm f > α, nếu tìm được đích thì dừng lại.

3. Tìm kiếm sâu dần , không mở rộng nút có hàm f > 2α, nếu tìm được đích thì dừng lại.

...

Ở đây, α là giá trị được thêm vào ngưỡng sau mỗi vòng lặp. Để mỗi vòng lặp có thể xét thêm các nút mới α cần có giá trị lớn hơn hoặc bằng giá thành nhỏ nhất để di chuyển giữa hai trạng thái trong không gian tìm kiếm. Ở đây cần lưu ý cách chọn α. Nếu α nhỏ, sau mỗi lần tăng ngưỡng, cây tìm kiếm mới sẽ thêm được ít nút do với cây tìm kiếm cũ và do vậy cần lặp lại quá trình tìm sâu nhiều lần, dẫn tới tăng độ phức tạp tính toán. Ví dụ, trong trường hợp đặc biệt, khi giá trị của $f(n)$ trên mọi nút đều khác nhau, mỗi bước lặp sẽ chỉ xem xét thêm được một nút so với bước trước. Khi đó, nếu A* cần mở rộng N nút, thì A* sâu dần sẽ phải mở rộng $1 + 2 + \ldots + N = O(N^2)$.

Giải pháp cho vấn đề độ phức tạp tính toán là sử dụng mức độ tăng ngưỡng $\beta > \alpha$, sao cho tại mỗi bước lặp sẽ mở rộng cây tìm kiếm thêm một số nút mới. Giá trị β như vậy cho phép giảm thời gian tìm kiếm, tuy nhiên chỉ trả lại lời giải β - tối ưu trong trường hợp xấu nhất. Một lời giải m^* được gọi là β - tối ưu nếu $g(m^*) < g^* + \beta$, trong đó g^* là giá thành của lời giải tối ưu. Như vậy, lời giải g^* là lời giải có giá thành được đi không vượt quá β so với giá thành đường đi tối ưu.

Thuật toán A* sâu dần chi tiết được thể hiện trên hình dưới đây:

Thuật toán: A*(Q, S, G, P, c, h, α)

- Đầu vào: bài toán tìm kiếm, hàm heuristic h, bước nhảy α cho ngưỡng
- Đầu ra: đường đi ngắn nhất từ nút xuất phát đến nút đích
- Khởi tạo: giá trị $i = 0$ là ngưỡng cho hàm f

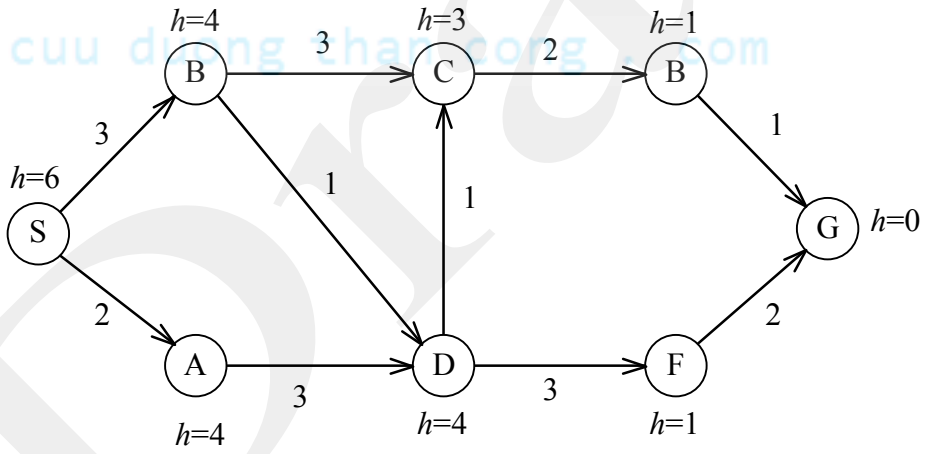
danh sách các nút biên (nút mở) O ← S

While (1) do

1. while (*O* không rỗng) do
   a) Lấy nút *n* từ đầu *O*
   b) Nếu $n \in G$, then return(đường đi tới n)
   c) Với mọi $m \in P(n)$
      i. $g(m) = g(n) + c(m, n)$
      ii. $f(m) = g(m) + h(m)$
      iii. If ($f(m) \le i$) then Thêm *m* vào đầu *O*
2. $i \leftarrow i + \alpha$, $O \leftarrow S$

Hình 2.15. Thuật toán A* sâu dần

Ví dụ. Xét bài toán tìm đường đi từ nút S tới nút G trên hình 2.16 với hàm heuristic cho cạnh các nút và giá thành đường đi bên cạnh các cung.



Hình 2.16. Ví dụ bài toán tìm đường đi

Kết quả các bước thực hiện thuật toán A* sâu dần với bước nhảy $\alpha = 2$ cho bài toán này được thể hiện trên bảng dưới đây. Kết quả bao gồm giá trị ngưỡng *i*, nút được mở rộng, tập biên O chứa các nút và giá trị hàm *f*. Thuật toán tìm sâu tại mỗi bước sử dụng phương pháp thứ hai, tức là nút được thêm vào tập biên nếu không nằm trên đường đi hiện thời. Đầu của danh sách O nằm phía trái và đuôi của O ở phía phải trong bảng.

Nút được mở rộng	Danh sách O. Giá trị hàm $f(n)$ được cho trong ngoặc
i = 0	
i = 2	
i = 4	

$i = 6$	
	S(6)
S	$A_S(6)$
A_S	
$i = 8$	
	S(6)
S	$A_S(6)$, $B_S(7)$
A_S	$B_S(7)$
B_S	$D_B(8)$
D_B	$C_D(8)$, $F_D(8)$
C_D	$F_D(8)$, $E_C(8)$
F_D	$E_C(8)$
E_C	$G_E(8)$
G_E	Đích

Theo con trỏ ngược, ta tìm được đường đi G←E←C←D←B←S. Có thể dễ dàng kiểm tra, đây chính là đường đi ngắn nhất từ S tới G.

Người đọc có thể tự kiểm tra trường hợp thực hiện thuật toán với bước nhảy $\alpha = 3$, lời giải sẽ không phải là lời giải tối ưu.

Tính chất của tìm kiếm A* sâu dần:

- Thuật toán đầy đủ và β - tối ưu (xem khái niệm β - tối ưu ở trên)
- Yêu cầu bộ nhớ tuyến tính (chỉ nhớ nhánh đang xét hiện thời tương tự tìm kiếm sâu dần)
- Độ phức tạp tính toán lớn hơn của thuật toán A* do tại mỗi vòng lặp ngoài, thuật toán lặp lại các bước của lần lặp trước.
- Không phụ thuộc vào hàm heuristic $h(n)$. Sau mỗi lần tăng giá trị ngưỡng, thuật toán lại xây dựng lại cây tìm kiếm từ đầu, tức là không sử dụng hàm heuristic $h(n)$ để thu hẹp không gian tìm kiếm và hướng về phía nhiều khả năng có trạng thái đích.

2.5. TÌM KIẾM CỤC BỘ

Các thuật toán tìm kiếm ở trên đều dựa trên việc khảo sát không gian tìm kiếm một cách hệ thống bằng cách di chuyển theo một số đường đi. Trong quá trình di chuyển, các thuật toán này lưu lại thông tin về đường đi đã qua cùng với thông tin về phương án đã hoặc chưa được xem xét tại mỗi trạng thái trên đường đi. Vấn đề của thuật toán như vậy là việc sử dụng đường

đi để khảo sát không gian tìm kiếm một cách hệ thống làm tăng số lượng trạng thái cần xem xét đồng thời đòi hỏi ghi nhớ nhiều trạng thái và do vậy không thích hợp với bài toán có không gian trạng thái lớn.

Trong phần này ta sẽ xem xét các thuật toán *tìm kiếm cục bộ* (local search), còn được gọi là tìm kiếm *cải thiện dần* (iterative improvement). Đặc điểm chính của thuật toán tìm kiếm cục bộ là tại mỗi thời điểm, thuật toán chỉ sử dụng một trạng thái hiện thời, và chỉ xem xét các láng giềng của trạng thái đó. Thuật toán không lưu đường đi hoặc các trạng thái đã khảo sát, do vậy không đòi hỏi nhiều bộ nhớ như các thuật toán đã trình bầy ở trên. Mặc dù tìm kiếm cục bộ thường không cho phép tìm được lời giải tối ưu, thuật toán loại này cho phép tìm được lời giải tương đối tốt với thời gian và bộ nhớ nhỏ hơn nhiều so với tìm kiếm hệ thống, và do vậy có thể sử dụng cho các bài toán có kích thước lớn.

Tìm kiếm cục bộ thường được sử dụng cho những bài toán **tối ưu hóa tổ hợp** hoặc tối ưu hóa rời rạc, tức là những bài toán trong đó cần tìm trạng thái tối ưu hoặc tổ hợp tối ưu trong không gian rời rạc các trạng thái, và *không quan tâm tới đường đi dẫn tới trạng thái đó*. Ví dụ, trong bài toán 8 con hậu, ta chỉ cần tìm vị trí của 8 con hậu mà không cần quan tâm tới đường di dẫn tới trạng thái đó.

Bài toán tối ưu hóa tổ hợp (tối ưu hóa rời rạc) có những đặc điểm sau.

- Tìm trạng thái tối ưu cực đại hóa hoặc cực tiểu hóa hàm mục tiêu. Không quan tâm tới đường đi.
- Không gian trạng thái rất lớn.
- Không thể sử dụng các phương pháp tìm kiếm trước để xem xét tất cả không gian trạng thái
- Thuật toán cho phép tìm lời giải tốt nhất với độ phức tạp tính toán nhỏ. Thuật toán cũng chấp nhận lời giải tương đối tốt.

Ví dụ: tối ưu hóa tổ hợp là lớp bài toán có nhiều ứng dụng trên thực tế. Có thể kể ra một số ví dụ sau: bài toán lập lịch, lập thời khóa biểu, thiết kế vi mạch, triệu con hậu,…

Tìm kiếm cục bộ được thiết kế cho bài toán tìm kiếm với không gian trạng thái rất lớn và cho phép tìm kiếm trạng thái tương đối tốt với thời gian tìm kiếm chấp nhận được.

Ý tưởng: tìm kiếm cục bộ dựa trên ý tưởng chung như sau:

- Chỉ quan tâm đến trạng thái đích, không quan tâm đến đường đi.
- Mỗi trạng thái tương ứng với một lời giải (chưa tối ưu) → cải thiện dần bằng cách chỉ quan tâm tới một trạng thái hiện thời, sau đó xem xét để để chuyển sang trạng thái hàm xóm của trạng thái hiện thời (thường là trạng thái có hàm mục tiêu tốt hơn).
- Thay đổi trạng thái bằng cách thực hiện các chuyển động (trạng thái nhận được từ trạng thái n bằng cách thực hiện các chuyển động được gọi là hàng xóm của n).

Do tìm kiếm cục bộ chỉ quan tâm tới trạng thái hiện thời và hàng xóm nên cần ít bộ nhớ hơn nhiều so với các phương pháp tìm kiếm hệ thống ở trên. Tìm kiếm cục bộ thường cho

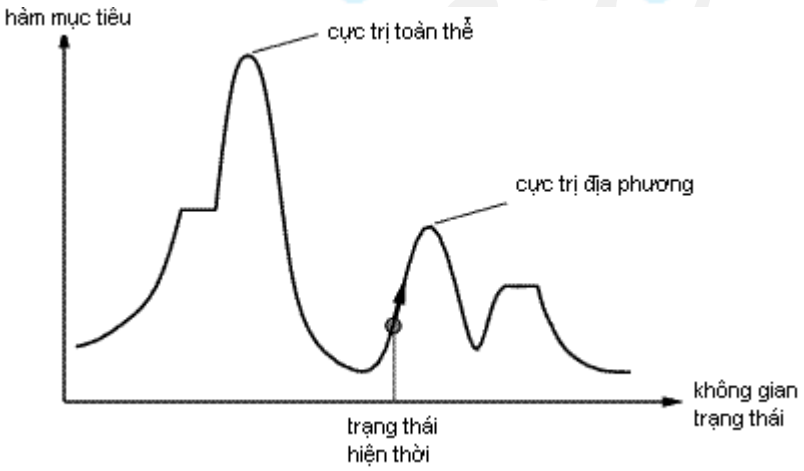
phép tìm được lời giải chấp nhận được kể cả khi bài toán lớn đến mức không dùng được những phương pháp tìm kiếm hệ thống.

Phát biểu bài toán:

Bài toán tìm kiếm cục bộ được cho bởi những thành phần sau:

- Không gian trạng thái X
- Tập chuyển động để sinh ra hàng xóm
- Hàm mục tiêu Obj: X → R
- Yêu cầu: Tìm trạng thái X* sao cho Obj(X*) là lớn nhất hoặc nhỏ nhất. Lưu ý: có thể dễ dàng biến đổi bài toán tìm trạng thái với hàm mục tiêu lớn nhất thành nhỏ nhất hoặc ngược lại bằng cách nhân hàm mục tiêu với –1.

Để minh họa cho bài toán tìm kiếm cục bộ, ta xét ví dụ trên hình 2.17 (được tham khảo từ tài liệu số 1). Trục hoành trên hình vẽ thể hiện không gian các trạng thái (để cho đơn giản, không gian trạng thái ở đây được thể hiện trong không gian một chiều dưới dạng các điểm trên trục hoành), trục tung là độ lớn của hàm mục tiêu. Yêu cầu bài toán tối ưu hóa tổ hợp là tìm được trạng thái (điểm trên trục hoành) có hàm mục tiêu lớn nhất. Lưu ý, hình vẽ minh họa trường hợp cần tìm trạng thái với hàm mục tiêu lớn nhất, tuy nhiên trong bài toán khác có thể yêu cầu tìm trạng thái với hàm mục tiêu nhỏ nhất.



Hình 2.17. Bài toán tìm kiếm cục bộ với không gian trạng thái và hàm mục tiêu

2.5.1. Thuật toán leo đồi

Leo đồi (Hill climbing) là tên chung để chỉ một họ các thuật toán có nguyên lý giống nhau. Thuật toán thực hiện bằng cách tạo ra hàng xóm cho trạng thái hiện thời và di chuyển sang hàng xóm có hàm mục tiêu tốt hơn, tức là di chuyển lên cao đối với trường hợp cần cực đại hóa hàm mục tiêu. Thuật toán dừng lại khi đạt tới một đỉnh của đồ thị hàm mục tiêu, tương ứng với trạng thái không có hàng xóm nào tốt hơn. Đỉnh này có thể là đỉnh cao nhất, hoặc cũng là những đỉnh thấp hơn (hình 2.17). Trong trường hợp thứ nhất, thuật toán tìm được giá trị cực trị, trong trường hợp thứ hai thuật toán chỉ tìm được cực trị địa phương. Thuật

toán leo đồi không lưu lại những trạng thái đã qua, đồng thời không nhìn xa hơn hàng xóm của trạng thái hiện thời.

Thuật toán leo đồi còn được gọi là thuật toán tìm cục bộ tham lam do tại mỗi thời điểm thuật toán này chỉ xét một láng giềng tốt mà không quan tâm tới tương lai xa hơn.

a) Di chuyển sang trạng thái tốt nhất

Có nhiều phiên bản khác nhau của thuật toán leo đồi. Một trong những phiên bản thông dụng nhất có tên là leo đồi *di chuyển sang trạng thái tốt nhất* (best-improvement hill climbing). Phiên bản này của leo đồi lựa chọn trong số hàng xóm hiện thời hàng xóm có hàm mục tiêu tốt nhất. Nếu hàng xóm đó tốt hơn trạng thái hiện thời thì di chuyển sang hàm xóm đó. Nếu ngược lại thì kết thúc và trả về trạng thái hiện thời. Thuật toán đầy đủ được thể hiện trên hình 2.18. Bước 1 là bước khởi tạo, trong đó ta chọn ngẫu nhiên trạng thái xuất phát. Bước 2 sinh ra các láng giềng của trạng thái hiện thời. Ở bước 3, thuật toán kiểm tra các láng giềng, nếu không có láng giềng nào tốt hơn thuật trạng thái hiện thời (tất cả láng giềng có giá trị hàm mục tiêu Obj không lớn hơn Obj của trạng thái hiện thời) thì kết thúc và trả về trạng thái hiện thời là kết quả. Trong trường hợp ngược lại, ở bước 4, thuật toán chọn láng giềng có giá trị hàm mục tiêu Obj lớn nhất và chuyển sang trạng thái đó, sau đó lặp lại từ bước 2.

Đầu vào: bài toán tối ưu tổ hợp

Đầu ra: trạng thái với hàm mục tiêu lớn nhất (hoặc cực đại địa phương)

---

1. Chọn ngẫu nhiên trạng thái x
2. Gọi Y là tập các trạng thái hàng xóm của x
3. If $\forall y_i \in Y$: Obj (y_i) < Obj (x) then
   Kết thúc và trả lại x là kết quả
4. $x \leftarrow y_i$, trong đó $i = \text{argmax}_i$ (Obj (y_i))
5. Go to 2

Hình 2.18. Thuật toán leo đồi di chuyển sang trạng thái tốt nhất

Đặc điểm của leo đồi

- Đơn giản, dễ lập trình. Trong các thuật toán trình bầy ở phần trước, khi lập trình cần sử dụng các cấu trúc dữ liệu để biểu diễn các nút, nhớ tập biên. Thuật toán leo đồi không đòi hỏi phải lập trình với các cấu trúc như vậy.
- Như các giải thuật tìm kiếm cục bộ khác, leo đồi sử dụng ít bộ nhớ do không phải lưu lại các trạng thái. Tại mỗi thời điểm, thuật toán chỉ cần lưu lại trạng thái hiện thời và một tráng thái láng giềng.
- Bên cạnh hai ưu điểm trên, leo đồi dễ bị lời giải tối ưu cục bộ (cực trị địa phương) tương ứng với đỉnh các "đồi" thấp trong hình 2.17. Nếu bắt đầu từ một

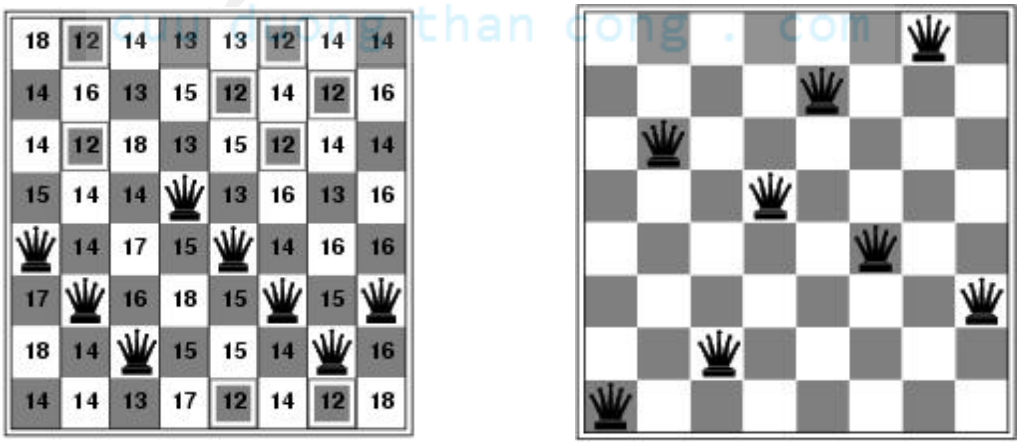
trạng thái nằm trong phạm vi của các vùng "đồi" thấp, thuật toán sẽ di chuyển về trạng thái tương ứng với đỉnh đồi, sau đó dừng lại ở đỉnh do các trạng thái xung quanh đều không tốt bằng. Mặc dù đỉnh đồi thấp tương ứng với trạng thái tốt nhất trong một vùng, trạng thái đó không phải là trạng thái tối ưu trong toàn bộ không gian trạng thái. Để khắc phục phần nào vấn đề này, thuật toán được thực hiện nhiều lần, mỗi lần sử dụng một trạng thái xuất phát sinh ngẫu nhiên khác với trạng thái xuất phát trong những lần trước đó. Nếu số lần thực hiện với trạng thái xuất phát ngẫu nhiên như vậy được thực hiện đủ nhiều thì xác suất tìm ra lời giải tối ưu sẽ tiến tới 1.

Khi thiết kế thuật toán leo đồi với cách chuyển động sang trạng thái láng giềng tốt nhất, việc lựa chọn chuyển động rất quan trọng. Nếu nhiều chuyển động, từ trạng thái hiện thời có thể sinh ra nhiều láng giềng. Do thuật toán cần so sánh tất các láng giềng để tìm ra trạng thái tốt nhất, mỗi bước của vòng lặp sẽ đòi hỏi nhiều thời gian do phải tính hàm mục tiêu cho tất cả láng giềng. Ngược lại, nếu sinh ra tập láng giềng nhỏ sẽ dễ dẫn tới cực trị địa phương do không vượt qua được những "hố" nhỏ trên đường đi.

Ví dụ minh hoạ 1: bài toán 8 con hậu. Ta sẽ quy định mỗi trạng thái là sắp xếp của cả 8 con hậu trên bàn cờ, sao cho mỗi cột chỉ gồm 1 con. Trạng thái xuất phát được khởi tạo ngẫu nhiên. Hàm mục tiêu là hàm heuristic được tính bằng số các đôi hậu đang đe dọa nhau. Như vậy hàm mục tiêu càng nhỏ thì càng gần với trạng thái đích cần tìm. Trạng thái đích là trạng thái có hàm mục tiêu nhỏ nhất (bằng 0). Để minh họa cho ảnh hưởng của việc lựa chọn hàm chuyển động, ta xét hai cách chuyển động sau.

Cách 1: chọn một cột, dịch chuyển quân hậu trong cột đó sang dòng khác. Với cách chuyển động như vậy, từ mỗi trạng thái có thể sinh ra 7 x 8 = 56 láng giềng. Trên hình 2.19 a là ví dụ một trạng thái với hàm mục tiêu bằng 17, đồng thời trên các ô trống là giá trị hàm mục tiêu (số đôi hậu đe dọa nhau) ứng với chuyển động di chuyển quân hậu trong cùng cột tới ô đó. Với số lượng láng giềng tương đối ít, cách này dễ dẫn tới cực trị địa phương, chẳng hạn trạng thái trên hình 2.18 b có hàm mục tiêu = 2 nhưng tất cả láng giềng đều có hàm mục tiêu lớn hơn và do vậy không tìm được lời giải.

Cách 2: chọn hai cột, dịch chuyển đồng thời mỗi quân hậu trong hai cột đó sang dòng khác với dòng hiện thời. Cách này tạo ra 7 x 8 + 7 x 7 láng giềng cho mỗi trạng thái. Mặc dù số lượng láng giềng lớn đòi hỏi mỗi bước lặp phải tính toán nhiều gần gấp đôi so với cách 1 nhưng sẽ ít khả năng dẫn tới cực trị địa phương như trạng thái trên hình 2.19 b hơn.



(a) (b)

Hình 2.19. Bài toán 8 con hậu với giá trị hàm mục tiêu cho mỗi trạng thái láng giềng (a) và trạng thái cực trị địa phương (b)

b) Leo đồi ngẫu nhiên

Leo đồi ngẫu nhiên (stochastic hill climbing) là một phiên bản khác của leo đồi. Thay vì tìm ra hàng xóm tốt nhất, phiên bản này lựa chọn ngẫu nhiên một hàng xóm. Nếu hàng xóm đó tốt hơn trạng thái hiện thời, hàng xóm đó sẽ được chọn làm trạng thái hiện thời và thuật toán lặp lại. Ngược lại, nếu hàng xóm được chọn không tốt hơn, thuật toán sẽ chọn ngẫu nhiên một hàng xóm khác và so sánh. Thuật toán kết thúc và trả lại trạng thái hiện thời khi đã hết "kiên nhẫn". Thông thường, độ kiên nhẫn được cho bằng số lượng tối đa hàng xóm mà thuật toán xem xét trong mỗi bước lặp hoặc trong toàn bộ thuật toán.

Thuật toán leo đồi ngẫu nhiên được thể hiện trên hình 2.20.

Đầu vào: bài toán tối ưu tổ hợp

Đầu ra: trạng thái với hàm mục tiêu lớn nhất (hoặc cực đại địa phương)

---

1. Chọn ngẫu nhiên trạng thái x
2. Gọi Y là tập các trạng thái hàng xóm của x
3. Chọn ngẫu nhiên $y_i \in Y$
4. Nếu Obj (yi) > Obj (x) thì

   $x \leftarrow yi$
5. Go to 2 nếu chưa hết kiên nhẫn

Hình 2.20. Thuật toán leo đồi ngẫu nhiên

Các thực nghiệm trên nhiều bài toán cho thấy, trong trường hợp mỗi trạng thái có nhiều láng giềng, leo đồi ngẫu nhiên cho kết quả nhanh hơn do thuật toán có thể chuyển sang trạng thái tiếp theo mà không cần khảo sát toàn bộ tập láng giềng. Ngoài ra, leo đồi ngẫu nhiên cũng ít gặp phải cực trị địa phương hơn leo đồi chuyển sang trạng thái tốt nhất.

Nhìn chung, khả năng tìm được lời giải tối ưu của các thuật toán leo đồi phụ thuộc nhiều vào không gian trạng thái. Đối với những không gian có ít cực trị địa phương, leo đồi thường tìm được lời giải khá nhanh. Trong trường hợp không gian trạng thái phức tạp, thuật toán thường chỉ tìm được cực trị địa phương. Tuy nhiên, bằng cách thực hiện nhiều lần với trạng thái xuất phát ngẫu nhiên, leo đồi thường tìm được cực trị địa phương khá tốt.

Ví dụ minh hoạ 2: Bài toán người bán hàng. Cho một đồ thị gồm n nút, trong đó hai nút bất kỳ đều được nối với nhau bằng một cung. Cần tìm đường đi xuất phát từ một nút, qua tất cả các nút khác đúng một lần và quay về nút xuất phát sao cho độ dài đường đi là nhỏ nhất.

Bài toán này mô phỏng trường hợp một người bán hàng cần đi qua tất cả các thành phố đúng một lần và quay về nơi xuất phát với chi phí nhỏ nhất.

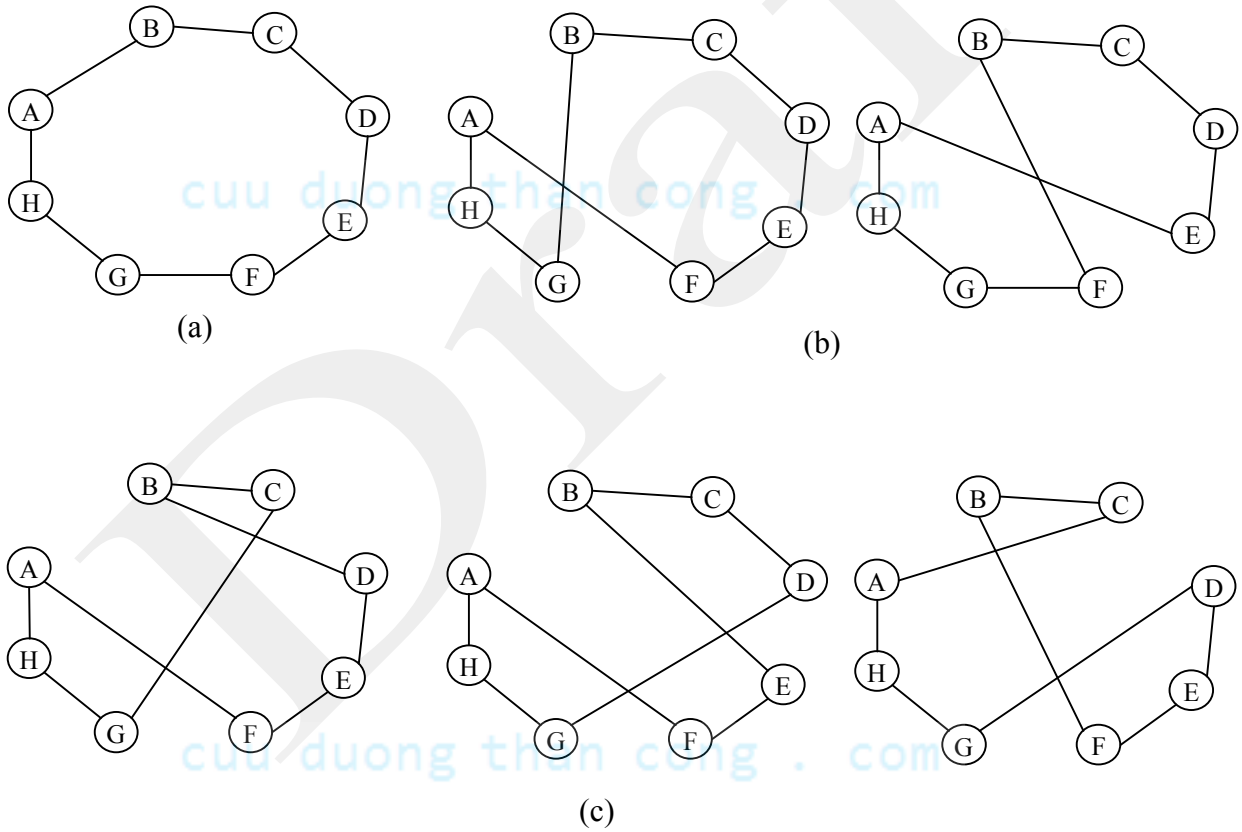
Có thể áp dụng thuật toán leo đồi cho bài toán người bán hàng như sau.

Trạng thái: Mỗi trạng thái là một chu trình Hamilton, tức là chu trình qua các đỉnh đúng một lần.

Hàm mục tiêu: Là độ dài của chu trình Hamilton. Cần tìm trạng thái có hàm mục tiêu nhỏ nhất.

Chuyển động: Chuyển động theo kiểu *thay_đổi_k_cung*, trong đó k = 2, 3, 4, … Ví dụ, với k = 2, từ trạng thái hiện thời ta lấy ra 2 cung không liền nhau, bỏ hai cung đó và nối nút đầu cung này với nút cuối cung kia để tạo ra 2 cung mới như ví dụ trên hình 2.21 (trên). Với mỗi cặp cung như vậy ta được một láng giềng mới của trạng thái hiện thời. Trong ví dụ đang xét, bằng cách thay đổi 2 cung, ta có thể tạo ra 20 láng giềng cho mỗi trạng thái.

Tương tự, hình 2.21 (dưới) minh họa một số trạng thái láng giềng được tạo ra khi thực hiện thay_đổi_3_cung. Với đồ thị có nhiều nút, khi k tăng lên, số chuyển động và láng giềng tương ứng cũng tăng theo.



Hình 2.21. Ví dụ một trạng thái của bài toán người bán hàng (a), một số láng giềng sinh ra nhờ thay_đổi_2_cung (b), và một số láng giềng sinh ra nhờ thay_đổi_3_cung.

Với các đồ thị có vài chục nút trở lên, các nghiên cứu thực nghiệm cho thấy một số kết quả sau. Giá trị k = 3 cho kết quả tốt hơn nhiều (thường tìm được đường đi ngắn hơn) so với k = 2. Việc sử dụng k = 4 và lớn hơn không cải thiện chất lượng lời giải đáng kể so với k = 3, trong khi đòi hỏi khối lượng tính toán lớn hơn nhiều do tập láng giềng lớn.

2.5.2. Thuật toán tôi thép

Một vấn đề lớn với leo đồi là thuật toán không có khả năng "đi xuống" và do vậy không thoát khỏi được cực trị địa phương khi đã rơi vào. Ngược lại, cách di chuyển hoàn toàn ngẫu nhiên (random walk) có thể khảo sát toàn bộ không gian trạng thái nhưng không hiệu quả. *Thuật toán tôi thép* hay *mô phỏng luyện kim* (simulated annealing) là một phương pháp tìm kiếm cục bộ cho phép giải quyết phần nào vấn đề cực trị địa phương một cách tương đối hiệu quả.

Có thể coi tôi thép là phiên bản của thuật toán leo đồi ngẫu nhiên, trong đó thuật toán chấp nhận cả những trạng thái kém hơn trạng thái hiện thời với một xác suất p nào đó. Cụ thể là khi lựa chọn ngẫu nhiên một hàng xóm, nếu hàng xóm đó kém hơn trạng thái hiện thời, thuật toán có thể quyết định di chuyển sang đó với một xác suất p.

Vấn đề quan trọng đối với thuật toán là lựa chọn xác suất p thế nào. Nguyên tắc chung là không chọn p cố định, giá trị p được xác định dựa trên hai yếu tố sau.

- Nếu trạng thái mới kém hơn nhiều so với trạng thái hiện thời thì p phải giảm đi. Có nghĩa là xác suất chấp nhận trạng thái tỷ lệ nghịch với độ kém của trạng thái đó. Gọi $\Delta(x,y) = Obj(x) - Obj(y)$ trong đó x là trạng thái hiện thời, ta cần chọn p tỷ lệ nghịch với $\Delta(x,y)$.
- Theo thời gian, giá trị của p phải giảm dần. Ý nghĩa của việc giảm p theo thời gian là do khi mới bắt đầu, thuật toán chưa ở vào vùng trạng thái tốt và do vậy chấp nhận thay đổi lớn. Theo thời gian, thuật toán sẽ chuyển sang vùng trạng thái tốt hơn và do vậy cần hạn chế thay đổi.

Thuật toán tôi thép được thể hiện trên hình 2.22. Khác với thuật toán leo đồi, trong đó ta luôn chuyển động sang trạng thái tốt hơn và do vậy trạng thái hiện thời luôn là trạng thái tốt nhất, thuật toán tôi thép có thể di chuyển sang trạng thái kém hơn. Do vậy, thuật toán tôi thép sử dụng một biến x^* để lưu trạng thái tốt nhất đã từng khảo sát, tính tới thời điểm hiện tại. Lệnh **if** rand[0,1] < p **then** x ← y có nghĩa là gán y cho x với xác suất p, trong đó rand[0,1] là hàm trả về giá trị ngẫu nhiên trong khoảng [0,1]. Lệnh này có thể triển khai trên máy tính có hàm sinh số tự nhiên.

SA(X, Obj, N, m, x, C) //Obj càng nhỏ càng tốt

Đầu vào: số bước lặp m

trạng thái bắt đầu x (chọn ngẫu nhiên)

sơ đồ làm lạnh C

Đầu ra: trạng thái tốt nhất x^*

Khởi tạo: $x^* = x$

For i = 1 to m

1. chọn ngẫu nhiên $y \in N(x)$

   a) tính $\Delta(x,y) = Obj(y) - Obj(x)$

   b) **if** $\Delta(x,y) < 0$ **then** p = 1

   c) **else** p = e - $\Delta(x,y)/T$

   d) **if** rand[0,1] < p **then** x ← y //gán y cho x với xác suất p.

      if Obj(x) < Obj(x*) then x* ← x

2. giảm T theo sơ đồ C

return x* //x* là trạng thái tốt nhất trong số những trạng thái đã xem xét

Hình 2.22. Thuật toán tôi thép

Thuật toán tôi thép vừa trình bày dựa trên một hiện tượng cơ học là quá trình làm lạnh kim loại để tạo ra cấu trúc tinh thể bền vững. Hàm mục tiêu khi đó được đo bằng độ vững chắc của cấu trúc tinh thể. Khi còn nóng, mức năng lượng trong kim loại cao, các nguyên tử kim loại có khả năng di chuyển linh động hơn. Khi nhiệt độ giảm xuống, tinh thể dần chuyển tới trạng thái ổn định và tạo ra mạng tinh thể. Bằng cách thay đổi nhiệt độ hợp lý, có thể tạo ra những mạng tinh thể rất rắn chắc.

Chính vì sự tương tự với cách tôi kim loại như vậy nên trong thuật toán, xác suất p giảm theo thời gian dựa vào một công thức gọi là sơ đồ làm lạnh C. Có nhiều dạng sơ đồ làm lạnh khác nhau. Sau đây là ví dụ một sơ đồ làm lạnh.

Sơ đồ làm lạnh C:

$$T_{t+1} = T_0 * \alpha^{t*k}$$

Trong đó:

$T_0 > 0$ là giá trị ban đầu, α thuộc (0,1), $1 < k < m$

Với sơ đồ làm lạnh này, khi số vòng lặp tăng lên, tức là t tăng, giá trị của T giảm. Tiếp theo, từ công thức tính xác suất sử dụng trong thuật toán, có thể nhận thấy khi T tăng và tiến tới vô cùng, xác suất $p \to 1$ mọi $\Delta(x, y)$. Khi đó, thuật toán chấp nhận bất chuyển động nào bất kể trạng thái mới tốt hơn hay kém hơn, tức là thuật toán di chuyển ngẫu nhiên trong không gian trạng thái.

Ngược lại, khi $T \to 0$, ta có $p \to 0$ với mọi $\Delta(x, y)$, tức là thuật toán không chấp nhận các trạng thái kém hơn trạng thái hiện thời và do vậy trở thành leo đồi ngẫu nhiên. Như vậy, có thể coi thuật toán tôi thép là sự kết hợp giữa leo đồi ngẫu nhiên với chuyển động ngẫu nhiên, trong đó khởi đầu xu hướng chuyển động ngẫu nhiên lớn hơn, và càng về cuối, xu hướng leo đồi càng chiếm ưu thế.

Việc lựa chọn các tham số cho sơ đồ làm lạnh thường được thực hiện bằng cách thực nghiệm với từng bài toán cụ thể.

Thuật toán tôi thép được dùng nhiều trong việc thiết kế vi mạch có độ tích hợp lớn cũng như giải quyết những bài toán tối ưu hóa tổ hợp có kích thước lớn trên thực tế. Nhiều ứng dụng cho thấy, thuật toán tôi thép có khả năng tìm được lời giải tốt hơn so với thuật toán leo đồi.

2.5.3. Giải thuật di truyền

Giải thuật di truyền (genetic algorithm) hay *thuật toán di truyền* là thuật toán tìm kiếm được thiết kế dựa trên sự tương tự với quá trình chọn lọc tự nhiên và thuyết tiến hoá của Charles Darwin. Giải thuật di truyền là trường hợp riêng của một lớp giải thuật được gọi là *giải thuật tiến hoá* (evolutionary algorithms). Giải thuật di truyền cho lời giải tốt trong nhiều bài toán tối ưu và được sử dụng rộng rãi trong rất nhiều ứng dụng khác nhau.

Về mặt thuật toán, có thể xem giải thuật di truyền như một phiên bản leo đồi ngẫu nhiên với một số khác biệt chính. Thứ nhất, thuật toán đồng thời duy trì nhiều lời giải tại mỗi thời điểm thay vì một lời giải như leo đồi. Thứ hai, từng cặp lời giải có thể trao đổi thành phần với nhau để tạo ra lời giải mới.

Về mặt ý tưởng, giải thuật di truyền học tập từ quá trình sinh tồn và chọn lọc tự nhiên của sinh vật trong tự nhiên, trong đó cá thể với khả năng thích nghi cao hơn sẽ chiếm ưu thế và tồn tại. Nếu ta coi đây là bài toán tối ưu thì quá trình tiến hoá sẽ sinh ra cá thể tối ưu, tức là cá thể thích nghi cao.

1) Nguyên lý chung.

Giải thuật di truyền mô phỏng quá trình tiến hoá qua các thế hệ trong tự nhiên như sau. Mỗi thế hệ gồm một số nhất định *lời giải*, còn gọi là *cá thể*. Mỗi lời giải hay cá thể được biểu diễn bằng một *nhiễm sắc thể*, chứa các *gen*. Độ tốt của lời giải được đánh giá bằng *hàm thích nghi* (fitness). Thông thường, hàm thích nghi chính là hàm mục tiêu của bài toán tối ưu. Các lời giải trong mỗi thế hệ được biến đổi trong quá trình thích nghi để tạo ra thế hệ tiếp theo.

Nguyên bản sinh học. Trong sinh học, mỗi cá thể được mã hoá di truyền bằng bộ gen, bộ gen có thể gồm nhiều nhiễm sắc thể như của người hoặc động vật bậc cao, mỗi nhiễm sắc thể gồm nhiều gen. Khi sinh sản, hai cá thể bố mẹ trao đổi các gen với nhau, cá thể con nhận một số gen từ bố và một số gen từ mẹ.

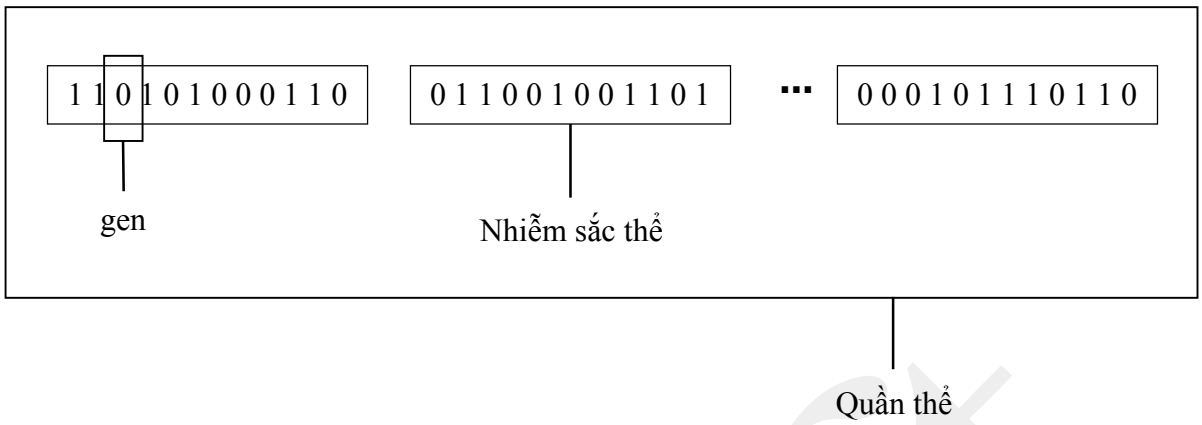
Giải thuật di truyền sử dụng các quy tắc sau để tạo ra thế hệ lời giải tiếp theo.

- Các cá thể cạnh tranh với nhau. Cá thể với độ thích nghi cao hơn sẽ tạo ra nhiều hậu duệ hơn cá thể kém thích nghi.
- Gen từ các cá thể thích nghi tốt được kết hợp với nhau, nhờ đó có thể tạo ra hậu duệ tốt hơn tổ tiên.
- Từng cá thể cũng có thể thay đổi gen của mình.
- Nhờ vậy, thế hệ tiếp theo sẽ chứa cá thể thích nghi hơn với môi trường.

2) Biểu diễn lời giải và không gian tìm kiếm

Để sử dụng giải thuật di truyền, trước hết mỗi lời giải phải được biểu diễn bằng một nhiễm sắc thể, chứa các gen. Ở đây, nhiễm sắc thể là một vec tơ có độ dài xác định, mỗi phần tử của vec tờ là một gen. Thông thường, lời giải được biểu diễn bằng vec tơ nhị phân, tức là

phần tử nhận giá trị {0, 1}, nhưng cũng có thể sử dụng phần tử là số hoặc chữ. Ví dụ của gen, nhiễm sắc thể, và quần thể được minh hoạ trên hình 2.23.



Hình 2.23. Ví dụ biểu diễn lời giải và quần thể trong giải thuật di truyền

Mỗi lời giải được gán một *độ thích nghi* thể hiện khả năng cạnh tranh của lời giải. Thông thường, độ thích nghi được tính từ hàm mục tiêu của bài toán, hàm mục tiêu tốt tương đương với độ thích nghi cao.

Giải thuật thực hiện qua nhiều bước lặp, mỗi bước lặp tương ứng với một thế hệ. Tại mỗi thế hệ, giải thuật duy trì một *quần thể*, tức là một tập hợp gồm N lời giải, trong đó N là tham số. Quần thể trong thế hệ đầu tiên được *khởi tạo ngẫu nhiên* bằng cách sinh ngẫu nhiên các lời giải. Từ quần thể hiện tại, từng cặp cá thể được lựa chọn dựa trên độ thích nghi và được lai ghép với nhau để tạo ra cá thể con. Cá thể với độ thích nghi cao được lựa chọn với xác suất cao hơn. Cá thể con được tạo ra như vậy sẽ thay thế cá thể bố mẹ để tạo ra quần thể mới trong thế hệ tiếp theo.

Như vậy, thế hệ tiếp theo sẽ chứa các lời giải với các gen tốt hơn mức trung bình trong thế hệ trước. Nói cách khác, thế hệ sau chứa các lời giải thành phần tốt hơn thế hệ trước. Thuật toán lặp lại cho đến khi cá thể trong thế hệ tiếp theo không khác nhiều so với thế hệ trước. Khi đó thuật toán được gọi là *hội tụ*. Cũng có thể sử dụng các tiêu chí kết thúc khác, chẳng hạn sau khi lặp một số lần nhất định.

2) Giải thuật

Quần thể đầu tiên được khởi tạo ngẫu nhiên. Sau đó thuật toán được thực hiện qua nhiều bước lặp, tại mỗi bước lặp thuật toán sinh ra một quần thể mới. Các quần thể tiếp theo được tạo ra từ quần thể trước đó bằng cách áp dụng ba thao tác: *chọn lọc*, *lai ghép*, và *đột biến*.

Để minh hoạ cho các bước của thuật toán, ta xét ví dụ bài toán 8 quân hậu, trong đó mỗi quân hậu được đặt trong một cột riêng. Mỗi trạng thái gồm thông tin về vị trí 8 quân hậu trong cột của mình. Nếu chọn biểu diễn dưới dạng chuỗi bit, cần 3 bit cho một vị trí, tức là 24 bit cho mỗi trạng thái. Nếu biểu diễn dưới dạng chuỗi số thập phân, cần 8 chữ số thập phân cho mỗi trạng thái. Trong ví dụ minh hoạ này, ta sẽ dùng biểu diễn dưới dạng thập phân như minh hoạ trên hình 2.24 và 2.25. Chẳng hạn, nhiễm sắc thể 24415124 cho biết quân hậu thứ nhất nằm ở hàng thứ 2 trong cột đầu tiên bên trái, quân thứ hai nằm ở hàng 4 trong cột tiếp theo

v.v. Do trạng thái tốt ứng với giá trị hàm thích nghi lớn nên ta sẽ chọn độ thích nghi bằng *số lượng đôi hậu không đe doạ nhau*. Trạng thái lời giải là trạng thái có độ thích nghi bằng 28.

Chọn lọc (selection). Chọn lọc cá thể để tham gia việc tạo ra thế hệ tiếp theo. Cá thể có độ thích nghi càng cao càng có nhiều khả năng được chọn. Có nhiều phương pháp thực hiện lựa chọn như vậy, chi tiết sẽ được trình bầy ở dưới. Trong ví dụ trên hình 2.24, cá thể thứ nhất được chọn 1 lần, các thể thứ hai được chọn 2 lần, cá thể thứ ba được chọn một lần, cá thể thứ tư không được chọn.

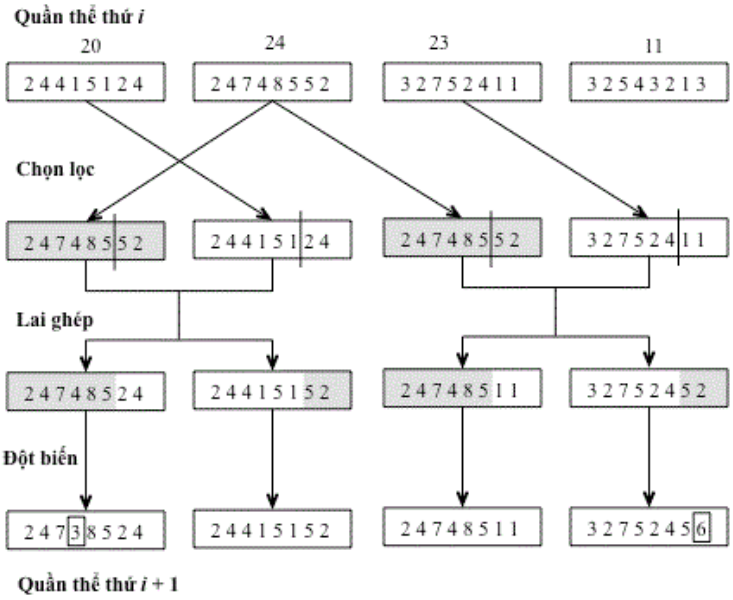
Lai ghép (crossover), còn gọi là trao đổi chéo, hay tái tổ hợp. Lai ghép là thao tác kết hợp các phần từ nhiễm sắc thể của bố và mẹ để tạo ra hai cá thể con với một xác suất nhất định. Cá thể bố và mẹ được lựa chọn bằng cách sử dụng thao tác chọn lọc ở trên. Sau đó, nhiễm sắc thể bố và mẹ được cắt ra. Phần đầu nhiễm sắc thể bố được nối với phần đuôi nhiễm sắc thể mẹ và ngược lại. Trên hình 2.24, vị trí cắt ở giữa gen thứ 6 và gen thứ 7.

Cũng có thể cắt mỗi nhiễm sắc thể bố mẹ thành nhiều hơn hai phần và ghép các phần con với nhau. Việc chọn điểm cắt và số phần con phụ thuộc bài toán cụ thể. Như vậy, nhờ kết quả lai ghép, mỗi cá thể con nhận một phần gen từ bố và một phần gen từ mẹ. Lưu ý là, với mỗi cặp bố mẹ, thao tác lai ghép được thực hiện với một xác suất nhất định gọi là *xác suất lai ghép*. Xác suất này thường được chọn lớn hơn 0.5.

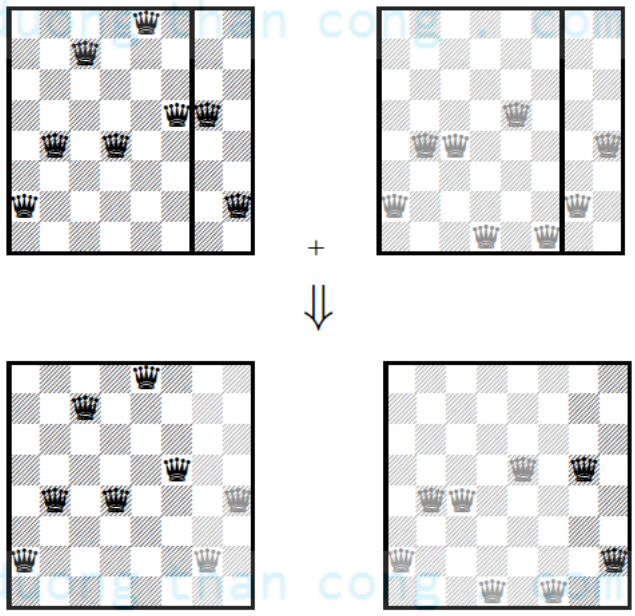
Đột biến (mutation). Thay đổi giá trị các gen của cá thể con vừa tạo ra với một xác suất nhất định. Cụ thể, với mỗi cá thể con, duyệt các gen và với một xác suất rất nhỏ thay đổi giá trị của gen đó (0 thành 1 và ngược lại nếu biểu diễn nhị phân). Xác suất này thường được chọn nhỏ hơn 0.1 để tránh đột biến quá nhiều. Trong ví dụ hình 2.24, các gen thứ 4 của cá thể con số 1 và gen thứ 8 của cá thể con số 4 được thay đổi giá trị.

Mục đích của thao tác đột biến là để tạo ra những đoạn gen hoàn toàn mới, chưa có trong quần thể cha mẹ. Đột biến cũng cho phép hạn chế việc hội tụ quá sớm của thuật toán. Về bản chất, đột biến có hiệu quả tương tự di chuyển ngẫu nhiên trong không gian trạng thái, có thể cho phép vượt qua các trường hợp cực trị địa phương.

Tại mỗi bước lặp, thuật toán thực hiện ba thao trên này để sinh ra quần thể mới. Quá trình lặp được thực hiện cho tới khi thuật toán hội tụ, tức là cá thể con không khác nhiều cá thể bố mẹ, hoặc khi thực hiện đủ một số lượng vòng lặp do người dùng quy định. Toàn bộ thuật toán được trình bầy trên hình 2.26.



Hình 2.24. Ví dụ minh hoạ các thao tác của giải thuật di truyền với bài toán 8 quân hậu. Mỗi lời giải được biểu diễn bằng chuỗi 8 chữ số tương ứng với vị trí của mỗi quân trong cột tương ứng. Hàm thích nghi được tính bằng số đôi hậu không đe doạ nhau và thể hiện bằng số ở phía trên lời giải (chỉ thể hiện cho quần thể ban đầu)



Hình 2.25. Bàn cờ minh hoạ kết quả lai ghép giữa hai nhiễm sắc thể bố mẹ thứ 1 và 2, dòng 2, trên hình 2.24

GA(X, f, N, c, m) Đầu vào: bài toán tối ưu với không gian trạng thái X hàm thích nghi f kích thước quần thể N

xác suất lai ghép c

xác suất đột biến m

Đầu ra: lời giải (cá thể) với độ thích nghi cao

Khởi tạo: Khởi tạo ngẫu nhiên quần thể G gồm N lời giải

While (chưa thoả mãn điều kiện dừng) do

1. For i = 1 to N do

   Tính giá trị hàm thích nghi f(i) cho cá thể thứ i

2. For i = 1 to ⌊N/2⌋ do

   i. Chọn lọc: chọn 2 cá thể bố mẹ x và y từ G tuỳ theo giá trị thích nghi

   ii. Lai ghép: với xác suất c, đổi chỗ đoạn gen trên x và y

   iii. Đột biến: với xác suất m, thay đổi giá trị các gen trên cá thể mới tạo ra

   iv. Thêm cá thể mới vào quần thể mới G'

3. Gán G ← G'

Return: Lời giải thuộc G với giá trị thích nghi tốt nhất

Hình 2.26. Giải thuật di truyền

3) Một số chi tiết

Các phương pháp thực hiện chọn lọc. Việc chọn lọc cá thể có thể thực hiện theo nhiều cách, ví dụ các cách sau:

- *Bánh xe ru let*: là phương pháp thường dùng nhất. Bánh xe ru let là phương pháp chọn ngẫu nhiên hay dùng trong một trò chơi ở sòng bạc. Theo phương pháp này, xác suất lựa chọn một lời giải tỷ lệ thuận với độ thích nghi của lời giải đó. Ta có thể hình dung tổng giá trị thích nghi của quần thể được biểu diễn bằng một hình tròn, hay một bánh xe ru let. Các hình quạt của bánh xe được gán cho các lời giải sao cho kích thước hình quạt tỷ lệ thuận với độ thích nghi của lời giải. Ta có thể cho bánh xe quay và tung hòn bi để chọn lời giải tương ứng với hình quạt mà hòn bi dừng lại. Giả sử $f(i)$ là độ thích nghi của lời giải thứ i. Theo phương pháp bánh xe ru let, xác suất chọn lời giải thứ i được tính theo công thức:

$$p(i) = \frac{f(i)}{\sum_{i=1..N} f(i)}$$

  Phương pháp bánh xe ru lét có thể cài đặt bằng thuật toán sau:

  + Tính tổng giá trị thích nghi của tất cả cá thể trong quần thể, gọi tổng này là S.
  + Sinh ra số ngẫu nhiên trong khoảng (0, S), gọi số này là r.
  + Lần lượt cộng giá trị thích nghi của từng cá thể trong quần thể, gọi tổng này là s. Khi nào s > r thì dừng và trả về cá thể hiện thời.

- *Thi đấu*: chọn ngẫu nhiên một đôi cá thể sử dụng phân bố xác suất đều, sau đó chọn cá thể tốt hơn trong hai cá thể đó như một cá thể cha mẹ. Thực hiện tương tự để chọn cá thể cha mẹ còn lại.
- *Lựa chọn cá thể tinh hoa* (Elitism). Các phương pháp trên đều không đảm bảo cá thể tốt nhất được lựa chọn. Trong phương pháp lựa chọn tinh hoa, một số lượng nhất định các cá thể tốt nhất được lựa chọn trước, sau đó phần còn lại được lựa chọn theo các phương pháp ru lét hay thi đấu như ở trên. Như vậy, các cá thể tốt nhất luôn được duy trì đoạn gen của mình sang thế hệ sau và tránh làm mất lời giải tốt nhất đã tìm được.
- Loại bỏ các cá thể có hàm thích nghi nhỏ hơn một ngưỡng nhất định, sử dụng các cá thể còn lại để lai ghép và tạo quần thể mới.

Giá trị xác suất.

Xác suất lai ghép. Nếu xác suất lai ghép là 1 (100%) thì toàn bộ cá thể con sẽ được tạo ra do lai ghép. Ngược lại, nếu xác suất lai ghép là 0 thì toàn bộ cá thể con là bản sao của một số cá thể bố mẹ nhưng không nhất thiết quần thể tiếp theo trùng với quần thể cũ. Như đã nói ở trên, xác suất lai ghép được lựa chọn tương đối lớn, thường từ 0.5 trở lên.

Xác suất đột biến. Nếu xác suất đột biến là 100% thì toàn bộ cá thể sau khi lai ghép sẽ bị thay đổi. Ngược lại, nếu xác suất này là 0 thì không cá thể nào bị thay đổi. Xác suất đột biến được lựa chọn rất nhỏ, ít khi vượt quá 0.1. Xác suất đột biến nhỏ để tránh cho thuật toán di chuyển theo kiểu ngẫu nhiên.

Kích thước quần thể.

Kích thước quần thể N là số lượng cá thể được duy trì trong mỗi thế hệ. Nếu N quá nhỏ, thuật toán có ít lựa chọn để thực hiện lai ghép, dẫn tới chỉ một phần không gian tìm kiếm được khảo sát và do vậy có thể không tìm được lời giải tốt. Nếu N quá lớn, thuật toán sẽ thực hiện chậm do phải xử lý nhiều trong mỗi vòng lặp. Giá trị tốt của N phụ thuộc vào bài toán cụ thể và cách mã hoá lời giải. Tuy nhiên, nhiều kết quả thực nghiệm cho thấy, khi N tăng tới một mức độ nào đó, chất lượng lời giải không tăng, trong khi thuật toán sẽ chậm hơn.

Mã hoá lời giải

Lời giải hay cá thể cần được mã hoá dưới dạng nhiễm sắc thể. Cách mã hoá cụ thể phụ thuộc rất nhiều vào từng bài toán. Sau đâu là một số cách mã hoá thường gặp.

- Mã hoá dưới dạng chuỗi bit hay dạng nhị phân. Mỗi nhiễm sắc thể là một vec tơ gồm các số 0 hoặc 1 như ví dụ trên hình 2.23. Đây là cách mã hoá thông dụng nhất. Cách mã hoá tạo ra rất nhiều nhiễm sắc thể. Tuy nhiên, với nhiều bài toán, cách mã hoá này không tự nhiên và việc lai ghép hay đột biến có thể tạo ra nhiễm sắc thể không tương ứng với lời giải hợp lệ nào và do vậy cần xử lý để nhận được lời giải hợp lệ.
- Mã hoá dưới dạng các hoán vị. Cách mã hoá này thường sử dụng trong bài toán cần xác định thứ tự như đường đi hay thời khoá biểu. Ví dụ, trong bài toán người bán hàng đã trình bầy trong phần trước, mỗi gen tương ứng với một thành phố và nhiễm sắc thể mã hoá thứ tự đường đi dưới dạng hoán vị thứ tự thành phố (1 ứng với A, 2 với B v.v.).

Nhiễm sắc thể	1	5	3	2	6	4	7	9	8

- Mã hoá dưới dạng các giá trị. Trong một số trường hợp, các giá trị như số thực được sử dụng để biểu diễn lời giải. Việc sử dụng trực tiếp số thực là cần thiết do việc biểu diễn bằng bit trong trường hợp này rất phức tạp. Với cách mã hoá này, nhiễm sắc thể là một chuỗi giá trị, trong đó giá trị có thể là số thực , số nguyên, chữ cái hoặc xâu ký tự. Trong bài toán 8 quân hậu ở trên, ta đã thấy vị trí các con hậu được biểu diễn bởi vec tơ các số nguyên như trên hình 2.24. Một ví dụ khác là khi huấn luyện mạng nơ ron, cần xác định trọng số cho các kết nối. Nhiễm sắc thể khi đó chứa các trọng số này. Dưới đây là ví dụ một số nhiễm sắc thể khi mã hoá kiểu này;

Nhiễm sắc thể	1.2324 5.3243 0.4556 2.3293 2.4545
Nhiễm sắc thể	ABDJEIFJDHDIERJFDL
Nhiễm sắc thể	(trên) (dưới) (dưới) (trái) (trên) (phải)

Các phương pháp lai ghép

Thao tác lai ghép rất đa dạng và phụ thuộc vào cấu trúc của lời giải trong bài toán cụ thể. Các lựa chọn bao gồm:

- Lựa chọn điểm cắt. Thông thường sử dụng cùng điểm cắt trên cả bố và mẹ nhưng cũng có thể sử dụng điểm cắt khác nhau. Trường hợp sau sinh ra cá thể con có độ dài thay đổi và do vậy cần xử lý riêng sau đó.
- Lựa chọn số đoạn gen. Thông thường nhiễm sắc thể được cắt làm hai phần nhưng cũng có thể nhiều hơn.

  Ví dụ lai ghép với một điểm cắt:

  11001|011+11011|**111** = **11001111**

  Ví dụ lai ghép với hai điểm cắt:

  11|0010|**11** + 11|**0111**|11 = **11011111**

  Với cách mã hoá kiểu hoán vị như trong bài toán người bán hàng, khi lai ghép với một điểm cắt, phần trước điểm cắt được lấy nguyên từ cá thể bố mẹ thứ nhất. Phần sau điểm cắt được tạo ra bằng cách lấy các số chưa được sử dụng từ cá thể bố mẹ thứ hai (thay vì lấy nguyên cả đoạn từ sau điểm cắt). Cách này đảm bảo để mỗi số chỉ xuất hiện đúng một lần (cần lưu ý là có những cách khác cho phép đảm bảo điều này). Ví dụ:

  (**1 2 3 4 5** 6 7 8 9) + (4 5 3 6 8 9 7 2 1) = (**1 2 3 4 5** 6 8 9 7)
- Lai ghép đều. Trong phương pháp này, mỗi gen được sao chép từ bố hoặc từ mẹ với xác suất bằng nhau.

  Ví dụ lai ghép đều:

  1**11**01**011** + **1**10**111**01 = **11111111**
- Lai ghép số học. Áp dụng phép tính số học hoặc logic trên cặp bit tương ứng của bố và mẹ. Ví dụ:

  11001011 + 11011111 = 11001001 (AND)

- Có thể sử dụng ba cha mẹ thay vì hai.

Các phương pháp tạo đột biến

Đối với cách mã hoá nhị phân, đột biến được thực hiện bằng cách đổi giá trị các bit được chọn.

Đối với mã hoá dạng hoán vị: hai số được chọn và đổi vị trí cho nhau. Ví dụ:

(1 **2** 3 4 5 6 **8** 9 7) => (1 **8** 3 4 5 6 **2** 9 7)

Đối với mã hoá dạng vec tơ các giá trị: thay đổi giá trị các gen được chọn như trong ví dụ 8 quân hậu ở trên.

4) Hiệu ứng của các thao tác

Các thao tác chọn lọc, lai ghép và đột biến có các hiệu ứng như sau khi được sử dụng riêng rẽ hoặc kết hợp với các thao tác khác.

- Nếu chỉ sử dụng thao tác chọn lọc thì thuật toán có xu hướng chọn cá thể tốt nhất từ thế hệ trước. Như vậy chỉ có thể tăng độ thích nghi trung bình và tránh lời giải ít thích nghi chứ không tạo ra lời giải tốt hơn lời giải tốt nhất của thế hệ trước.
- Kết hợp chọn lọc và lai ghép có xu hướng khiến thuật toán hội tụ ở lời giải tương đối tốt nhưng không tối ưu.
- Sử dụng một mình thao tác đột biến sinh ra chuyển động ngẫu nhiên. Nếu chỉ sử dụng một mình thao tác đột biến, thuật toán sẽ không di chuyển về hướng có lời giải tốt.
- Sử dụng chọn lọc kết hợp với đột biến tạo ra một dạng leo đồi song song ổn định hơn leo đồi thông thường.

Như vậy, giải thuật di truyền có thể coi như một biến thể mở rộng của tìm kiếm leo đồi ngẫu nhiên thực hiện song song trên một tập hợp các lời giải.

2.5.4. Một số thuật toán tìm kiếm cục bộ khác

Ngoài phương pháp leo đồi và tôi thép, nhiều thuật toán tìm kiếm cục bộ khác được đề xuất và sử dụng trong thực tế, trong đó phải kể đến những thuật toán sau:

1) Tìm kiếm tabu.

Tìm kiếm tabu (Tabu search) có thể coi như sự kết hợp của leo đồi với một số kỹ thuật cho phép tránh cực trị địa phương. Cụ thể, thuật toán cho phép chuyển sang các láng giềng không tốt hơn trạng thái hiện thời. Tuy nhiên, để tránh xét lại các trạng thái đã sử dụng trước đó, thuật toán lưu một danh sách những trạng thái đã đi qua gọi là tabu list (tạm dịch là danh sách cấm). Khi duyệt tập hàng xóm, những trạng thái thuộc danh sách này sẽ bị loại và không được xem xét nữa. Kích thước của danh sách cấm là hữu hạn và sau một thời gian, các nút được lưu vào danh sách cấm từ trước sẽ bị đẩy ra khỏi danh sách và lại có thể xem xét lại.

2) Tìm kiếm chùm cục bộ

Tìm kiếm chùm cục bộ (Local beam search) là phương pháp tìm kiếm cục bộ tương tự leo đồi nhưng thay vì chỉ lưu một trạng thái tại mỗi thời điểm, tìm kiếm chùm lưu k trạng thái. Ở mỗi vòng lặp, thuật toán sinh ra tất cả láng giềng của toàn bộ k trạng thái và kiểm tra các láng giềng này. Nếu một trong các trạng thái mới sinh ra là đích thì thuật toán dừng và trả về

kết quả. Nếu ngược lại, thuật toán chọn *k* trạng thái tốt nhất trong số mới sinh ra và chuyển sang vòng lặp tiếp theo. Như vậy, thuật toán đồng thời duy trì một "chùm" gồm *k* trạng thái.

Hiệu quả của việc duy trì *k* trạng thái là nếu một trạng thái sinh ra các láng giềng tốt hơn các trạng thái còn lại thì thuật toán sẽ tập trung vào các láng giềng của trạng thái này và như vậy sẽ bỏ hướng tìm kiếm sinh ra từ các trạng thái còn lại. Điều này giúp tìm kiếm chùm có kết quả khác với việc chạy đồng thời *k* phiên bản leo đồi ngẫu nhiên. Tìm kiếm chùm sẽ nhanh chóng từ bỏ những hướng tìm kiếm trông có vẻ không hứa hẹn.

Do có xu hướng từ bỏ sớm những hướng tìm kiếm trông không hứa hẹn, tìm kiếm chùm có xu hướng tập trung vào một vùng nhỏ trong không gian tìm kiếm và bỏ qua các vùng có lời giải tối ưu. Để giảm bớt hiệu ứng này, có thể sử dụng phiên bản tìm kiếm chùm ngẫu nhiên, theo đó thay vì giữ lại *k* trạng thái tốt nhất cho vòng lặp sau, thuật toán chọn ngẫu nhiên *k* trạng thái sao cho trạng thái tốt hơn có xác suất lớn hơn. Cách chọn này tương tự thao tác chọn lọc sử dụng bánh xe ru lét trong giải thuật di truyền.

Phương pháp tối ưu đàn kiến

Tối ưu đàn kiến (Ant colony optimization) là phương pháp tối ưu xác suất được Marco Dorigo đề xuất vào năm 1992, ban đầu để giải bài toàn tìm đường tối ưu trên đồ thị. Phương pháp tối ưu đàn kiến sau đó có thể mở rộng để giải quyết các bài toán tối ưu bằng cách quy về bài toán tìm đường ngắn nhất trên đồ thị.

Ý tưởng phương pháp này dựa trên hành vi của đàn kiến khi di chuyển giữa tổ và nơi có thức ăn. Trong trường hợp giữa tổ kiến và nguồn thức ăn có nhiều đường đi, sau một thời gian đàn kiến sẽ tập trung đi theo đường đi ngắn nhất mặc dù kiến không có bản đồ và có thể mù. Kết quả này đạt được nhờ cơ chế sau. Khởi đầu kiến di chuyển ngẫu nhiên theo các đường đi có thể giữa hai điểm. Trong quá trình di chuyển, các con kiến để lại trên đường đi các pheromone, một dạng chất sinh học mà kiến sinh ra và có thể cảm nhận được. Nếu kiến đi theo đường ngắn hơn thì sẽ mau trở lại hơn và chuyển động được nhiều lần hơn trên đường ngắn hơn này. Kết quả là sau một thời gian, đường đi ngắn hơn sẽ có lượng pheromone nhiều hơn. Các con kiến đi sau đó sẽ tập trung theo đường đi có nhiều pheromone, tức là đường ngắn.

Thuật toán tối ưu đàn kiến được thực hiện bằng cách lựa chọn các cung trên đồ thị với xác suất tỷ lệ thuận với lượng pheromone trên cung đó. Mỗi khi di chuyển qua một cung, thuật toán cũng cập nhật để tăng lượng pheromone trên cung. Phương pháp tối ưu đàn kiến được sử dụng trong một số bài toán tối ưu như định tuyến mạng, quy hoạch mạng giao thông. Ưu điểm của phương pháp này so với tôi thép và giải thuật di truyền là có thể thực hiện rất nhanh trong trường hợp cấu trúc mạng thay đổi. Nhờ vậy, thuật toán cho phép tìm giải pháp cho bài toàn định tuyến mạng khi cấu trúc mạng là động và thay đổi theo thời gian thực.

2.6. ỨNG DỤNG MINH HOẠ

Trên thực tế, tìm kiếm cục bộ và tìm kiếm heuristic được sử dụng rộng rãi nhất do khả năng giải quyết những bài toán thực kích thước lớn. Các thuật toán heuristic và tìm kiếm cục bộ được dùng để giải quyết nhiều bài toán tối ưu như tìm đường đi, thiết kế vi mạch, lập lịch sản xuất, làm thời khoá biểu v.v.

Trong phần 2.5.1, ta đã xem xét cách giải quyết bài toán người bán hàng bằng thuật toán leo đồi. Bài toán người bán hàng cũng có thể giải quyết bằng thuật toán tôi thép với cùng cách lựa chọn trạng thái xuất phát, chuyển động và hàm mục tiêu như đã trình bầy cho leo đồi. Để giải quyết bằng giải thuật di truyền có thể dùng cách biểu diễn lời giải và các thao tác lựa chọn, lai ghép, đột biến như trình bầy trong phần về giải thuật di truyền. Các thực nghiệm cho thấy thuật toán tôi thép và giải thuật di truyền thường cho kết quả tốt hơn (đường đi ngắn hơn) so với leo đồi.

Trong phần này, ta sẽ xem xét cách sử dụng tìm kiếm cục bộ, cụ thể là thuật toán leo đồi ngẫu nhiên và tìm kiếm tabu, trong một bài toán có nhiều ứng dụng: **xếp thời khoá biểu cho trường trung học** (cơ sở và phổ thông). Các kỹ thuật trình bầy ở đây cũng có thể áp dụng trong xếp thời khoá biểu cho trường đại học với một số bổ sung và điều chỉnh phù hợp.

Yêu cầu chung: Xếp thời khoá biểu là bài toán xác định giáo viên nào dậy lớp nào vào tiết nào sao cho thoả mãn một số ràng buộc như không có giáo viên nào dậy hai lớp cùng một lúc hay không có lớp nào học hai môn cùng một lúc v.v. Để đơn giản, ta sẽ giả sử thời khoá biểu của các tuần là giống nhau và do vậy chỉ cần lập thời khoá biểu cho một tuần.

Phát biểu bài toán: Giả sử có m lớp học c_1, …, c_m , n giáo viên t_1,…,t_n , và p tiết học trong một tuần 1,…,p. Ma trận R kích thước m x n thể hiện phân công giáo viên, sao cho phần tử r_{ij} là số tiết mà giáo viên t_j phải dậy cho lớp c_i trong một tuần. Hai ma trận $T_{n \times p}$ và $C_{m \times p}$ thể hiện các tiết mà giáo viên và lớp có thể dậy hoặc học. Cụ thể, $t_{jk} = 1$ nếu giáo viên j có thể dậy vào tiết k, và $t_{jk} = 0$ nếu ngược lại, chẳng hạn giáo viên nữ có con nhỏ không phải dậy tiết đầu buổi sáng. Tương tự, $c_{ik} = 1$ (hoặc 0) nếu lớp i có thể học (hoặc không) vào tiết k, chẳng hạn có thể quy định để mỗi lớp có một số tiết trống vào những giờ nhất định dành cho ngoại khoá. Tiếp theo, $D_{m \times p}$ là ma trận ràng buộc sao cho $t_{jk} = 1$ nếu lớp i bắt buộc phải học vào tiết k và $t_{jk} = 0$ nếu không bắt buộc.

Bài toán xếp thời khoá biểu được phát biểu như sau:

Tìm x_{ijk} $(i = 1..m, j = 1..n, k = 1..p)$

Thoả mãn các ràng buộc sau:

$x_{ijk} = 0$ hoặc 1 $\quad (i = 1..m, j = 1..n, k = 1..p)$ (1) // $x_{ijk} = 1$ nếu giáo viên j dậy lớp i vào tiết k và $x_{ijk} = 0$ nếu ngược lại.

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i = 1..m, j = 1..n) \qquad (2)$$

$$\sum_{i=1}^{m} x_{ijk} \le t_{jk} \qquad (j = 1..n, k = 1..p) \qquad (3)$$

$$\sum_{j=1}^{n} x_{ijk} \le c_{ik} \qquad (\mathrm{i} = 1..m, k = 1..p) \qquad (4)$$

$$\sum_{j=1}^{n} x_{ijk} \ge d_{ik} \qquad (\mathrm{i} = 1..m, k = 1..p) \qquad (5)$$

Ràng buộc cứng: các ràng buộc (1) – (5) ở trên là các ràng buộc cứng, tức là ràng buộc mà lời giải buộc phải thoả mãn. Trong trường hợp cụ thể có thể thêm hoặc bớt ràng buộc cứng. Chẳng hạn có thể thêm ràng buộc thể hiện việc học lớp ghép trong một số môn học.

Ràng buộc mềm. Ngoài ràng buộc cứng còn có ràng buộc mềm, tức là ràng buộc không bắt buộc nhưng mong muốn thoả mãn càng nhiều càng tốt. Ràng buộc mềm sẽ đóng góp vào hàm mục tiêu của bài toán. Dưới đây là một số ràng buộc mềm thường gặp. Để tiện cho việc xây dựng hàm mục tiêu ở bên dưới, mỗi ràng buộc mềm được gán một trọng số w thể hiện điểm phạt khi một ràng buộc mềm như vậy bị vi phạm.

- Gián đoạn ($w_1 = 6$): hai tiết của một giáo viên cho cùng một lớp không nằm liền nhau.
- Tiết trống ($w_2 = 1$): giáo viên có tiết trống giữa các tiết dậy.
- Dậy quá ít ($w_3 = 3$): giáo viên dậy ít hơn mức tối thiểu trong một ngày.
- Dậy quá nhiều ($w_4 = 3$): giáo viên dậy nhiều hơn mức tối đa trong một ngày.
- Phân bố không đều ($w_5 = 5$): một giáo viên dậy quá nhiều tiết cho cùng một lớp trong một ngày (chẳng hạn nhiều hơn 2 tiết). Ràng buộc này đảm bảo các tiết của một giáo viên cho một lớp phân bố đều trong tuần.
- Dậy vào thời gian không mong muốn ($w_6 = 3$): giáo viên có thể đăng ký một số giờ không mong muốn. Số vi phạm được chuẩn hoá bằng cách chia đều cho số đăng ký.
- Không có giờ thực hành ($w_7 = 10$): không xếp được tiết liền nhau cho các môn có giờ thực hành.

Cần lưu ý rằng các ràng buộc trên có thể thay đổi (thêm/bớt) cho từng trường hợp cụ thể. Giá trị mặc định của trọng số được dùng để minh hoạ và có thể thay đổi tuỳ vào độ quan trọng của từng loại ràng buộc với mỗi trường cụ thể.

Hàm mục tiêu. Để sử dụng với thuật toán tìm kiếm cục bộ, các ràng buộc cứng và mềm cần được kết hợp thành một hàm mục tiêu duy nhất. Hàm mục tiêu được tính bằng số ràng buộc cứng bị vi phạm nhân với trọng số cho ràng buộc cứng cộng với số ràng buộc mềm bị vi phạm nhân với trọng số tương ứng. Các ràng buộc bị vi phạm được tính trên toàn bộ giáo viên và lớp.

$$\textit{Hàm mục tiêu} = \textit{số ràng buộc cứng vi phạm} * w_0 + \textit{số ràng buộc mềm loại } i \textit{ vi phạm} * w_i$$

Giá trị trọng số mặc định w_0 cho mỗi ràng buộc cứng bị vi phạm được chọn cao hơn ràng buộc mềm. Chẳng hạn, có thể chọn trọng số cho ràng buộc cứng: $w_0 = 20$.

Biểu diễn thời khoá biểu. Để thuật tiện cho việc thực hiện các chuyển động, thời khoá biểu (tức là các giá trị x_{ijk}) được biểu diễn dưới dạng ma trận $B_{n \times k}$, trong đó mỗi dòng tương ứng với một giáo viên, mỗi cột tương ứng với một tiết. Phần tử b_{jk} chứa số thứ tự của lớp mà giáo viên j dậy vào tiết k. Giá trị $b_{jk} = 0$ nếu giáo viên không dậy nếu giáo viên không dậy vào tiết đó. Giá trị lớn hơn số lớp có thể dùng để biểu diễn các hoạt động khác của giáo viên.

Bên cạnh cách biểu diễn này, có thể sử dụng cách biểu diễn trong đó dòng tương ứng với lớp, cột với tiết học và mỗi phần tử chứa số thứ tự giáo viên dậy cho lớp tương ứng vào tiết tương ứng. Ở đây, ta sẽ sử dụng cách biểu diễn bằng ma trận B như đề cập ở trên.

Trên hình 2.27 là một phần của thời khoá biểu với cách biểu diễn dùng mà trận B. Để tiện cho việc hình dung, trên hình vẽ thể hiện tên lớp thay vì số thứ tự được dùng trong thuật toán. Các ô trống tương ứng với giờ mà giáo viên không dậy. Các ô "--" tương ứng với giờ mà giáo viên không thể dậy (tương ứng $t_{jk} = 0$).

Cần lưu ý rằng, với cách biểu diễn như vậy, việc một giáo viên dậy nhiều hơn một lớp tại một thời điểm là không thể, do vậy ràng buộc (3) luôn được thoả mãn một phần.

Giáo viên (số thứ tự)	Thứ hai	Thứ ba	…
	t1 t2 t3 t4 t5 t1 t2 t3 t4 t5	t1 t2 t3 t4 t5 t1 t2 t3 t4 t5	
1	8A 8A 9B 9B	-- -- -- -- -- -- -- -- -- --	…
2	9A 9A 7A 7A	7B 7B 7C 7C	
3	-- 8A 8B 8C 7B 7C	9B 9B 7C 7C	
4	-- -- -- -- -- -- -- -- -- --	6A 6C 6B 6D	
5	6B 6B 7A 7A	6A 16A 7B 7B	
…	…	…	

Hình 2.27. Một phần thời khoá biểu

Khởi tạo. Trạng thái đầu tiên (thời khoá biểu đầu tiên) được khởi tạo bằng cách xếp lịch cho từng giáo viên một cách ngẫu nhiên sao cho ràng buộc (2) được thoả mãn, tức là tổng số giờ lên lớp cho mỗi giáo viên được đảm bảo.

Chuyển động. Để sử dụng các thuật toán như leo đồi, tôi thép hay tìm kiếm tabu, ta cần xác định các chuyển động cho phép sinh ra các trạng thái (thời khoá biểu) láng giềng từ trạng thái hiện thời. Các nghiên cứu trước đây cho thấy có thể sử dụng kết hợp hai loại chuyển động sau.

Dạng thứ nhất là dạng *chuyển động đơn*, được thực hiện bằng cách đổi chỗ hai giờ giảng của cùng một giáo viên, hoặc, nếu một trong hai giờ là giờ trống thì chuyển giờ dậy sang giờ trống đó. Chuyển động này được xác định bằng bộ ba $<t, p_1, p_2>$.

Dạng thứ hai là dạng *chuyển động kép*, được tạo thành từ hai chuyển động đơn, sao cho chuyển động đơn thứ hai khắc phục ràng buộc cứng mà chuyển động đơn thứ nhất phá vỡ. Trong trường hợp chuyển động đơn thứ nhất không dẫn tới việc vi phạm ràng buộc cứng thì chuyển động đơn thứ hai sẽ được bỏ qua và chuyển động kép trở thành chuyển động đơn thông thường.

Thuật toán tìm kiếm. Cách biểu diễn bài toán và chuyển động như trên có thể sử dụng trong các thuật toán leo đồi, tôi thép, tìm kiếm Tabu. Một số nghiên cứu trước đây cho thấy có thể kết hợp leo đồi ngẫu nhiên và tìm kiếm Tabu để cho ra kết quả tốt. Cụ thể, quá trình tìm kiếm được thực hiện như sau.

Bước 1: Khởi tạo như mô tả ở trên.

Bước 2: Thuật toán leo đồi ngẫu nhiên được sử dụng để cải thiện lời giải hiện thời. Thuật toán leo đồi ngẫu nhiên được sửa đổi để cho phép đi ngang, tức là chuyển sang các trạng thái tốt bằng trạng thái hiện thời tối đa N lần (N là tham số).

Bước 3: Sau khi leo đồi ngẫu nhiên dừng lại do không cải thiện được nữa, thuật toán Tabu sẽ được thực hiện trên lời giải do leo đồi ngẫu nhiên trả về.

Bước 4: Lặp lại bước 2 và bước 3 M lần (M là tham số).

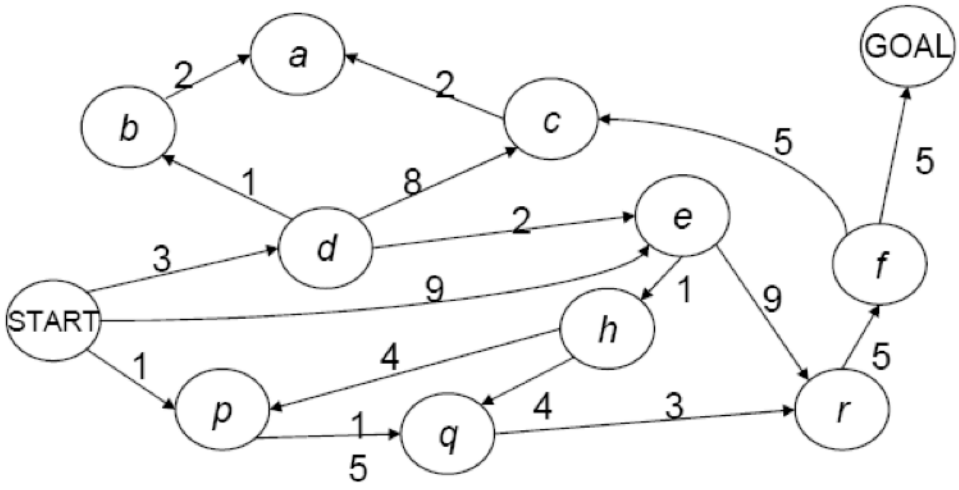
Việc sử dụng leo đồi ngẫu nhiên có hai tác dụng. Thứ nhất, leo đồi ngẫu nhiên tạo ra trạng thái xuất phát cho tìm kiếm Tabu để tiết kiệm thời gian do thời gian thực hiện Tabu lâu hơn leo đồi. Thứ hai, sau khi Tabu không thể cải thiện tiếp lời giải, leo đồi cho phép di chuyển sang vùng lời giải khác để bước thực hiện Tabu tiếp theo có thể khảo sát vùng không gian lời giải khác vòng chạy trước. Cách kết hợp hai thuật toán cho phép tìm ra lời giải chất lượng hơn với độ phức tạp tính toán chập nhận được.

Kết quả. Thực nghiệm với các trường phổ thông với trên 30 lớp và trên 50 giáo viên cho thấy các kỹ thuật trình bầy ở trên tìm ra thời khoá biểu có chất lượng tốt hơn nhiều (hàm mục tiêu nhỏ hơn nhiều) so với thời khoá biểu xếp bằng tay và các phương pháp heuristic. Quá trình xếp thời khoá biểu có thể thực hiện trên máy tính cá nhân thông thường trong vòng vài phút tới vài chục phút tuỳ kích thước trường.

2.7. CÂU HỎI VÀ BÀI TẬP CHƯƠNG

1. Giả sử ta có ba can đựng nước với dung tích 3 lít, 8 lít và 12 lít. Ta có thể đổ nước đầy các can hoặc rót toàn bộ nước trong can ra ngoài hoặc sang can khác. Cần tìm cách đổ đầy và rót nước khỏi can để đong được 1 lít nước. Trình bầy dưới bài toán này dưới dạng bài toán tìm kiếm và viết chương trình để tìm lời giải, sử dụng một thuật toán tìm kiếm phù hợp.
2. Bài toán nhà truyền giáo và người ăn thịt người. Có ba nhà truyền giáo và ba người thuộc bộ lạc ăn thịt người ở trên bờ một con sông. Cần chuyển cả sáu người sang bờ bên kia bằng một con thuyền có thể chở tối đa hai người. Yêu cầu đặt ra là không có lúc nào số người ăn thịt trên một bờ sông hoặc trên thuyền lớn hơn số nhà truyền giáo.
   - Hãy phát biểu bài toán dưới dạng bài toán tìm kiếm trong không gian trạng thái và sử dụng thuật toán tìm kiếm phù hợp để tìm ra lời giải.
   - Xây dựng chương trình máy tính để thực hiện hiện thuật toán.
3. Các khẳng định sau đúng hay sai, giải thích tại sao:
   - Để tìm được lời giải, tìm theo chiều sâu không bao giờ mở rộng ít nút hơn tìm kiếm A* với hàm heuristic chấp nhận được.
   - h(n) = 0 là hàm heuristic chấp nhận được cho bài toán 8 quân hậu.
   - Tìm theo chiều rộng là đầy đủ kể cả khi giá thành đường đi giữa hai trạng thái có thể bằng không.
4. Giả sử cần tìm chuỗi các link cho phép di chuyển từ trang Web này sang trang Web khác.
   - Hãy lựa chọn thuật toán tìm kiếm phù hợp cho bài toán này và viết chương trình hiện thức hóa thuật toán.
   - Việc sử dụng tìm kiếm theo hai hướng cho bài toán này có hiệu quả không?
5. Các khẳng định sau là đúng hay sai, giải thích (chứng minh) câu trả lời:
   - Tìm theo chiều rộng là trường hợp riêng của tìm theo giá thành thống nhất.
   - Tìm theo giá thành thống nhất là trường hợp riêng của tìm kiếm A*.

6. Cho đồ thị trên hình sau:



Hãy xác định đường đi từ START tới GOAL sử dụng các thuật toán tìm kiếm sau:

a) Tìm theo chiều rộng.

b) Tìm theo chiều sâu.

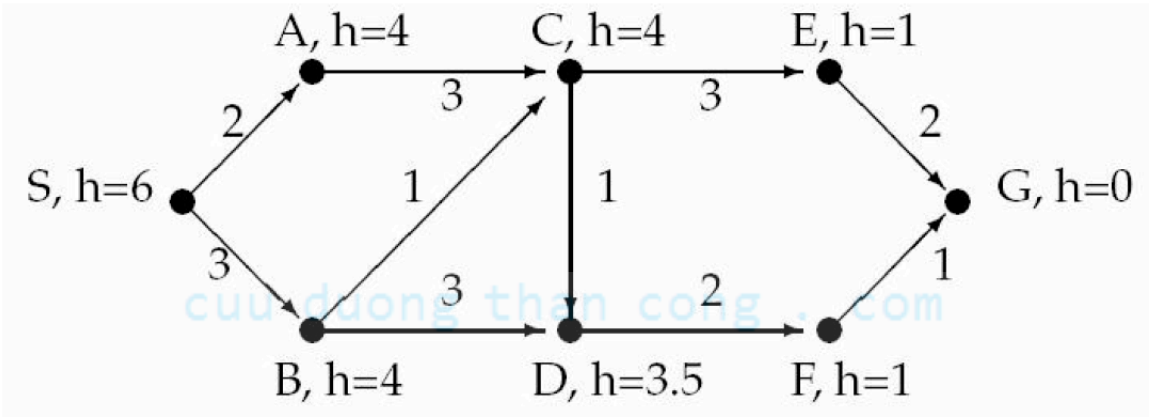
c) Tìm theo giá thành thống nhất.

d) Tìm kiếm sâu dần.

Thể hiện nút được mở rộng và danh sách các nút trong tập biên tại mỗi vòng lặp của thuật toán. Sử dụng con trỏ ngược để khôi phục lại đường đi khi tìm được nút đích.

Hãy cho biết trong trường hợp nào đường đi tìm được là ngắn nhất.

7. Cho đồ thị trên hình sau:



trong đó giá thành đường đi giữa hai nút được thể hiện cạnh cung tương ứng, giá trị hàm heuristic h được thể hiện bên cạnh các nút.

a) Hãy cho biết hàm h có phải là hàm heuristic chấp nhận được hay không? Tại sao?

b) Tìm đường đi từ S tới G sử dụng thuật toán tìm kiếm tham lam với hàm h là hàm heuristic.

c) Thay đổi giá trị h tại nút D thành h = 3, sau đó tìm đường đi từ S tới G sử dụng thuật toán A* với h là hàm heuristic. Hãy cho biết đường đi trong trường hợp này có phải là đường đi ngắn nhất không.

d) Thay đổi giá trị h tại nút D thành h = 5, sau đó tìm đường đi từ S tới G sử dụng thuật toán A* với h là hàm heuristic. Hãy cho biết đường đi trong trường hợp này có phải là đường đi ngắn nhất không.

e) Sử dụng thuật toán A* sâu dần để tìm đường đi từ S tới G với bước nhẩy bằng 2. Hãy cho biết đường đi tìm được có tối ưu không?

f) Sử dụng thuật toán A* sâu dần để tìm đường đi từ S tới G với bước nhẩy bằng 3. Hãy cho biết đường đi tìm được có tối ưu không? So sánh với kết quả ở câu e.

8. Viết chương trình giải bài toán n quân hậu với n lớn (từ 1000 trở lên) sử dụng thuật toán leo đồi với trạng thái xuất phát được khởi tạo ngẫu nhiên nhiều lần. Có thể sử dụng không gian trạng thái với đầy đủ cả n quân hậu, mỗi quân nằm trong một cột. Thử nghiệm và so sánh kết quả sử dụng các kiểu chuyển động khác nhau: thay đổi vị trí 1, 2, 3, 4, quân hậu. Trong trường hợp nào thuật toán tìm được lời giải tốt hơn. Trong trường hợp nào thời gian thực hiện thuật toán ngắn hơn.
9. Viết chương trình giải bài toán n quân nhưhậu với n lớn, sử dụng thuật toán tôi thép. Có thể thử nghiệm các kiểu chuyển động tương tự như ở bài tập 8.

CHƯƠNG 3: BIỂU DIỄN TRI THỨC VÀ LẬP LUẬN LOGIC

3.1. SỰ CẦN THIẾT SỬ DỤNG TRI THỨC TRONG GIẢI QUYẾT VẤN ĐỀ

Sự cần thiết của tri thức và lập luận

Một yêu cầu quan trọng đối với hệ thống thông minh là phải có khả năng sử dụng tri thức về thế giới xung quanh và lập luận (reasoning) với tri thức đó. Rất khó để đạt được những hành vi thông minh và mềm dẻo mà không có tri thức về thế giới xung quanh và khả năng suy diễn với tri thức đó. Sử dụng tri thức và lập luận đem lại những lợi ích sau.

- Hệ thống dựa trên tri thức có tính mềm dẻo cao. Việc kết hợp tri thức và lập luận (bao gồm suy diễn và suy luận) cho phép tạo ra tri thức khác, giúp hệ thống đạt được những mục tiêu khác nhau, đồng thời có khả năng lập luận về bản thân mục tiêu. Chương trước đã đề cập tới kỹ thuật giải quyết vấn đề bằng cách tìm kiếm. Những hệ thống tìm kiếm chỉ sử dụng tri thức hạn chế, thể hiện trong việc biểu diễn bài toán (như cách sinh ra các chuyển động) và các heuristic. Hệ thống như vậy không có khả năng tự thay đổi mục đích cũng như không có khả năng hành động một cách mềm dẻo, ngoài những gì chứa trong giải thuật và mô tả bài toán. Vì vậy kỹ thuật tìm kiếm là chưa đủ để tạo ra hệ thống thông minh.
- Sử dụng tri thức và lập luận cho phép hệ thống hoạt động cả trong trường hợp thông tin quan sát về môi trường là không đầy đủ. Hệ thống có thể kết hợp tri thức chung đã có để bổ sung cho thông tin quan sát được khi cần ra quyết định. Ví dụ, khi giao tiếp bằng ngôn ngữ tự nhiên, có thể hiểu một câu ngắn gọn nhờ sử dụng tri thức đã có về ngữ cảnh giao tiếp và nội dung liên quan tới chủ đề.
- Việc sử dụng tri thức thuận lợi cho việc xây dựng hệ thống. Thay vì lập trình lại hoàn toàn hệ thống, có thể thay đổi tri thức trang bị cho hệ thống và mô tả mục đích cần đạt được, đồng thời giữ nguyên thủ tục lập luận.

Các hệ thống có sử dụng tri thức được gọi là hệ dựa trên tri thức. Hệ thống loại này gồm thành phần cơ bản là *cơ sở tri thức* (tiếng Anh là Knowledge Base, viết tắt là KB). Cơ sở tri thức gồm các câu hay các công thức trên một ngôn ngữ nào đó và chứa các tri thức về thế giới của bài toán. Cùng với cơ sở tri thức, hệ thống còn có khả năng *lập luận*, gồm cả *suy diễn* (inference) và *suy luận* (deduction), cho phép đưa ra các hành động hoặc câu trả lời hợp lý dựa trên tri thức và thông tin quan sát được. Thực chất, suy diễn hay suy luận là cách tạo ra các câu mới từ những câu đã có. Như vậy, một hệ dựa trên tri thức bao gồm cơ sở tri thức và thủ tục suy diễn.

Biểu diễn tri thức

Để có thể sử dụng tri thức, tri thức cần được biểu diễn dưới dạng thuận tiện cho việc mô tả và suy diễn. Nhiều ngôn ngữ và mô hình biểu diễn tri thức đã được thiết kế để phục vụ mục đích này. Ngôn ngữ biểu diễn tri thức phải là *ngôn ngữ hình thức* để tránh tình trạng nhập nhằng như thường gặp trong ngôn ngữ tự nhiên. Một ngôn ngữ biểu diễn tri thức tốt phải có những tính chất sau:

- Ngôn ngữ phải có khả năng biểu đạt tốt, tức là cho phép biểu diễn mọi tri thức và thông tin cần thiết cho bài toán.
- Cần đơn giản và hiệu quả, tức là cho phép biểu diễn ngắn gọn tri thức, đồng thời cho phép đi đến kết luận với khối lượng tính toán thấp.
- Gần với ngôn ngữ tự nhiên để thuận lợi cho người sử dụng trong việc mô tả tri thức.

Sau khi đã có ngôn ngữ biểu diễn tri thức, tri thức về thế giới của bài toán được biểu diễn dưới dạng tập hợp các *câu* hay các *công thức* và tạo thành *cơ sở tri thức* (ký hiệu KB trong các phần sau). Thủ tục suy diễn được sử dụng để tạo ra những câu mới nhằm trả lời cho các vấn đề của bài toán. Thay vì trực tiếp hành động trong thế giới thực của bài toán, hệ thống có thể suy diễn dựa trên cơ sở tri thức được tạo ra.

Logic

Trong chương này, ta sẽ xem xét logic với vai trò là phương tiện để biểu diễn tri thức và suy diễn.

Dạng biểu diễn tri thức cổ điển nhất trong máy tính là logic, với hai dạng phổ biến là logic mệnh đề và logic vị từ. Logic là một ngôn ngữ biểu diễn tri thức trong đó các câu nhận hai giá trị đúng (True) hoặc sai (False)[1] . Cũng như mọi ngôn ngữ biểu diễn tri thức, logic được xác định bởi 3 thành phần sau:

– *Cú pháp*: bao gồm các ký hiệu và các quy tắc liên kết các ký hiệu để tạo thành câu hay biểu thức logic. Một ví dụ cú pháp là các ký hiệu và quy tắc xây dựng biểu thức toán học trong số học và đại số.

– *Ngữ nghĩa* của ngôn ngữ cho phép ta xác định ý nghĩa của các câu trong một miền nào đó của thế giới hiện thực, xác định các sự kiện hoặc sự vật phản ánh thế giới thực của câu mệnh đề. Đối với logic, ngữ nghĩa cho phép xác định câu là đúng hay sai trong thế giới của bài toán đang xét. Ví dụ, trong ngôn ngữ toán học, câu a + 1 = 3 là câu đúng cú pháp. Theo ngữ nghĩa của ngôn ngữ toán học, câu này là đúng trong miền bài toán có a = 2 và sai trong những miền bài toán có a ≠ 2.

– *Cơ chế suy diễn* là phương pháp cho phép sinh ra các câu mới từ các câu đã có hoặc kiểm tra liệu các câu có phải là hệ quả logic của nhau. Ta có thể sử dụng suy diễn để sinh ra các tri thức mới từ tri thức đã có trong cơ sở tri thức.

Logic cung cấp một công cụ hình thức để biểu diễn và suy luận tri thức. Các phần tiếp theo sẽ trình bày về hai dạng logic mệnh đề và logic vị từ cũng như cách sử dụng các hệ thống logic này trong biểu diễn tri thức và suy diễn.

3.2. LOGIC MỆNH ĐỀ

3.2.1. Cú pháp

[1] Một số hệ thống logic được phát triển về sau sử dụng nhiều giá trị hơn như logic đa trị, logic mờ

[2] Lưu ý, đây là tiên đề dùng cho suy diễn xác suất, trong trường hợp tổng quát, ta có P(Ω)=1, P(∅)=0, trong đó Ω là toàn bộ không gian lấy mẫu.

Logic mệnh đề là logic rất đơn giản, tuy khả năng biểu diễn của nó còn một số hạn chế nhưng thuận tiện cho ta đưa vào nhiều khái niệm quan trọng trong logic.

Cú pháp của logic mệnh đề bao gồm tập các ký hiệu và tập các quy tắc kết hợp các ký hiệu tạo thành công thức hay câu.

a) Các ký hiệu

Các ký hiệu được dùng trong logic mệnh đề bao gồm:

- Các ký hiệu chân lý hay các hằng logic: True (ký hiệu T) và False (ký hiệu F).
- Các ký hiệu mệnh đề (còn được gọi là các biến mệnh đề và thường được ký hiệu bằng các chữ cái): P, Q,...
- Các kết nối logic $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$.
- Các dấu ngoặc, chẳng hạn "(" và ")".

b) Các câu hay công thức

Các câu hay công thức trong logic mệnh đề được xác định theo các quy tắc sau:

Quy tắc 1: Mọi ký hiệu chân lý và ký hiệu mệnh đề là câu.

Ví dụ: True, P

Các câu chỉ gồm ký hiệu chân lý hoặc ký hiệu mệnh đề như vậy gọi là các *câu đơn* hay *câu nguyên tử*.

Quy tắc 2: Thêm ngoặc ra ngoài một câu sẽ được một câu.

Ví dụ, nếu P là câu thì (P) cũng là câu.

Quy tắc 3: Kết hợp các câu bằng phép nối logic sẽ tạo ra câu mới. Cụ thể là:

Nếu A và B là câu thì:

$(A \wedge B)$ (đọc là "A hội B" hoặc "A và B")

$(A \vee B)$ (đọc là "A tuyển B" hoặc "A hoặc B")

$(\neg A)$ (đọc là "phủ định A")

$(A \Rightarrow B)$ (đọc là "A kéo theo B" hoặc "nếu A thì B"). Phép kéo theo còn được gọi là quy tắc "nếu – thì"

$(A \Leftrightarrow B)$ (đọc là "A và B kéo theo nhau" hay "A và B tương đương nhau", một số tài liệu sử dụng ký hiệu $\equiv$ cho phép nối logic này)

là các câu.

Các câu được tạo ra như vậy không phải câu đơn và được gọi là *câu phức hợp*.

Để cho ngắn gọn các công thức được bỏ đi các cặp dấu ngoặc không cần thiết. Chẳng hạn, thay cho $((A \vee B) \wedge C)$ ta sẽ viết là $(A \vee B) \wedge C$. Trong trường hợp một câu chứa nhiều phép nối, các phép nối sẽ được thực hiện theo thứ tự sau:

$\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

Nếu P là câu đơn thì P và $\neg$ P được gọi là *literal*, P là *literal dương*, còn $\neg$ P là *literal âm*. Câu phức hợp có dạng $A_1 \vee ... \vee A_m$ trong đó A_i là các literal sẽ được gọi là *câu tuyển* (clause).

3.2.2. Ngữ nghĩa

Ngữ nghĩa của logic mệnh đề cho phép xác định một câu (công thức) logic là đúng hay sai trong thế giới của bài toán đang xét, tức là cách diễn giải của các ký hiệu mệnh đề, ký hiệu chân lý và phép nối logic trong thế giới đó.

Trong logic mệnh đề, người sử dụng xác định giá trị đúng hay sai cho ký hiệu mệnh đề. Mỗi ký hiệu mệnh đề có thể tương ứng với một phát biểu (mệnh đề), ví dụ ký hiệu mệnh đề A có thể tương ứng với phát biểu: "Hà Nội là thủ đô của Việt Nam" hoặc bất kì một phát biểu nào khác. Một phát biểu chỉ có thể đúng (True) hoặc sai (False). Chẳng hạn, phát biểu "Hà Nội là thủ đô của Việt Nam" là đúng còn phát biểu "Voi là gia cầm" là sai (trong thế giới của chúng ta).

Một *minh họa* là một cách gán cho mỗi biến mệnh đề một giá trị chân lý True hoặc False. Nếu biến mệnh đề A được gán giá trị chân lý True/False (A←True/ A←False) thì ta nói mệnh đề A đúng/sai trong minh họa đó.

Ngữ nghĩa của logic cần quy định cách tính giá trị (đúng hoặc sai) cho các câu theo mô hình thế giới của bài toán. Do bất cứ câu nào cũng được tạo ra từ câu đơn và 5 kết nối logic, ta có thể tính giá trị các câu nếu xác định được giá trị câu nguyên tử và giá trị tạo ra bởi từng phép nối. Giá trị của câu đơn được xác định như đã nói ở trên, tức là người sử dụng sẽ quy định giá trị đúng hay sai cho từng mệnh đề. Hằng True luôn đúng, và hằng False luôn sai.

Tiếp theo, ta cần xác định giá trị câu được tạo ra bởi kết nối logic. Ý nghĩa các kết nối logic được cho bởi bảng chân lý, trong đó liệt kê giá trị của câu phức cho tất cả tổ hợp giá trị các thành phần của câu. Bảng chân lý cho năm kết nối logic được cho trong bảng sau:

Bảng 3.1. Bảng chân lý của các kết nối logic

A	B	¬ A	A∧B	A v B	A=>B	A<=>B
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Sử dụng bảng chân lý, ta có thể tính được giá trị bất cứ câu phức nào bằng cách thực hiện đệ quy những kết nối thành phần.

Các công thức tương đương

Các phép biến đổi tương đương giúp đưa các công thức về dạng thuận lợi cho việc lập luận và suy diễn. Hai công thức A và B được xem là tương đương nếu chúng có cùng một giá trị chân lý trong mọi minh họa.

Ký hiệu: Để chỉ A tương đương với B ta viết A ≡ B.

Bằng phương pháp bảng chân lý, dễ dàng chứng minh được sự tương đương của các công thức sau đây :

- $A \Rightarrow B \equiv \neg A \vee B$
- $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$

Luật phủ định kép

- $\neg (\neg A) \equiv A$

Luật De Morgan

- $\neg (A \vee B) \equiv \neg A \wedge \neg B$
- $\neg (A \wedge B) \equiv \neg A \vee \neg B$

Luật giao hoán

- $A \vee B \equiv B \vee A$
- $A \wedge B \equiv B \wedge A$

Luật kết hợp

- $(A \vee B) \vee C \equiv A \vee (B \vee C)$
- $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$

Luật phân phối

- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

3.3. SUY DIỄN VỚI LOGIC MỆNH ĐỀ

3.3.1. Suy diễn logic

Một công thức H được xem là hệ quả logic của một tập công thức G =$\{G_1,.....,G_m\}$ nếu trong bất kỳ minh họa nào mà $\{G_1,.....,G_m\}$ đúng thì H cũng đúng.

Khi có một cơ sở tri thức dưới dạng tập hợp các câu logic, ta muốn sử dụng các tri thức trong cơ sở này để sinh ra tri thức mới, tức là sinh ra là hệ quả logic của các công thức trong cơ sở tri thức. Điều đó được thực hiện bằng cách suy diễn. Suy diễn hay suy lý thường dùng chỉ quá trình cho phép rút ra kết luận. Để thực hiện suy diễn ta sử dụng luật suy diễn. Một luật suy diễn gồm hai phần : một tập các điều kiện và một kết luận.

Định nghĩa.

Thủ tục suy diễn được gọi là đúng đắn (sound) nếu kết quả suy diễn là hệ quả logic của điều kiện.

Thủ tục suy diễn được gọi là đầy đủ (complete) nếu cho phép tìm ra mọi hệ quả logic của điều kiện.

Ta sẽ sử dụng những kí hiệu sau:

KB: kí hiệu tập các câu đã có hay cơ sở tri thức (Knowledge Base)

KB $\models \alpha$: Khi các câu trong KB là đúng (True) thì α là đúng (True), hay α là hệ quả logic của KB.

Nếu một câu α được sinh ra từ KB nhờ thủ tục suy diễn (hay thuật toán suy diễn) *i* thì ta ký hiệu:

KB $\vdash_i \alpha$

3.3.2. Suy diễn sử dụng bảng chân lý

Suy diễn là xác định liệu một công thức α có phải là hệ quả logic của các công thức trong cơ sở tri thức KB không. Nói cách khác, cần xác định KB $\models \alpha$ đúng hay sai. Thủ tục suy diễn đơn giản nhất được thực hiện bằng cách liệt kê tất cả các minh họa và kiểm tra xem trong các minh họa mà các câu trong cơ sở tri thức KB đúng thì α có đúng không, theo đúng định nghĩa về hệ quả logic ở phần trước. Như ta đã biết ở trên, mỗi minh họa là một cách gán các giá trị True hoặc False cho từng ký hiệu mệnh đề. Sử dụng bảng chân lý, có thể xác định giá trị cho các câu trong KB và giá trị của α cho từng minh họa. Từ đây có thể xác định được một công thức có phải là hệ quả logic của các công thức trong cơ sở tri thức hay không.

Ví dụ: cho KB: $A \vee C$, $B \vee \neg C$

và $\alpha = A \vee B$

Để kiểm tra α có phải hệ quả logic của KB không, ta xây dựng bảng sau (bảng 3.2):

Kết quả xây dựng bảng cho thấy, α là hệ quả logic của KB, hay nói cách khác từ KB suy ra được α.

Bảng 3.2. Bảng chân lý

A	B	C	$A \vee C$	$B \vee \neg C$	$(A \vee C) \wedge (B \vee \neg C)$	$A \vee B$	KB $\models \alpha$
T	T	T	T	T	T	T	√
T	T	F	T	T	T	T	√
T	F	T	T	F	F	T	
T	F	F	T	T	T	T	√
F	T	T	T	T	T	T	√
F	T	F	F	T	F	T	
F	F	T	T	F	F	F	
F	F	F	F	T	F	F	

Suy diễn với logic mệnh đề sử dụng bảng chân lý là thủ tục suy diễn *đầy đủ* và *đúng đắn*. Tính đúng đắn là hiển nhiên do bảng chân lý sử dụng đúng ngữ nghĩa được quy định với kết nối logic. Tính đầy đủ là do số lượng các tổ hợp giá trị đối với logic mệnh đề là hữu hạn và do vậy có thể liệt kê đầy đủ trường hợp KB có giá trị đúng.

Tuy nhiên, cần lưu ý rằng, một công thức chứa n biến mệnh đề, thì số các minh họa của nó là 2^n , tức là bảng chân lý có 2^n dòng. Như vậy việc kiểm tra một công thức có phải là một hệ quả lôgic hay không bằng phương pháp bảng chân lý có độ phức tạp tính toán lớn do đòi hỏi thời gian theo hàm mũ. Cook (1971) đã chứng minh rằng, phương pháp chứng minh thuật suy diễn là bài toán NP-đầy đủ.

3.3.3. Sử dụng các quy tắc suy diễn

Do việc suy diễn sử dụng bảng như trên có độ phức tạp lớn nên cần có những thuật toán suy diễn hiệu quả hơn cho logic mệnh đề. Các thủ tục suy diễn đều dựa trên một số khái niệm như công thức tương được và các quy tắc suy diễn. Sau đây là một số luật suy diễn quan trọng trong logic mệnh đề.

Trong các luật này α, α_i , β, γ là các câu. Phần tiền đề hay phần điều kiện được viết dưới dạng tử số, phần hệ quả được viết dưới dạng mẫu số.

1. Luật Modus Ponens

$$\frac{\alpha => \beta, \alpha}{\beta}$$

Từ một kéo theo và giả thiết của kéo theo, ta suy ra kết luận của nó.

2. Luật Modus Tollens

$$\frac{\alpha => \beta, \neg\ \beta}{\neg\ \alpha}$$

Từ một kéo theo và phủ định kết luận của nó, ta suy ra phủ định giả thiết của kéo theo.

3.Luật loại trừ và

$$\frac{\alpha_1 \wedge \ldots\ldots \wedge \alpha_i \wedge \ldots\ldots \wedge \alpha_m}{\alpha_i}$$

Từ một công thức và ta đưa ra một nhân tử bất kỳ của công thức đó .

4. Luật nhập đề và

$$\frac{\alpha_1, \ldots\ldots, \alpha_i, \ldots\ldots \alpha_m}{\alpha_1 \wedge \ldots\ldots \wedge \alpha_i \wedge \ldots\ldots \wedge \alpha_m}$$

Từ một danh sách các công thức, ta suy ra phép và của chúng.

5. Luật nhập đề hoặc

$$\frac{\alpha_i}{\alpha_1 v \ldots\ldots v \alpha_i . v \ldots\ldots v \alpha_m}$$

Từ một công thức, ta suy ra một phép hoặc mà một trong các hạng tử của phép hoặc là công thức đó.

6. Luật loại trừ phủ định kép

$$\frac{\neg(\neg \alpha)}{\alpha}$$

Phép phủ định của phủ định một công thức, ta suy ra chính công thức đó.

7. Luật bắc cầu

$$\frac{\alpha => \beta, \beta => \gamma}{\alpha => \gamma}$$

Từ hai kéo theo, mà kết luận của nó là của kéo theo thứ nhất trùng với giả thiết của kéo theo thứ hai, ta suy ra kéo theo mới mà giả thiết của nó là giả thiết của kéo theo thứ nhất, còn kết luận của nó là kết luận của kéo theo thứ hai.

8. Phép giải đơn vị

$$\frac{\alpha \vee \beta, \neg \beta}{\alpha}$$

Từ một phép hoặc, một hạng tử đối lập với một hạng tử trong tuyển kia, ta suy ra hạng tử còn lại.

9. Phép giải

$$\frac{\alpha \vee \beta, \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

Từ hai phép hoặc, một phép hoặc chứa một hạng tử đối lập với một hạng tử trong phép hoặc kia, ta suy ra phép hoặc của các hạng tử còn lại.

Một luật suy diễn được xem là tin cậy nếu bất kỳ một mô hình nào của giả thiết của luật cũng là mô hình kết luận của luật. Chúng ta chỉ quan tâm đến các luật suy diễn tin cậy.

Với các quy tắc suy diễn vừa trình bày, việc suy diễn trên logic mệnh đề được thực hiện nhờ một số thủ tục nhất định, trong đó thông dụng nhất là suy diễn tiến (forward chaining), suy diễn lùi (backward chaining), suy diễn bằng phép giải (resolution) và phản chứng (refutation). Tuy nhiên trong phần này chúng ta không đi sâu vào các thủ tục này mà sẽ xem xét các thủ tục chứng minh trong phần trình bày về logic vị từ.

3.4. LOGIC VỊ TỪ (LOGIC BẬC 1)

Trong phần trước ta đã xem xét logic mệnh đề và cách sử dụng logic mệnh để biểu diễn tri thức. Bên cạnh ưu điểm là đơn giản, logic mệnh đề có một nhược điểm lớn là khả năng biểu đạt hạn chế, không thể sử dụng để biểu diễn tri thức một cách ngắn gọn cho những bài toán có độ phức tạp lớn. Cụ thể là logic mệnh để thuận lợi cho biểu diễn sự kiện, sự kiện đơn giản được biểu diễn bằng câu nguyên tử, sự kiện phức tạp được biểu diễn bằng cách sử dụng kết nối logic để kết hợp câu nguyên tử. Logic mệnh đề không cho phép biểu diễn một cách ngắn gọn môi trường với nhiều đối tượng. Chẳng hạn để thể hiện nhận xét "tất cả sinh viên trong lớp nào đó chăm học" ta phải sử dụng các câu riêng rẽ để thể hiện từng sinh viên cụ thể trong lớp chăm học. Nói chung, logic mệnh đề không cho phép biểu diễn ngắn gọn các yếu tố về thời gian, không gian, số lượng hoặc các quan hệ có tính phổ quát giữa các đối tượng.

Trong phần này ta sẽ xem xét logic vị từ - một hệ thống logic có khả năng biểu diễn ngắn gọn và mạnh hơn, đồng thời xem xét chi tiết thủ tục suy diễn với logic vị từ.

3.4.1. Đặc điểm

Đặc điểm quan trọng nhất của logic vị từ là cho phép biểu diễn thế giới xung quanh dưới dạng các đối tượng, tính chất đối tượng, và quan hệ giữa các đối tượng đó. Việc sử dụng đối tượng là rất tự nhiên trong thế giới thực và trong ngôn ngữ tự nhiên, với danh từ biểu diễn đối tượng, tính từ biểu diễn tính chất và động từ biểu diễn quan hệ giữa các đối tượng. Có thể kể ra rất nhiều ví dụ về đối tượng, tính chất và quan hệ:

- Đối tượng : một cái bàn, một cái nhà, một cái cây, một con người, một sinh viên, một con số. ...
- Tính chất : Cái bàn có thể có tính chất : có bốn chân, làm bằng gỗ, không có ngăn kéo, sinh viên có thể có tính chất là thông minh, cao, gầy…
- Quan hệ : cha con, anh em, bè bạn (giữa con người); lớn hơn nhỏ hơn, bằng nhau (giữa các con số) ; bên trong, bên ngoài nằm trên nằm dưới (giữa các đồ vật)...
- Hàm : Một trường hợp riêng của quan hệ là quan hệ hàm, trong đó với mỗi đầu vào là một hoặc nhiều đối tượng, ta có một giá trị hàm duy nhất, cũng là một đối tượng. Ví dụ: tay trái của ai đó, bố của ai đó, bội số chung nhỏ nhất của hai số.

Logic vị từ có cú pháp và ngữ nghĩa được xây dựng dựa trên khái niệm đối tượng. Hệ thống logic này đóng vai trò quan trọng trong việc biểu diễn tri thức do có khả năng biểu diễn phong phú và tự nhiên, đồng thời là cơ sở cho nhiều hệ thống logic khác.

3.4.2. Cú pháp và ngữ nghĩa

Trong phần này ta sẽ xem xét cú pháp, tức là quy tắc tạo ra những câu hay biểu thức logic, của logic vị từ cùng với ngữ nghĩa của những cấu trúc đó.

Các ký hiệu và ý nghĩa

Logic vị từ sử dụng những dạng ký hiệu sau.

- Ký hiệu hằng logic: True, False

- Các ký hiệu hằng: Nam, 3, Vịnh Hạ long,… dùng để thể hiện các đối tượng
- Các ký hiệu biến: x, y, z,... biểu diễn lớp đối tượng
- Các ký hiệu vị từ: Thích (Nam, Bắc), Làm_từ_gỗ (tủ), Anh_em (An, Ba, Út)

  Ký hiệu vị từ thể hiện quan hệ giữa các đối tượng hoặc tính chất của đối tượng. Mỗi vị từ có thể có n tham số (n≥0). Ví dụ Thích là vị từ của hai tham số, Làm_từ_gỗ là vị từ một tham số. Các ký hiệu vị từ không tham số là các ký hiệu mệnh đề.
- Các ký hiệu hàm: Mẹ_của(An), min(3,4,9),...

  Ký hiệu hàm thể hiện quan hệ hàm. Mỗi hàm có thể có n tham số (n≥1). Mặc dù cú pháp của hàm tương tự vị từ nhưng hàm trả về giá trị là đối tượng, trong khi vị từ có giá trị là True hay False.
- Các ký hiệu kết nối logic: ∧ (hội), ∨ (tuyển), ¬ (phủ định), ⇒ (kéo theo), ⇔ (tương đương).
- Các ký hiệu lượng tử: ∀ (với mọi), ∃ (tồn tại).
- Các ký hiệu ngăn cách: dấu phẩy, dấu mở ngoặc và dấu đóng ngoặc.

Tương tự như với logic mệnh đề, ngữ nghĩa cho phép liên kết biểu thức logic với thế giới của bài toán để xác định tính đúng hoặc sai của biểu thức. Một liên kết cụ thể như vậy được gọi là một *minh họa*. Minh họa xác định cụ thể đối tượng, quan hệ và hàm mà các ký hiệu hằng, vị từ, và ký hiệu hàm thể hiện.

Để xác định một minh họa, trước hết ta cần xác định một *miền đối tượng* (nó bao gồm tất cả các đối tượng trong thế giới mà ta quan tâm). Cũng có thể xác định miền đối tượng cho từng tham số của một vị từ hoặc một hàm nào đó. Ví dụ trong vị từ Thích(x,y), miền của x là tất cả mọi người, miền của y là các loại động vật. Số đối tượng có thể là vô hạn, chẳng hạn trong trường hợp miền đối tượng là toàn bộ số thực.

Việc lựa chọn tên cho hằng, biến, vị từ, và hàm hoàn toàn do người dùng quyết định. Có thể có nhiều minh họa khác nhau cho cùng một thế giới thực. Tương tự như với logic mệnh đề, việc suy diễn, tính đúng đắn của biểu thức, hay việc xác định hệ quả logic được xác định dựa trên toàn bộ minh họa. Tuy nhiên, việc liệt kê toàn bộ minh họa trong logic vị từ phức tạp hơn nhiều so với logic mệnh đề, thậm chí không thể thực hiện được, so số lượng minh họa có thể là vô hạn.

Hạng thức (term)

Hạng thức (term) là biểu thức logic có kết quả là đối tượng. Hạng thức được xác định đệ quy như sau.

- Các ký hiệu hằng và các ký hiệu biến là hạng thức.
- Nếu t_1, t_2, t_3, ..., t_n là n hạng thức và f là một ký hiệu hàm n tham số thì $f(t_1, t_2, ..., t_n)$ là hạng thức. Một hạng thức không chứa biến được gọi là một hạng thức cụ thể hay hạng thức nền (ground term).

Chẳng hạn, An là ký hiệu hằng, Mẹ_của là ký hiệu hàm, thì Mẹ_của(An) là một hạng thức cụ thể. Nhờ sử dụng ký hiệu hàm, ta không cần đặt tên cho tất cả các đối tượng. Chẳng hạn, thay vì dùng một hằng cụ thể để biểu diện mẹ của An, ta có thể dụng ký hiệu hàm Mẹ_của (An).

Ngữ nghĩa của hạng thức như sau: các hằng, biến, tham số tương ứng với đối tượng trong miền đối tượng; ký hiệu hàm tương ứng với quan hệ hàm trong thế giới thực; hạng thức tương ứng với đối tượng là giá trị của hàm khi nhận tham số.

Ký hiệu "="

Hai hạng thức bằng nhau và được ký hiệu "=" nếu cùng tương ứng với một đối tượng.

Ví dụ: Mẹ_của(Vua_Tự_Đức) = Bà_Từ_Dũ

Tính đúng đắn của quan hệ bằng được xác định bằng cách kiểm tra hai vế của ký tự "=".

Câu nguyên tử (câu đơn)

Các *câu nguyên tử*, còn gọi là *câu đơn*, được xác định như sau:

- True và False là các câu nguyên tử.
- Vị từ có tham số là hạng thức là câu nguyên tử.
- Hạng thức 1 = hạng thức 2 là câu nguyên tử.

Ví dụ : Yêu (Hoa, Mẹ_của(Hoa))

Mẹ_của(Vua_Tự_Đức) = Bà_Từ_Dũ

Câu nguyên tử nhận giá trị đúng (true) trong một minh họa nào đó nếu quan hệ được biểu diễn bởi ký hiệu vị từ là đúng đối với các đối tượng được biểu diễn bởi các hạng thức đóng vai trò thông số. Như vậy, câu nguyên tử thể hiện những sự kiện (đơn giản) trong thế giới của bài toán và do vậy tương đương với các mệnh đề.

Một câu nguyên tử là đúng nếu trong một minh họa nào đó nếu quan hệ được biểu diễn bởi vị tự của câu tồn tại giữa các tham số của vị từ trong minh họa đó. Nhắc lại: minh họa là cách gán giá trị cụ thể cho các ký hiệu và biến trong thế giới nào đó.

Câu

Từ các câu nguyên tử, sử dụng các kết nối logic và các lượng tử, ta xây dựng nên các câu. Câu được định nghĩa đệ quy như sau:

- Câu nguyên tử là câu.
- Nếu G và H là các câu nguyên tử, thì các biểu thức (G ∧ H), (G ∨ H), (¬ G), (G⟹ H), (G⟺H) là câu
- Nếu G là một câu nguyên tử và x là biến thì các biểu thức (∀ x G), (∃ x G) là câu, trong đó ∀, ∃ là các lượng tử logic sẽ được đề cập tới trong phần sau.

Các câu không phải là câu nguyên tử sẽ được gọi là các *câu phức hợp*. Các câu không chứa biến được gọi là câu cụ thể. Khi viết các công thức ta sẽ bỏ đi các dấu ngoặc không cần thiết, chẳng hạn các dấu ngoặc ngoài cùng.

Ví dụ: ¬ Ghét (Hoa, Mẹ_của (Hoa))

Anh_em (Nam, Dũng) ∧ Anh_em (Dũng, Nam)

Thuận_tay_trái (x) ∨ Thuận_tay_phải (x)

Anh_của(x, y) → ¬ Anh_của (y, x)

Ngữ nghĩa của câu phức hợp được xác định một cách đệ quy từ ngữ nghĩa các câu đơn và các phép nối logic tương tự như trong logic mệnh đề. Cụ thể là, nếu P, Q là các câu thì:

¬ P là phủ định của P và nhận giá trị true trong một minh họa nếu P sai trong minh họa đó và ngược lại.

P ∧ Q nhận giá trị true nếu cả P và Q đều đúng và nhận giá trị false nếu ít nhất một trong hai câu P, Q là sai.

P ∨ Q nhận giá trị true nếu ít nhất một trong hai câu P, Q đúng và nhận giá trị false nếu cả hai câu đều sai.

P → Q nhận giá trị false nếu P đúng và Q sai, nhận giá trị true trong các trường hợp còn lại.

P ↔ Q nhận giá trị true nếu các P và Q cùng đúng hoặc cả P và Q cùng sai, nhận giá trị false trong các trường hợp còn lại.

∀ x P nhận giá trị true nếu tất cả các câu nhận được từ P bằng cách thay x bởi một đối tượng trong miền giá trị của x đều có giá trị đúng, và nhận giá trị false nếu ít nhất một câu như vậy sai.

∃ x P nhận giá trị true nếu tồn tại một đối tượng nào đó trong miền giá trị của biến x làm cho câu P nhận giá trị true.

Trừ ngữ nghĩa của các câu có chứa lượng tử, ngữ nghĩa của các phép nối tương tự như trong logic mệnh đề và có thể thể hiện bằng bảng chân lý.

Các lượng tử

Logic mệnh đề sử dụng hai lượng tử: với mọi và tồn tại.

Lượng tử với mọi (ký hiệu ∀) cho phép mô tả tính chất của cả một lớp các đối tượng, chứ không phải của một đối tượng, mà không cần phải liệt kê ra tất cả các đối tượng trong lớp. Ví dụ ta sử dụng vị từ Voi(x) (đối tượng x là con voi) và vị từ Xám(x) (đối tượng x có màu xám) thì câu " tất cả các con voi đều có màu xám" có thể biểu diễn bởi công thức

∀x (Voi (x) ⟹ Xám(x)).

Như vậy câu ∀ x P có nghĩa là câu P đúng với mọi đối tượng x thuộc miền giá trị đã được quy định của thế giới bài toán. Lượng tử với mọi có thể coi như phép *hội của nhiều câu*.

Lưu ý: lượng tử với mọi được dùng với kéo theo chứ không dùng với "và". Chẳng hạn, để nói rằng mọi sinh viên đều chăm học thì câu

$\forall x$ Sinh_viên(x) $\Rightarrow$ Chăm_học(x) là đúng

trong khi

$\forall x$ Sinh_viên(x) $\wedge$ Chăm_học(x) là sai do câu này sẽ có ý nghĩa tất cả mọi người đều là sinh viên và đều chăm học.

Lượng tử tồn tại (ký hiệu $\exists$) cho phép ta tạo ra các câu nói đến một đối tượng nào đó trong một lớp đối tượng mà nó có một tính chất hoặc thoả mãn một quan hệ nào đó. Ví dụ ta sử dụng các câu nguyên tử Sinh_viên(x) (x là sinh viên) và Ở_trong(x, P308), (x ở trong phòng 308), ta có thể biểu diễn câu " Có một sinh viên ở phòng 308" bởi biểu thức

$\exists x$ (Sinh_viên(x) $\wedge$ Ở_trong (x, P308).

Ngữ nghĩa của công thức $\exists x$ P được xác định như là ngữ nghĩa của công thức là tuyển của tất cả các công thức nhận được từ P bằng cách thay x bởi một đối tượng trong miền đối tượng.

Lưu ý: Lượng tử tồn tại được dùng với "và" chứ không dùng với "kéo theo". Chẳng hạn để nói rằng có một số sinh viên chăm học thì câu:

$\exists x$ Sinh_viên(x) $\wedge$ Chăm_học(x) là đúng

trong khi

$\exists x$ Sinh_viên(x) $\Rightarrow$ Chăm_học(x)

là sai. Thật vậy, do phép kéo theo đúng khi tiền đề là sai nên câu trên đúng khi có một người x nào đó không phải là sinh viên, trong khi đây không phải là ý mà ta muốn khẳng định.

Quan hệ giữa lượng tử với mọi và lượng tử tồn tại: lượng tử này có thể biểu diễn bằng lượng tử kia bằng cách sử dụng phép phủ định. Ví dụ:

$\forall x$ Thích (x, Kem) tương đương với $\neg \exists x \neg$ Thích(x, Kem)

$\exists y$ Thích (x, Kem) tương đương với $\neg \forall x \neg$ Thích (x, Kem)

Như vậy, ta có thể dùng một trong hai lượng tử để biểu diễn cho lượng tử còn lại. Tuy nhiên để thuận tiện cho việc đọc và hiểu các câu logic, logic vị từ vẫn sử dụng cả hai lượng tử với mọi và tồn tại.

Các lượng tử lồng nhau

Có thể sử dụng đồng thời nhiều lượng tử trong một câu phức tạp. Vùng ảnh hưởng của lượng tử có thể bao hàm lượng tử khác và khi đó ta nói lượng tử lồng nhau. Ví dụ:

$\forall x \forall y$ Anh_em(x, y) $\Rightarrow$ Họ_hàng(x, y)

$\forall x \exists y$ Yêu (x, y)

Nhiều lượng tử cùng loại có thể được viết gọn bằng một ký hiệu lượng tử, ví dụ câu thứ nhất có thể viết gọn thành

$\forall x, y$ Anh_em(x, y) $\Rightarrow$ Họ_hàng(x, y)

Trong trường hợp lượng tử với mọi được sử dụng cùng lượng tử tồn tại thì thứ tự lượng tử ảnh hưởng tới ngữ nghĩa của câu và không được phép thay đổi. Chẳng hạn câu

$\forall x \exists y$ Yêu (x, y)

có nghĩa là mọi người đều có ai đấy để yêu, trong khi câu

$\exists y \forall x$ Yêu (x, y)

có nghĩa là có ai đó mà tất cả đều yêu.

Trong trường hợp nhiều lượng tử khác nhau cùng sử dụng một tên biến thì có thể gây nhầm lẫn vì vậy cần sử dụng tên biến khác nhau cho ký hiệu lượng tử khác nhau.

Literal

Một câu là câu nguyên tử hoặc là phủ định của câu nguyên tử được gọi là literal.

Ví dụ: Chơi (x, bóng_đá),

¬ Thích (Lan, hoa_hồng)

là các literal, trong đó câu thứ nhất là literal dương và câu thứ hai là literal âm.

Câu tuyển (clause)

Một công thức là tuyển của các literal sẽ được gọi là câu tuyển. Chẳng hạn:

Đàn_ông (x) $\vee$ ¬ Thích (x, bóng_đá) là câu tuyển.

Các công thức tương đương

Cũng như trong logic mệnh đề, ta nói hai công thức G và H tương đương (viết là G $\equiv$ H) nếu chúng cùng đúng hoặc cùng sai trong một minh hoạ. Ngoài các tương đương đã biết trong logic mệnh đề, trong logic vị từ cấp một còn có các tương đương khác liên quan tới các lượng tử.

Sau đây là các tương đương của logic vị từ

$\forall x$ G(x) $\equiv \forall y$ G(y)

$\exists x$ G(x) $\equiv \exists y$ G(y)

Đặt tên lại biến đi sau lượng tử tồn tại, ta nhận được công thức tương đương .

¬ ($\forall x$ G(x)) $\equiv \exists x$ (¬ G(x))

¬ ($\exists x$ G(x)) $\equiv \forall x$ (¬ G(x))

$\forall x$ (G(x) $\wedge$ H(x)) $\equiv \forall x$ G(x) $\wedge \forall x$ H(x)

$\exists x$ (G(x) $\vee$ H(x)) $\equiv$ $\exists x$ G(x) $\vee \exists x$ H(x)

Ví dụ : $\forall x$ Yêu(x, Mẹ_của(x)) $\equiv \forall y$ Yêu(y, Mẹ_của(y)).

Một số ví dụ

Dưới đây là một số ví dụ các câu tiếng Việt và biểu diễn các câu đó bằng logic vị từ. Cần lưu ý rằng mỗi câu tiếng Việt có thể có nhiều câu tương đương trong logic vị từ. Trong mỗi ví dụ dưới đây chỉ sử dụng một câu như vậy.

Tất cả sinh viên đều chăm học

$\forall x$ (Sinh_viên(x) $\Rightarrow$ Chăm_học(x))

Có một số sinh viên

$\exists x$ Sinh_viên(x)

Một số sinh viên chăm học

$\exists x$ (Sinh_viên(x) $\wedge$ Chăm_học(x))

Mỗi sinh viên đều thích một sinh viên nào đó

$\forall x$ (Sinh_viên (x) $\Rightarrow \exists y$ (Sinh_viên (y) $\wedge$ Thích (x, y)))

Mỗi sinh viên đều thích một sinh viên khác

$\forall x$ (Sinh_viên(x) $\Rightarrow \exists y$ (Sinh_viên(y) $\wedge \neg$ (x = y) $\wedge$ Thích(x,y)))

Có một sinh viên được tất cả sinh viên khác thích

$\exists x$ (Sinh_viên(x) $\wedge \forall y$ (Sinh_viên(y) $\wedge \neg$(x = y) $\Rightarrow$ Thích(y,x)))

Nam là sinh viên

Sinh_viên (Nam)

Nam học một trong hai ngành: công nghệ thông tin hoặc kế toán

Học_ngành (Nam, CNTT) $\Leftrightarrow \neg$ Học_ngành (Nam, kế_toán)

Nam hoặc CNTT hoặc kế toán hoặc cả hai ngành một lúc

Học_ngành (Nam, CNTT) $\vee$ Học_ngành (Nam, kế_toán)

Tất cả sinh viên đều học ít nhất một ngành

$\forall x$ (Sinh_viên(x) $\Rightarrow \exists y$ (Ngành(y) $\wedge$ Học_ngành(x,y)))

Chỉ một sinh viên trượt môn Triết

$\exists x$ (Sinh_viên(x) $\wedge$ Trượt(x, Triết) $\wedge \forall y$ (Sinh_viên(y) $\wedge$ Trượt(y, Triết) $\Rightarrow$ x = y)

Không sinh viên nào giúp đỡ tất cả sinh viên khác

$\neg \exists x$ (Sinh_viên(x) $\wedge \forall y$ (Sinh_viên(y) $\wedge \neg$ (x = y) $\Rightarrow$ Giúp_đỡ(x,y)))

3.5. SUY DIỄN VỚI LOGIC VỊ TỪ

Trong phần về logic mệnh đề, ta đã xem xét một số quy tắc suy diễn. Quy tắc suy diễn là những thủ tục suy diễn đúng đắn và đơn giản. Trong phần này sẽ giới thiệu thêm các quy tắc suy diễn dùng cho các lượng tử. Cùng với các quy tắc suy diễn đã biết, các quy tắc suy diễn với lượng tử tạo thành tập quy tắc được sử dụng trong suy diễn với logic vị từ. Phần này cũng giới thiệu phép *hợp nhất* và cách sử dụng hợp nhất để suy diễn với các biểu thức logic vị từ. Sau đó, ta sẽ xem xét ba phương pháp suy diễn: *suy diễn tiến*, *suy diễn lùi*, và *suy diễn bằng hợp giải*.

3.5.1. Quy tắc suy diễn

Mọi quy tắc suy diễn cho logic mệnh đề cũng đúng với logic vị từ. Ngoài ra, logic vị từ còn có thêm một số quy tắc suy diễn khác, chủ yếu được dùng với câu có chứa lượng tử, cho phép biến đổi những câu này thành câu không có lượng tử.

Phép thế (substitution)

Trước khi đi xem xét quy tắc suy diên, ta định nghĩa khái niệm phép thế, cần thiết cho những câu có chứa biến.

Một *phép thế* θ là một danh sách các đôi $\theta = \{v_1 / t_1,..,v_n / t_n\}$, trong đó mỗi đôi v_i / t_i bao gồm biến v_i và hạng thức t_i được sử dụng để thay thế biến v_i.

Ví dụ: $\theta = \{x / y, .., z / \text{Mẹ\_của(An)}\}$

có nghĩa là x được thay bằng y và z được thay bằng Mẹ_của(An).

Với câu α và phép thế θ cho các biến của α, ký hiệu

SUBST(θ, α)

xác định một câu mới được tạo ra bằng cách thực hiện các phép thế được quy định trong θ vào câu α.

Ví dụ: SUBST ({x/Nam, y/An}, Thích(x,y))

có nghĩa là thay thế x bằng Nam và y bằng An trong câu Thích(x,y)), và cho kết quả như sau:

SUBST ({x/Nam, y/An}, Thích(x,y)) = Thích(Nam, An)

Với ký hiệu phép thế, ta có thể viết các quy tắc suy diễn như sau.

Phép loại trừ với mọi (universal elimination)

$$\frac{\forall x\ \alpha}{\text{SUBST}(\{x/g\}, \alpha)}$$

Tức là có thể thay thế biến x bằng một hằng số g và bỏ lượng tử ∀ để được câu mới không chứa lượng tử và không chứa biến của lượng tử.

Ví dụ:

$\forall x$ Thích(x, Kem) $\xrightarrow{\{x/Nam\}}$ Thích (Nam, Kem)

Lưu ý rằng loại trừ với mọi có thể thực hiện nhiều lần để tạo ra nhiều câu khác nhau bằng cách thay thế biến x bằng các hằng khác nhau.

Loại trừ tồn tại (existential elimination)

$$\frac{\exists x\ \alpha \quad \text{, k là kí hiệu hằng chưa xuất hiện trong KB}}{\text{SUBST}(\{x/k\}, \alpha)}$$

Tức là có thể thay thế biến x của lượng tử ∃ bằng một hằng số nào đó chưa xuất hiện trong KB để được câu mới không chứa lượng tử ∃.

Ví dụ:

$\exists x$ Học_giỏi(x) $\xrightarrow{\{x/Siêu\_nhân\}}$ Học_giỏi (Siêu_nhân),

trong đó Nam phải là một hằng chưa từng xuất hiện trong cơ sở tri thức.

Lý do phải chọn một hằng chưa xuất hiện trong KB là do câu ∃x Học_giỏi(x) chỉ có nghĩa là trong lớp có người học giỏi và không xác định rõ người đó, do vậy ta không được phép chọn tên của một sinh viên trong lớp mà phải đặt cho người học giỏi một các tên nào đó, chẳng hạn gọi người đó là "siêu nhân". Trong logic, một hằng số mới k như vậy được gọi là *hằng Skolem* và ta có thể đặt tên cho hằng này. Yêu cầu với hằng Skolem là hàm này chưa được phép xuất hiện trong cơ sở tri thức.

Nhập đề tồn tại (existential introduction)

Với câu α, biến x không thuộc câu α và hạng thức cơ sở g thuộc câu α

Ta có:

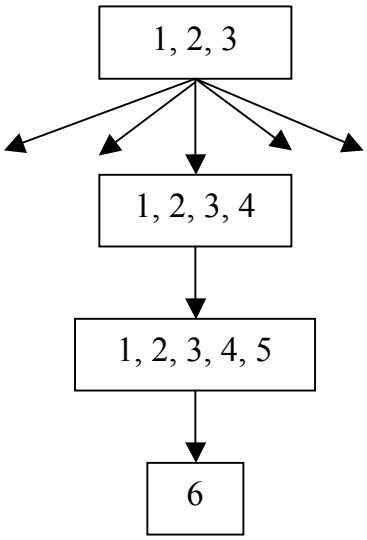
$$\frac{\alpha}{\exists x\ SUBST(\{g/x\}, \alpha)}$$

Ví dụ: Thích(Nam, Kem) $\xrightarrow{\{Nam/x\}}$ ∃x Thích (x, Kem)

Với các quy tắc suy diễn ở trên, ta có thể thực hiện suy diễn như trong ví dụ dưới đây.

Ví dụ suy diễn:

KB: Bob là trâu Pat là lợn Trâu to hơn lợn	KB: trâu (Bob) (1) lợn (Pat) (2) $\forall x, y$ trâu (x) ∧ lợn (y) => to_hơn (x,y) (3)
Câu hỏi: Bob to hơn Pat ?	 to_hơn (Bob, Pat) ?
Nhập đề và: (1) (2)	trâu (Bob) ∧ lợn(Pat) (4)
Loại trừ với mọi: (3)	trâu(Bob) ∧ lợn (Pat) => to_hơn (Bob,Pat) (5)
Modus Ponens: (4) (5)	to_hơn (Bob, Pat) (6) vậy to_hơn (Bob, Pat) là hệ quả logic của KB



Quá trình áp dụng các luật suy diễn để chứng minh câu truy vấn có thể coi như quá trình tìm kiếm, trong đó ta cần tìm cách áp dụng các luật suy diễn, tìm cách thay giá trị các biến trong loại trừ với mọi để dẫn tới câu truy vấn. Hình vẽ trên minh hoạ cho một phần của cây tìm kiếm của ví dụ trên.

Thực chất, trong ví dụ này ta đã sử dụng các quy tắc suy diễn với lượng tử để biến đổi các câu vị từ thành các câu trong logic mệnh đề, sau đó áp dụng các quy tắc suy diễn như với logic mệnh đề. Tuy nhiên, suy diễn tự động trên logic vị từ khó hơn so với suy diễn trên logic mệnh đề do các biến có thể nhận vô số các giá trị khác nhau. Ta cũng không thể sử dụng bảng chân lý do kích thước của bảng có thể là vô hạn. Trong các phần dưới đây sẽ trình bầy một số thủ tục suy diễn cho phép áp dụng trực tiếp vào các biểu thức logic vị từ.

Phép hợp nhất (Unification)

Câu trong logic vị từ có thể chứa các biến. Khi thực hiện suy diễn, thường xuất hiện yêu cầu thay thế các biến (thực hiện các phép thế) sao cho các câu trở nên giống nhau. Ví dụ, với hai câu Sinh_viên (x) và Sinh_viên(Nam), nếu thay thế {x/Nam} ta sẽ được hai câu giống nhau.

Giả sử p và q là hai câu, *hợp nhất là thủ tục xác định phép thế cần thiết để làm cho 2 câu giống nhau và được ký hiệu như sau*:

$$\text{UNIFY}(p, q) = \theta,$$

Sao cho $$\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

trong đó θ được gọi là *hợp tử* (phần tử hợp nhất - unifier)

Ví dụ: trong bảng sau cho một số cặp câu và hợp tử của các cặp câu đó.

p	q	θ
Biết (Nam, x)	Biết (Nam, Bắc)	{x/Bắc}
Biết (Nam, x)	Biết (y, Mẹ (y))	{y/Nam, x/ Mẹ (Nam)}

Biết (Nam, x)	Biết (y, z)	{y/Nam, x/z} {y/Nam, x/Nam, z/Nam}

Trong ví dụ thứ ba ở trên, tồn tại nhiều hơn một phép thế làm cho hai câu *p* và *q* hợp nhất với nhau. Trong đó, phép thế {y/Nam, x/z} đòi hỏi thay thế ít biến hơn so với phép thế {y/Nam, x/Nam, z/Nam}. Ta nói rằng phép thế {y/Nam, x/z} tổng quát hơn phép thế {y/Nam, x/Nam, z/Nam} do tạo ra ít ràng buộc hơn với giá trị các biến.

Trong các trường hợp tồn tại nhiều phép thế như vậy, ta sử dụng *hợp tử tổng quát nhất* (MGU: most general unifier) tức là hợp tử sử dụng ít phép thế cho biến nhất.

Phép hợp nhất có thể thực hiện tự động bằng thuật toán có độ phức tạp tỉ lệ tuyến tính với số lượng biến. Chi tiết thuật toán không được trình bầy ở đây, nhưng việc tồn tại thuật toán như vậy rất quan trọng khi xây dựng các thủ tục suy diễn có độ phức tạp tính toán thấp.

Modus Ponens tổng quát (GMP)

Giả sử ta có các câu cơ sở, p_i, p_i', q và tồn tại phép thế θ sao cho UNIFY $(p_i, p_i') = \theta$ với mọi i

Khi đó ta có:

$$\frac{p_1', p_2', p_3', \ldots p_n', \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{SUBST(\theta, q)}$$

Ví dụ: từ các câu

Sinh_viên (Nam)

Chăm_học (x)

(Sinh_viên(y) ∧ Chăm_học(y) ⇒Học_giỏi(y))

sử dụng GMP cho phép suy ra:

Học_giỏi (Nam)

Có thể coi quy tắc Modus Ponens tổng quát là sự kết hợp của modus ponens với các phép nhập đề và. Ngoài ra, GMP là phương án mở rộng của modus ponens thông thường, cho phép làm việc trực tiếp với các câu trong logic vị từ.

Thủ tục suy diễn với GMP là đúng đắn nhưng không đầy đủ với logic vị từ nói chung. Chi tiết về tính đúng đắn và đầy đủ của suy diễn sử dụng GMP sẽ được trình bầy trong phần về suy diễn tiến và suy diễn lùi.

Suy diễn bằng GMP chỉ đầy đủ trong trường hợp KB chỉ chứa các câu tuyển Horn (Horn clause), sẽ được định nghĩa dưới đây.

Các câu tuyển Horn (Horn clause)

Các câu tuyển Horn (Horn clause) đóng vai trò quan trọng trong một số phương pháp suy diễn. Các câu này được đặt theo tên của Alfred Horn, người đã chỉ ra vai trò của các câu dạng này trong suy diễn logic. Dưới đây là định nghĩa và ví dụ câu Horn.

Như đã nói ở trên, literal là câu nguyên tử hoặc phủ định của câu nguyên tử. *Clause* (dịch là *câu tuyển* hoặc *mệnh đề tuyển*) là tuyển của các literal. Dưới đây là ví dụ một clause:

$$\forall x, y\ Cao\_hơn(x, y) \vee Cao\_hơn(y, x)$$

Thông thường, ta sẽ viết câu trên mà không có lượng tử, tức là

$$Cao\_hơn(x, y) \vee Cao\_hơn(y, x)$$

do các biến tự đo được hiểu như lượng tử với mọi.

Câu tuyển Horn (Horn clause) là câu tuyển có tối đa một literal dương. Dưới đây là một số ví dụ câu Horn:

$$Cao\_hơn\ (An, Nam)$$

$$Cao\_hơn\ (An, Bố\_của\ (An))$$

$$\neg Cao\_hơn(x, y) \vee \neg Cao\_hơn(y, z) \vee Cao\_hơn(x, z)$$

$$\neg Cao\_hơn\ (An, Nam) \vee \neg Anh\_của\ (Nam, An)$$

Bên cạnh cách thể hiện dưới dạng tuyển của các literal, câu Horn còn được thể hiện bằng cách sử dụng phép kéo theo. Khi đó câu Horn có dạng

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$$

Chẳng hạn, ví dụ thứ ba và thứ tư trên sẽ được viết thành:

$$Cao\_hơn(x, y) \wedge Cao\_hơn(y, z) \Rightarrow Cao\_hơn(x, z)$$

$$Cao\_hơn\ (An, Nam) \wedge Anh\_của\ (Nam, An) \Rightarrow False$$

Trên thực tế, khi suy diễn thường sử dụng cách biểu diễn câu Horn sử dụng phép kéo theo hơn. Cách biểu diễn này cũng làm cho các câu Horn giống như các quy tắc *Nếu … Thì …* thường được sử dụng trong thực tế.

Các câu Horn có chứa đúng một literal dương gọi là *câu xác định* (*definite clause*). Các ví dụ thứ một, hai, ba ở trên đều là các câu xác định. Các câu Horn chỉ chứa literal dương mà không chứa literal âm nào như các ví dụ 1 và 2 ở trên được gọi là *sự kiện* (*fact*).

Trong phần dưới đây, ta sẽ xem xét cách suy diễn sử dụng Modus Ponens tổng quát trong trường hợp cơ sở tri thức chỉ gồm các câu xác định.

3.5.2. Suy diễn tiến và suy diễn lùi

Sử dụng quy tắc Modus Ponens tổng quát cho phép xây dựng thuật toán suy diễn tự động, cụ thể là phương pháp suy diễn tiến và suy diễn lùi. Suy diễn tiến và lùi có thể áp dụng đối với *KB chỉ chứa các câu xác định*, tức là các câu Horn với đúng một literal dương.

Suy diễn tiến (forward chaining)

Giả sử ta có KB bao gồm các câu xác định. Thủ tục suy diễn tiến được thực hiện như sau: bắt đầu từ các câu trong KB, áp dụng Modus Ponens để sinh ra các câu mới cho đến khi không thể sinh ra thêm câu nào nữa. Nếu các câu trong KB được biểu diễn dưới dạng quy tắc kéo theo thì việc suy diễn được thực hiện theo chiều của phép kéo theo, tức là từ các tiền để suy ra kết luận, do vậy suy diễn được gọi là suy diễn tiến.

Thủ tục suy diễn tiến được mô tả như dưới đây.

- Khi câu *p* mới được thêm vào KB:
  - với mỗi quy tắc *q* mà *p* hợp nhất được với một phần vế trái:
    - Nếu các phần còn lại của vế trái đã có thì thêm vế phải vào KB và suy diễn tiếp

Hình 3.2. Thủ tục suy diễn tiến

Ví dụ:

Cho KB gồm các câu sau

1. Mèo thích cá
2. Mèo ăn gì nó thích
3. Có con mèo tên là Tom

Cần xác định:

Tom có ăn cá không?

Lời giải: Trước hết, viết các câu trên dưới dạng logic vị từ

(1) ∀x Mèo(x) => Thích(x, cá)

(2) ∀x, y Mèo(x) ∧ Thích (x,y) => Ăn(x,y)

(3) Mèo (Tom)

Câu truy vấn Q: Ăn (Tom, cá) ?

Lần lượt thêm các câu vào KB, ta có:

Thêm câu (1): không hợp nhất được với vế trái câu nào.

Thêm câu (2): không hợp nhất được với vế trái câu nào.

Thêm câu (3): hợp nhất được với vế trái câu (1), quy tắc GMP sinh ra câu (4)

(4) GMP (1) (3) => Thích (Tom, cá) { x / Tom}

Thêm câu (4): hợp nhất được với một phần vế trái câu (2), quy tắc GMP sinh ra câu (5)

(5) GMP (4) (3) (2) => Ăn(Tom, cá) { x / Tom, y / cá}

Đây chính là câu truy vấn cần tìm.

Thêm câu (5): không hợp nhất được với vế trái câu nào, do đó quá trình suy diễn dừng lại.

Nhận xét: Suy diễn tiến thêm dần các câu vào KB khi có các câu mới xuất hiện. Quá trình suy diễn này không hướng tới câu truy vấn hay kết luận cụ thể nào mà chỉ được khởi động khi có thêm câu mới.

Nếu KB chỉ chứa các câu Horn xác định thì suy diễn tiến là thủ tục suy diễn *đúng đắn*, tức là chỉ sinh ra những câu thực sự là hệ quả logic của KB. Tính đúng đắn của suy diễn tiến được suy ra từ tính đúng đắn của Modus Ponens tổng quát.

Nếu KB chỉ chứa các câu Horn xác định thì suy diễn tiến là thủ tục suy diễn đầy đủ, tức là có thể sinh ra tất cả các câu là hệ quả logic của KB. Tuy nhiên, do không phải câu logic vị

từ nào cũng có thể biến đổi về dạng câu xác định nên suy diễn tiến không phải là thủ tục suy diễn đầy đủ đối với logic vị từ nói chung.

Suy diễn lùi (Backward chaining)

Thủ tục suy diễn tiến trình bầy ở trên bắt đầu từ các câu đã có trong KB và sinh ra các câu mới bằng cách sử dụng quy tắc GMP. Một vấn đề với suy diễn tiến là số câu sinh ra có thể rất nhiều, trước khi sinh ra được câu truy vấn mà cần xác định tính đúng sai. Ngược lại với suy diễn tiến, *suy diễn lùi* bắt đầu từ cầu truy vấn, sau đó tìm các sự kiện và quy tắc trong KB cho phép chứng minh câu truy vấn là đúng. Quá trình suy diễn có thể coi như được tiến hành ngược với chiều của phép kéo theo, tức là từ hệ quả ta tìm cách tìm ra các tiền đề làm cho hệ quả đó đúng. Suy diễn lùi rất phù hợp với việc trả lời câu hỏi hoặc chứng minh một câu cụ thể là đúng hay sai từ các câu có trong KB.

Quá trình suy diễn lùi được tiến hành như sau. Thủ tục suy diễn nhận câu hỏi hoặc câu cần chứng minh (gọi chung là câu truy vấn) dưới dạng một câu nguyên tử. Câu truy vấn có thể chứa lượng tử với mọi và biến tương ứng. Kết quả trả về là chuỗi các phép thế nếu có thể chứng minh câu truy vấn là đúng. Trước hết, thủ tục suy diễn tìm cách hợp nhất câu truy vấn với các các sự kiện trong KB (nhắc lại: sự kiện là câu Horn chỉ chứa một literal dương và không có literal âm). Nếu không được thì tìm luật và vế phải có thể hợp nhất với câu truy vấn sau đó tìm cách chứng minh vế trái một cách đệ quy bằng cách dùng cùng một thủ tục như chứng minh câu truy vấn.

Thủ tục suy diễn lùi được mô tả trên hình sau.

- Với câu hỏi q, nếu tồn tại q' hợp nhất với q thì trả về hợp tử
- Với mỗi quy tắc có vế phải q' hợp nhất với q cố gắng chứng minh các phần tử vế trái bằng suy diễn lùi

Hình 3.3. Thủ tục suy diễn lùi

Để minh họa cho thủ tục suy diễn lùi, ta sẽ sử dụng lại ví dụ đã dùng trong phần suy diễn tiến. Cụ thể, để chứng minh Ăn(Tom, cá) là đúng, theo suy diễn lùi, ta nhận thấy:

Ăn(Tom, cá) hợp nhất với vế phải của (2) với phép thế {x/ Tom, y/ cá}.

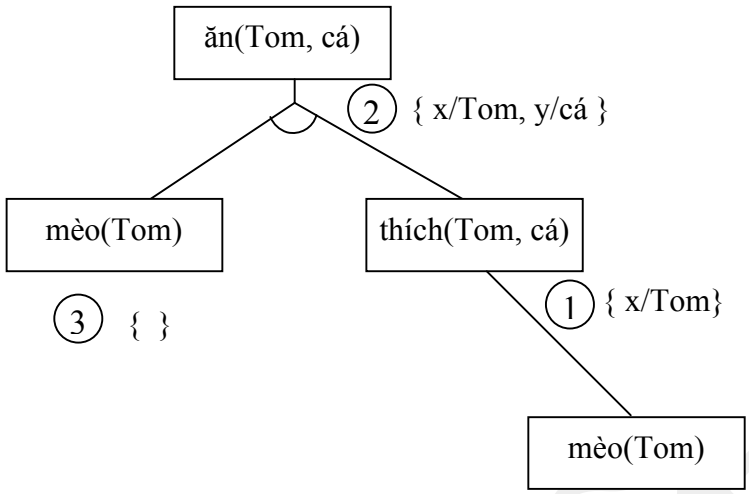
Vế trái của (2), sau khi thực hiện phép thế trên, sẽ gồm hai phần Mèo (Tom) và Thích (Tom, cá), cần chứng minh hai phần này.

Mèo(Tom) có thể chứng minh do hợp nhất với sự kiện trong câu (3).

Thích (Tom, cá) hợp nhất với vế phải câu (1) nhờ phép thế {x / Tom}, cần chứng minh vế trái của (1) là Mèo(Tom). Câu này cũng là đúng do hợp nhất được với (3).

Vậy ta hoàn thành việc chứng minh Ăn (Tom, cá) bằng suy diễn lùi với phép thế {x/ Tom, y/ cá}.

Quá trình chứng minh có thể minh họa một cách thuận tiện dưới dạng đồ họa như trên hình sau.



Hình 3.4. Cây suy diễn cho trường hợp suy diễn lùi

Tính chất suy diễn lùi: Suy diễn lùi là đúng đắn. Nếu KB chỉ gồm các câu xác định thì suy diễn lùi là thủ tục suy diễn đầy đủ, tức là có thể chứng minh mọi hệ quả logic của KB. Trong trường hợp chung, do không phải tất cả các câu đều có thể đưa về dạng câu Horn nên suy diễn lùi không đầy đủ. Thủ tục suy diễn lùi khá hiệu quả về mặt độ phức tạp tính toán và được sử dụng làm cơ chế suy diễn trong ngôn ngữ Prolog.

3.5.3. Suy diễn sử dụng phép giải

Phép giải (resolution) cho logic vị từ

Như đã đề cập ở trên, ta có phép giải cho logic mệnh đề

$$\alpha \vee \beta \vee \lambda \ , \neg \ \lambda \vee \gamma \vee \mu \ \Rightarrow \alpha \vee \beta \vee \gamma \vee \mu$$

Đối với logic vị từ, phép giải có dạng như sau:

Cho câu: $P_1 \vee P_2 \vee \ldots \vee P_n$

Và câu: $Q_1 \vee Q_2 \vee \ldots \vee Q_m$

Trong đó P_i, Q_i là literal

Nếu P_j và $\neg\ Q_k$ có thể hợp nhất bởi hợp tử θ thì ta có phép giải:

$$\frac{P_1 \vee P_2 \vee \ldots \vee P_n \ , Q_1 \vee Q_2 \vee \ldots \vee Q_m}{\text{SUBST}(\theta, P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \vee \ldots \vee P_n \vee Q_1 \vee \ldots \vee Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m)}$$

Ví dụ:

1. Cho hai câu

Giàu (x) ∨ Giỏi (x)

và ¬ Giỏi (Bắc) ∨ Đẹp trai (Bắc)

suy ra

Giàu (Bắc) ∨ Đẹp trai (Bắc)

2. Cho các câu

P (x, f(a)) ∨ P(x, f(y)) ∨ Q (y)

và ¬ P(z, f (a)) ∨ ¬ Q(z)

suy ra

P (z, f(y)) ∨ Q (y) ∨ ¬ Q(z) θ {x/z}

hoặc P (x, f(a)) ∨ P (x, f(z)) ∨ ¬ P (z, f(a)) θ {y/z}

Dạng Conjunctive Normal Form (CNF) và câu tuyển

Các công thức tương đương có thể xem như các biểu diễn khác nhau của cùng một sự kiện. Để dễ dàng viết các chương trình máy tính thao tác trên các công thức, chúng ta sẽ chuẩn hóa các công thức, đưa chúng về dạng biểu diễn chuẩn.

Như đã nói ở trên, ta định nghĩa mỗi câu tuyển là tuyển của literal, có dạng $A_1 \vee A_2 \vee ... . \vee A_m$ trong đó các A_i là literal.

Một dạng chuẩn được gọi là Conjunctive Normal Form (CNF - dạng chuẩn hội), là câu bao gồm hội của phép tuyển của các literal, tức là hội của các câu tuyển.

KB dưới dạng chuẩn hội có khả năng biểu diễn tốt hơn các câu Horn. Chúng ta có thể biến đổi một công thức bất kỳ về công thức ở dạng CNF bằng cách áp dụng một số bước thủ tục nhất định sẽ được trình bày ở phần sau.

Suy diễn sử dụng phép giải và phản chứng (Resolution Refutation)

Nếu KB là tập hữu hạn các câu thì các literal có mặt trong các câu của KB cũng là hữu hạn. Do đó số các câu tuyển thành lập được từ các literal đó là hữu hạn. Vì vậy chỉ có một số hữu hạn câu được sinh ra bằng luật giải. Phép giải sẽ dừng lại sau một số hữu hạn bước. Sử dụng phép giải ta có thể chứng minh được một câu có là tập con của một KB đã cho hay không bằng phương pháp chứng minh phản chứng.

KB: ├ Q?

Thêm ¬ Q vào KB, sau đó chứng minh tồn tại một tập con của KB mới có giá trị False

(KB ├ Q) <=> (KB ∧ ¬ Q ├ False)

Nói cách khác: phương pháp chứng minh tạo cơ sở tri thức mới bao gồm KB và ¬ Q, sau đó dùng phép giải để chứng minh từ cơ sở tri thức mới suy ra False. Một trong các lý do kết hợp phép giải và phản chứng là do suy diễn sử dụng phép giải là thủ tục suy diễn không đầy đủ. Ví dụ, dùng phép giải ta không thể chứng minh P ∨ ¬P từ KB rỗng mặc dù đây là công thức vững chắc, tức là đúng trong mọi minh họa, do trong KB không tồn tại câu nào để áp dụng phép giải. Việc sử dụng phản chứng cho phép suy diễn trong những trường hợp như vậy.

Thủ tục suy diễn sử dụng phép giải và phản chứng được thể hiện trên hình sau.

Đầu vào: cơ sở tri thức KB, câu truy vấn Q
KB = UNION (KB, ¬ Q) // thêm ¬ Q vào KB
While (KB không chứa False) do
    Chọn 2 câu S_1, S_2 từ KB sao cho có thể áp dụng phép giải cho 2 câu này
    IF không có hai câu như vậy
        Return *Không chứng minh được*
    Thêm kết quả phép giải vào KB
Return *Câu Q được chứng minh*

Hình 3.5. Suy diễn bằng phép giải và phản chứng

Về tính đầy đủ của suy diễn sử dụng phép giải

Suy diễn sử dụng phép giải là phản chứng – đầy đủ, tức là nếu một tập hợp các câu là không thỏa được trong một minh họa nào đó thì thủ tục suy diễn sử dụng phép giải luôn cho phép tìm ra mâu thuẫn. Như vậy, phép giải cho phép xác định một câu có là hệ quả logic của một tập các câu khác không. Tuy nhiên, phép giải không cho phép sinh ra tất cả các câu là hệ quả logic của một tập câu cho trước.

Ví dụ: ví dụ dưới minh hoạ cho việc chứng minh bằng phép giải và phản chứng trong trường hợp KB gồm các câu clause.

KB: ¬ A ∨ ¬ B ∨ P (1)
¬ C ∨ ¬ D ∨ P (2)
¬ E ∨ C (3)
A (4)
E (5)
D (6)

Ta cần chứng minh KB: ├ P.

Các bước chứng minh sẽ như sau:

Thêm vào KB câu sau:

¬ P (7)

Áp dụng phép giải cho câu (2) và (7) ta được câu:

¬ C ∨ ¬ D (8)

Từ câu (6) và (8) ta nhận được câu:

¬ C (9)

Từ câu (3) và (9) ta nhận được câu:

¬ E (10)

Câu (10) mang giá trị False. Tới đây ta đã tìm thấy một tập con của KB mới có giá trị False .

Kết luận : Từ KB suy ra P

Biến đổi các câu về dạng CNF và Clause Form

Khi biểu diễn tri thức bởi các câu trong logic vị từ, KB là một tập các câu. Để sử dụng thuật toán trên thì ta thực hiện chuẩn hóa các câu trong KB để chuyển về dạng Clause form bằng cách sử dụng các bước sau:

Bước 1: Khử tương đương

Để khử phép tương đương thay P⇔Q bằng (P=>Q)∧(Q=> P).

Bước 2: Loại bỏ kéo theo

Để loại bỏ các kéo theo, ta chỉ cần thay thế công thức P⇒Q bởi công thức tương đương ¬ P ∨ Q

Bước 3: Đưa các phủ định vào gần các vị từ

Chuyển các dấu phủ định (¬) vào sát các vị từ bằng cách áp dụng luật De Morgan và thay ¬ (¬ A) bởi A . Điều này được thực hiện bằng cách thay công thức ở vế trái bằng công thức ở vế phải trong các tương đương sau:

¬ (¬ P) ≡ P

¬ (P∧Q) ≡ ¬ P∨ ¬ Q

¬ (P ∨ Q) ≡ ¬ P∧¬ Q

¬ (∀x P) ≡ ∃ x (¬ P)

¬ (∃ x P) ≡ ∀ x (¬ P)

Bước 4: Chuẩn hóa tên biến sao cho mỗi lượng tử có biến riêng

Ví dụ :

∀x ¬ P (x) ∨ Q (x) ⟹ ∀x ¬ P (x) ∨ Q (x)

∀x ¬ R (x) ∨ Q (x) ∀y ¬ R (y) ∨ Q (y)

Bước 5: Loại bỏ các lượng tử tồn tại bằng cách sử dụng hằng Skolem và hàm Skolem

Trong bước này ta bỏ lượng tử tồn tại bằng cách thay các biến của lượng tử này bằng hằng Skolem hoặc hàm Skolem. Cụ thể là, khi lượng tử tồn tại nằm trong lượng tử với mọi thì ta dùng hàm Skolem, còn trong trường hợp khác ta sử dụng hằng Skolem.

Ví dụ, $\exists x\ P(x)$ được biến đổi thành $P(C)$, trong đó C là hằng Skolem chưa xuất hiện trong KB, do lượng tử tồn tại không nằm trong bất cứ lượng tử với mọi nào.

Ngược lại, nếu lượng tử tồn tại nằm trong lượng tử với mọi thì ta sử dụng hàm Skolem, tức là một hàm chưa xuất hiện trong KB. Sự cần thiết sử dụng hàm Skolem được giải thích qua ví dụ sau. Giả sử P(x,y) là các vị từ có nghĩa rằng "y lớn hơn x" trong miền các số. Khi đó câu ∀x (∃y (P(x,y)) có nghĩa là "với mọi số x tồn tại y sao cho y lớn hơn". Ta có thể xem y trong câu đó là hàm của đối số x, chẳng hạn f(x) và loại bỏ lượng tử ∃y, câu đang xét trở thành ∀x (P(x,f(x)).

Một cách tổng quát, giả sử ∃y (G) là một câu con của câu đang xét và nằm trong miền tác dụng của lượng tử $\forall x_1,.....,\forall x_n$. Khi đó ta có thể xem y là hàm của n biến $x_1,.....,$

x_n, chẳng hạn $f(x_1,....., x_n)$. Sau đó ta thay các xuất hiện của y trong câu G bởi hạng thức $f(x_1,....., x_n)$ và loại bỏ các lượng tử tồn tại. Các hàm f được đưa vào để loại bỏ các lượng tử tồn tại được gọi là hàm Skolem.

Ví dụ: Xét câu sau:

$$\forall x\ (\exists y\ (P(x,y)\ \vee\ \forall u\ (\exists b\ (Q(a,b) \wedge \exists\ t \neg\ R(x,t))) \qquad (1)$$

Câu con $\exists y\ P(x,y)$ nằm trong miền tác dụng của lượng tử $\forall x$, ta xem y là hàm của x: F(x). Các câu con $\exists b\ (Q(a,b)$ và $\exists\ t \neg R(x,y)$ nằm trong miền tác dụng của các lượng tử $\forall x$, $\forall u$ nên ta xem b là hàm g(x,u) và t là hàm h(x,u) của 2 biến x,u. Thay các xuất hiện của y và b, t bởi các hàm tương ứng, sau đó loại bỏ các lượng tử tồn tại, từ câu (1) ta nhận được câu :

$$\forall x\ (\ (P(x,f(x))\ \vee \forall u\ (Q(a,g(x,u)) \wedge \neg\ R(x,h(x,u))))$$

Bước 6: Loại bỏ các lượng từ với mọi (∀)

Để loại bỏ lượng từ với mọi (∀), ta đưa các lượng tử với mọi (∀) sang trái sau đó bỏ lượng tử với mọi (∀).

Ví dụ: $\forall x\ (\ P\ (x,y)\ \vee\ Q\ (x)\ )\ \Rightarrow\ P\ (x,y)\ \vee\ Q\ (x)$

Bước 7: Sắp xếp các phép và, hoặc để có dạng CNF

Ví dụ:

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

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

Bước 8: Loại bỏ các phép và

Ta thực hiện loại bỏ các phép và để tạo thành các clause riêng

Ví dụ :

$$(\ P \vee R \vee S\ ) \wedge (\ Q \vee \neg\ R\ ) \Rightarrow P \vee R \vee S,\quad Q \vee \neg\ R$$

Bước 9 : Chuẩn hóa tên biến sao cho mỗi câu có biến riêng của mình

Ví dụ:

1) $\neg\ P(x) \vee\ P(y) \vee\ Q\ (f(x,y)\ )$ ⟹ 1) $\neg\ P(x) \vee\ P(y) \vee\ Q\ (f(x,y)\ )$

2) $\neg\ P(x) \vee\ Q(x,\ g(x)\ )$ ⟹ 2) $\neg\ P(z) \vee\ Q(z,\ g(z)\ )$

3) $P(x) \vee \neg\ R(g(x))$ ⟹ 3) $P(u) \vee \neg\ R(g(u))$

Ví dụ: Sau đây là ví dụ minh hoạ cho việc áp dụng đầy đủ các bước kể trên

Chuẩn hóa công thức sau:

$$\forall x\ (\ P(x) \Rightarrow (\ \forall y\ (P(y) \Rightarrow P\ (f(x,y)\ )) \wedge \neg\ \forall y\ (Q(x,y) \Rightarrow P(y))\ )\ )$$

Ta sẽ lần lượt thực hiện theo từng bước cụ thể như sau:

1. Khử tương đương: không cần thực hiện
2. Loại bỏ kéo theo

$$\forall x\ (\neg P(x) \vee (\forall y\ (\neg P(y) \vee P(f(x,y))) \wedge \neg \forall y\ (\neg Q(x,y) \vee P(y))))$$

3. Đưa phủ định vào gần các vị từ

$$\forall x\ (\neg P(x) \vee (\forall y\ (\neg P(y) \vee P(f(x,y))) \wedge \exists y\ \neg\ (\neg Q(x,y) \vee P(y))))$$

$$\forall x\ (\neg P(x) \vee (\forall y\ (\neg P(y) \vee P(f(x,y))) \wedge \exists y\ (Q(x,y) \wedge \neg P(y))))$$

4. Chuẩn hóa tên biến sao cho mỗi lượng tử có biến riêng

$$\forall x\ (\neg P(x) \vee (\forall y\ (\neg P(y) \vee P(f(x,y))) \wedge \exists z\ (Q(x,z) \wedge \neg P(z))))$$

5. Loại bỏ các lượng tử tồn tại bằng cách sử dụng hằng Skolem và hàm Skolem

$$\forall x\ (\neg P(x) \vee (\forall y\ (\neg P(y) \vee P(f(x,y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$$

6. Loại bỏ lượng từ với mọi (∀)

$$(\neg P(x) \vee ((\neg P(y) \vee P(f(x,y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$$

7. Sắp xếp phép và và phép hoặc để có dạng CNF

$$[\neg P(x) \vee \neg P(y) \vee P(f(x,y))] \wedge [\neg P(x) \vee Q(x, g(x))] \wedge [\neg P(x) \vee \neg P(g(x))]$$

8. Bỏ phép và để tạo thành các clause riêng: ta được 3 câu clause sau

$$1)\ \neg P(x) \vee \neg P(y) \vee P(f(x,y))$$

$$2)\ \neg P(x) \vee Q(x, g(x))$$

$$3)\ \neg P(x) \vee \neg P(g(x))$$

9. Chuẩn hóa tên biến sao cho mỗi câu có biến riêng của mình

$$1)\ \neg P(x) \vee \neg P(y) \vee P(f(x,y))$$

$$2)\ \neg P(z) \vee Q(z, g(z))$$

$$3)\ \neg P(k) \vee \neg P(g(k))$$

Ví dụ. Sau đây là ví dụ minh hoạ cho bài toán suy diễn sử dụng phép giải và phản chứng. Ta sẽ sử dụng lại ví dụ đã dùng trong phần về suy diễn tiến, lùi. Cụ thể, cho KB gồm các câu sau

1. Mèo thích cá
2. Mèo ăn gì nó thích
3. Có con mèo tên là Tom

Cần xác định: Tom có ăn cá không?

Lời giải: Trước hết, viết các câu trên dưới dạng logic vị từ

(1) ∀x Mèo(x) => Thích(x, cá)

(2) ∀x, y Mèo(x) ∧ Thích (x,y) => Ăn(x,y)

(3) Mèo (Tom)

Câu truy vấn Q: Ăn (Tom, cá) ?

Lời giải:

Theo nguyên tắc phản chứng, ta giả sử câu Q sai, tức là ¬Q đúng. Thêm ¬Q vào KB, ta được:

(1) ∀x Mèo(x) => Thích(x, cá)

(2) ∀x, y Mèo(x) ∧ Thích (x,y) => Ăn(x,y)

(3) Mèo (Tom)

(4) ¬ Ăn (Tom, cá)

Biến đổi về dạng clause, ta được:

(1) ¬Mèo(x) ∨ Thích(x, cá)

(2) ¬Mèo(z) ∨ ¬Thích (z,y) ∨ Ăn(z,y)

(3) Mèo (Tom)

(4) ¬ Ăn (Tom, cá)

Áp dụng phép giải:

(5): (1)+(3) Thích(Tom, cá) {x/Tom}

(6): (2)+(3) ¬Thích (Tom, y) ∨ Ăn(Tom, y) {z/Tom}

(7): (5)+(6) Ăn(Tom, cá) {y/cá}

(8): (4)+(7) False điều phải chứng minh

Vậy câu truy vấn Ăn(Tom, cá) là đúng, theo nguyên tắc phản chứng.

3.5.4.Hệ thống suy diễn tự động: lập trình logic

Trên thực tế, việc biểu diễn tri thức và suy diễn logic được thực hiện bằng cách sử dụng một số ngôn ngữ lập trình được thiết kế đặc biệt. Kỹ thuật xây dựng hệ thống suy diễn như vậy được gọi là lập trình logic (logic programming). Ngôn ngữ lập trình logic tiêu biểu là Prolog. Rất nhiều hệ chuyên gia trong nhiều lĩnh vực khác nhau đã được xây dựng trên ngôn ngữ Prolog.

Chương trình trên Prolog là một tập hợp các câu xác định (definite clause). Tuy nhiên, để thuận tiện cho việc viết trên máy tính, các câu này có cú pháp không hoàn toàn giống với logic vị từ truyền thống.

Suy diễn được thực hiện theo kiểu suy diễn lùi và tìm kiếm theo chiều sâu, trong đó các câu được xét theo thứ tự xuất hiện của câu trong chương trình. Ngoài ra, Prolog cũng cho phép chứng minh bằng cách phủ định câu truy vấn, sau đó dẫn tới kết luận rằng không thể chứng minh được câu phủ định này.

3.6. CÂU HỎI VÀ BÀI TẬP CHƯƠNG

1. Các khẳng định nào sau đây là đúng:

a) False ⊨ True

b) True ⊨ False

c) (A ⇔ B) ⊨ (A ∧ B)

d) (A ∧ B) ⊨ (A ⇔ B)

e) (A ⇔ B) ⊨ (A v B)

f) (A ⇔ B) ⊨ (¬A v B)

g) A ∧ B ⇒ C ⊨ (A ⇒ C) v (B ⇒ C)

2. Cho KB = (A v B) ∧ (¬ C v ¬ D v E)

Các câu nào sau đây sinh ra từ KB?

1- A v B

2- (A v B v C) ∧ ((B ∧ C ∧ D) => E)

3. Cho KB:

Red	Blue => Silver
¬ Pink v Blue	Pink => Tan
Tan ∨ Orange	Silver
¬ Pink	¬ (Violet ∨ White)
Pink => Red	Blue => Orange

Hãy chứng minh:

a) ¬ Orange

b) Silver ∧ Red

c) Silver ∨ White

4. Viết các câu sau dưới dạng logic vị từ:

1 - Mọi nhà nông thích mặt trời

2 - Lúc nào cũng có người bị lừa

3 - Nấm có màu đỏ là nấm độc

4 - An không cao

5 - Chỉ có 2 sinh viên nước ngoài học lớp công nghệ thông tin

6 - Trên trời có muôn vàn vì sao

5. Cho các câu sau trên logic vị từ:

1 - trâu (x) ∧ lợn (y) => to_hơn(x, y)

2 - lợn (y) ∧ chuột (z) => to_hơn(y, z)

3 - to_hơn(x, y) ∧ to_hơn (y, z) => to_hơn(x, z)

4 - trâu (Bob)

5 - lợn (Pat)

6 - chuột (Jerry)

Thực hiện suy diễn tiến ra các câu có thể sử dụng suy diễn tiến.

6. Cho KB gồm các câu sau:

1- Lợn (y) ∧ Ốc_ sên (z) => Nhanh_hơn (y, z)

2- Nhỏ (z) ∧ Biết_ bò (z) => Ốc_sên (z)

3- Lợn (Pat)

4- Nhỏ (Steve)

5- Biết_bò (Steve)

Hãy chứng minh bằng suy diễn tiến và lùi câu sau:

Nhanh_hơn (Pat, Steve)?

7. Tìm MGU (hợp tử tổng quát nhất) cho các cặp câu sau:

a. P (A, B, B) , P (x, y, z)
b. Q (y, G(A, B)) , Q (G (x, x), y)
c. Older (Father (y), y) , Older(Father (x), John)
d. Knows (Father(y), y), Knows (x, x)

8. Biểu diễn các câu sau dưới dạng logic vị từ phù hợp với việc sử dụng Modus Ponens tổng quát

1. Ngựa, Bò, Lợn là động vật
2. Con của ngựa là ngựa
3. Ngựa tên là Xích Thố
4. Xích Thố là bố của con Chiếu Dạ
5. Quan hệ cha con là quan hệ nghịch đảo
6. Tất cả động vật đều có bố

9. Từ các câu trong câu 9 hãy sử dụng suy diễn lùi chứng minh con Chiếu Dạ là con ngựa.

10. Cho biết:

Tôi không có anh chị em

Bố của người đó là con trai của bố tôi

Hãy xây dựng KB gồm các câu liên quan, sau đó sử dụng phép giải và phản chứng để tìm xem người đó là ai.

11. Cho biết:

Có 3 người Bắc, Đông, Nam tham gia câu lạc bộ.

Mỗi thành viên câu lạc bộ là leo núi hoặc trượt tuyết

Không có người leo núi nào thích mưa

Tất cả những người trượt tuyết thích tuyết

Đông ghét tất cả những gì Nam thích

Đông thích tất cả những gì Nam ghét

Nam thích mưa và tuyết

Hỏi có thành viên nào của câu lạc bộ là người leo núi không phải là người trượt tuyết? Trả lời câu hỏi bằng cách thực hiện các bước sau:

a. Dịch sang logic vị từ

b. Chuyển dạng clause form

c. Dùng phép giải và phản chứng để chứng minh.

12. Cho các câu sau:

1. ∀x Kem (x) => Thức_ăn (x)
2. ∀x Caramen(x) => Thức_ăn (x)
3. ∀x, y Thức_ăn (x) ∧ Thức_ăn(y) ∧ Lạnh(x) ∧ Trộn (x,y) => Lạnh (y)
4. ∃x ∃y Kem (x) ∧ Lạnh (x) ∧ Caramen (y) ∧ Trộn (x,y)

a. Dịch các câu trên sang tiếng Việt

b. Giả sử KB gồm các câu trên, hãy sử dụng phản chứng và phép giải để chứng minh câu sau:

∃x (Caramen (x) ∧ Lạnh (x))

CHƯƠNG 4: LẬP LUẬN XÁC SUẤT

4.1. VẤN ĐỀ THÔNG TIN KHÔNG CHẮC CHẮN KHI LẬP LUẬN

Trong chương trước, ta đã xem xét cách biểu diễn tri thức bằng logic cũng như một số phương pháp suy diễn logic để đưa ra quyết định. Mặc dù các hệ thống logic cho phép biểu diễn tri thức một cách rõ ràng và tường minh, việc sử dụng logic đòi hỏi tri thức phải được cung cấp một cách đầy đủ, chính xác, không mâu thuẫn. Nếu yêu cầu này không thỏa mãn, các hệ thống suy diễn dựa trên tri thức vị từ hoặc logic mệnh đề không thể sử dụng được.

Trên thực tế, các thông tin, tri thức hay mô hình về thế giới xung quanh thường không đầy đủ, không rõ ràng, không chính xác, có thể mâu thuẫn với nhau. Có thể liệt kê một số nguyên nhân gây ra sự không rõ ràng, không chắc chắn như dưới đây.

- *Do thông tin có chứa đựng yếu tố ngẫu nhiên, yếu tố xác suất*. Ví dụ khi chơi các trò chơi liên quan tới xác suất như chơi bài, chơi cá ngựa.
- *Do không có hiểu biết đầy đủ về vấn đề đang xét*. Ví dụ khi xây dựng hệ thống chẩn đoán bệnh trong y học, do y học hiện đại chưa hiểu biết hoàn toàn chính xác về cơ chế bên trong của nhiều loại bệnh, việc xây dựng quy tắc suy diễn dựa trên kiến thức như vậy sẽ không chắc chắn, không chính xác.
- *Do số các yếu tố liên quan đến quá lớn, không thể xem xét hết*. Ví dụ khi chẩn đoán bệnh, bác sĩ có thể phải chẩn đoán trong khi không có đầy đủ thiết bị để làm mọi loại xét nghiệm cần thiết.
- *Do sai số khi ta lấy thông tin từ môi trường.* Ví dụ trong trường hợp các thiết bị đo có thể có sai số.

Như vậy, trên thực tế, nhiều bài toán đòi hỏi lập luận, ra quyết định trong khi chưa có đầy đủ thông tin hay thông tin không rõ ràng, không chắc chắn. Vấn đề này thường được gọi là lập luận trong điều kiện không rõ ràng (reasoning under uncertainty) và không thể giải quyết được bằng lập luận logic truyền thống.

Để lập luận trong điều kiện không rõ ràng, nhiều cách tiếp cận khác nhau đã được nghiên cứu phát triển, trong đó phải kể đến:

- Cách tiếp cận dựa trên logic đa trị: thay vì chỉ cho phép đưa ra kết luận "đúng" hoặc "sai" như logic truyền thống, logic đa trị cho phép sử dụng nhiều giá trị hơn, ví dụ thêm giá trị "không đúng không sai".
- Logic mờ (fuzzy logic): thay vì hai giá trị "đúng" hoặc "sai", biểu thức logic mờ có thể nhận giá trị "đúng" với một giá trị hàm thuộc nằm trong khoảng [0,1], thể hiện mức độ đúng của biểu thức đó.
- Lý thuyết khả năng (possibility theory): các sự kiện hoặc công thức được gán một số thể hiện khả năng xảy ra sự kiện đó.
- Suy diễn xác suất: kết quả suy diễn trả về xác suất một sự kiện hay công thức nào đó là đúng.

Trong chương này, ta sẽ xem xét phương pháp lập luận xác suất cho trường hợp lập luận có yếu tố không rõ ràng.

4.2. NGUYÊN TẮC LẬP LUẬN XÁC SUẤT

Khác với suy diễn logic, trong đó các sự kiện, mệnh đề, công thức nhận giá trị đúng hoặc sai, trong suy diễn xác suất, thay vì kết luận một sự kiện hay công thức đúng hoặc sai, ta tính toán niềm tin là sự kiện, công thức đó là đúng hay sai. Niềm tin ở đây được định lượng bằng xác suất, quá trình tính toán tuân theo các công thức của lý thuyết xác suất. Như vậy, thay vì chỉ sử dụng hai giá trị đúng hoặc sai, suy diễn xác suất cho phép làm việc với vô số giá trị.

Cụ thể, mỗi mệnh đề hoặc biểu thức được gắn một số đo giá trị niềm tin là mệnh đề đó đúng. Mức đo niềm tin được biểu diễn như giá trị xác suất và lý thuyết xác suất được sử dụng để làm việc với mức đo niềm tin này.

Chẳng hạn, với mệnh đề A, ta sử dụng xác suất $P(A)$, $0 \leq P(A) \leq 1$, với ý nghĩa $P(A) = 1$ nếu A đúng, $P(A) = 0$ nếu A sai. Như vậy, $P(A)$ có thể diễn giải như phần thế giới của bài toán trong đó mệnh đề A đúng. Nếu A luôn đúng trong thế giới đó thì $P(A) = 1$, ngược lại, nếu A luôn sai thì $P(A) = 0$.

Ví dụ:

- P (Cảm = true) = 0.6 : người bệnh bị cảm với xác suất 60%, "Cảm" là biến ngẫu nhiên có thể nhận 1 trong 2 giá trị {True, False}
- P (trời = nắng ^ gió = mạnh) = 0.8: ta tin rằng trời nắng và gió mạnh với xác suất 80%, trời là biến ngẫu nhiên nhận các giá trị {nắng, mưa, u ám}, gió là biến ngẫu nhiên nhận giá trị {mạnh, yếu, trung bình}.

Bản chất xác suất sử dụng trong suy diễn

Bản chất thống kê: trong lý thuyết xác suất truyền thống, giá trị xác suất được xác định dựa trên quan sát, thực nghiệm, thống kê. Ví dụ, để xác định xác suất trời mưa vào ngày lễ 1-5 ta cần quan sát và thống kê số ngày 1-5 trời mưa trong rất nhiều năm. Việc xác định xác suất như vậy không phải khi nào cũng thực hiện được.

Xác suất dựa trên chủ quan: trong suy diễn xác suất, khi không thể xác định giá trị xác suất bằng thống kê, xác suất có thể xác định một cách chủ quan, dựa trên niềm tin của chuyên gia, của người dùng về sự đúng, sai của các sự kiện. Ví dụ, bác sĩ có thể cung cấp giá trị xác suất về một triệu chứng bệnh nào đó dựa trên kinh nghiệm và ước lượng, thay vì dựa trên thống kê chính xác và cụ thể.

Thu thập và biểu diễn thông tin cho suy diễn xác suất

Để suy diễn xác suất cho một vấn đề nào đó, các bước sau cần được thực hiện:

- Xác định các tham số liên quan tới vấn đề, chẳng hạn trong chẩn đoán bệnh, tham số có thể là "đau đầu", "chán ăn". Mỗi tham số xác định ở trên được biểu diễn bằng một biến ngẫu nhiên tương ứng.
- Xác định miền giá trị cho các biến ngẫu nhiên. Thông thường, mỗi biến ngẫu nhiên có thể nhận một giá trị rời rạc trong miền giá trị của mình. Với những trường hợp đơn giản, biến chỉ có hai giá trị "đúng", "sai" như đối với biến "chán ăn". Những biến như vậy được gọi là biến bool hay biến nhị phân. Trong trường hợp chung, biến có thể

nhận nhiều giá trị hơn, chẳng hạn "đau đầu" có thể nhận giá trị "nhẹ", "dữ dội", "không".

- Xác định xác suất ứng với sự kiện biến nhận giá trị nào đó. Ví dụ, P(¬ chán ăn) = 0.9, P(đau đầu = dữ dội) = 0.15.

Việc suy diễn xác suất khi đó sẽ qui về tính xác suất cho một hoặc một số các biến nhận những giá trị nào đó.

4.3. MỘT SỐ KHÁI NIỆM VỀ XÁC SUẤT

Trước khi xem xét vấn đề biểu diễn tri thức và suy diễn xác suất, phần này nhắc lại một số kiến thức về xác suất cần thiết cho những nội dung tiếp theo:

4.3.1. Các tiên đề xác suất

Giả sử *A, B* là biến ngẫu nhiên có thể nhận một trong hai giá trị true (đúng) và false (sai), hay *A*, *B* là mệnh đề có thể nhận hai giá trị đúng hoặc sai. Khi sử dụng trong suy diễn xác suất, ta có ba tiên đề xác suất như sau:

1) $0 \le P(A) \le 1$
2) $P(\text{true}) = 1$; $P(\text{false}) = 0$. [2]
3) $P(A \vee B) = P(A) + P(B)$ nếu A và B không giao nhau

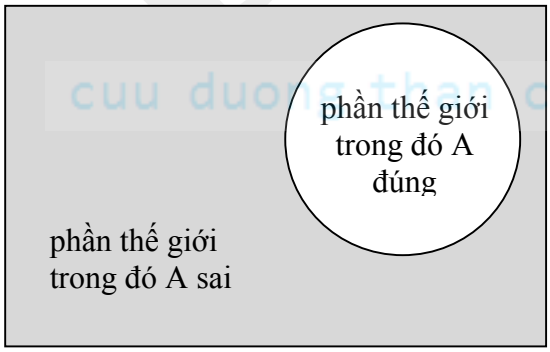
trong đó $P(A \vee B)$ là xác suất hoặc A đúng hoặc B đúng và $P(A \wedge B)$ là xác suất cả A và B đều đúng.

Tiên đề 3 có thể mở rộng cho trường hợp có nhiều hơn hai sự kiện ngẫu nhiên:

$P(X_1 \vee X_2 \vee ... \vee X_n) = \sum_{i=1}^{n} P(X_i)$ với X_i là sự kiện ngẫu nhiên không giao nhau

Diễn giải

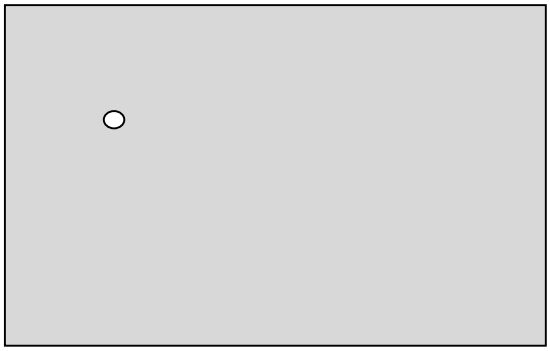
Ý nghĩa các tiên đề xác suất có thể diễn giải bằng cách sử dụng sơ đồ Venn. Trên hình ngay dưới đây, hình chữ nhật biểu diễn thế giới của bài toán; hình tròn mầu trắng thể hiện phần thế giới của bài toán trong đó A đúng. Phần mầu xám bên ngoài hình tròn là phần thế giới trong đó A sai. Xác suất $P(A)$ khi đó được xác định như tỷ lệ diện tích vùng A đúng so với toàn bộ thế giới của bài toán.

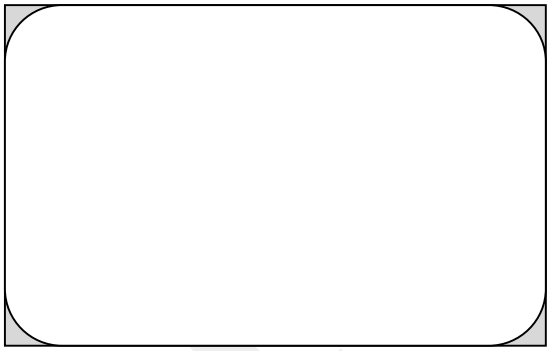


[2] Lưu ý, đây là tiên đề dùng cho suy diễn xác suất, trong trường hợp tổng quát, ta có P(Ω)=1, P(∅)=0, trong đó Ω là toàn bộ không gian lấy mẫu.

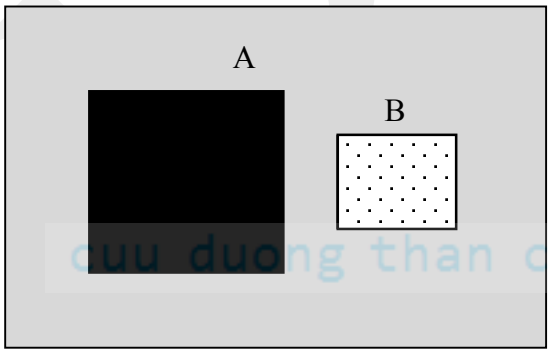
Do diện tích vùng hình tròn không thể nhỏ hơn 0 như thể hiện trên hình dưới bên trái nên ta có $0 \leq P(A)$. Diện tính hình tròn bằng 0 tương ứng với trường hợp A sai trong toàn bộ thế giới đang xét của bài toán.

Tương tự như vậy, do diện tích phần trắng không thể lớn hơn phần hình chữ nhật, tức là không thể vượt ra ngoài thế giới của bài toán như trong hình dưới bên phải nên ta có $P(A) \leq 1$. Diện tích phần trắng bằng 1 tương ứng với trường hợp A đúng trong toàn bộ thế giới đang xét của bài toán. Kết hợp các ý trên lại, ta được diễn giải của tiên đề 1 và tiên đề 2.





Ý nghĩa của tiên đề 3 được diễn giải trên hình dưới, trong đó hình chữ nhật gạch chéo với khung liền thể hiện phần của thế giới bài toán trong đó mệnh đề A đúng ($P(A)$), hình chữ nhật với nền có chấm thể hiện phần của thế giới bài toán trong đó mệnh đề B đúng ($P(B)$). Phần thế giới bài toán trong đó hoặc A đúng hoặc B đúng bao gồm cả hai hình chữ nhật. Do hai vùng chữ nhật không giao nhau nên phần thế giới trong đó hoặc A đúng hoặc B đúng bằng tổng của $P(A)$ và $P(B)$. Từ đây ta có tiên đề 3.



Các tính chất xác suất:

Ngoài các tiên đề trên, xác suất có một số tính chất quan trọng sau

- $P(\neg A) = 1 - P(A)$
- $P(A) = P(A \wedge B) + P(A \wedge \neg B)$
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

- $\Sigma_a P(A = a) = 1$, trong đó tổng lấy theo các giá trị a thuộc miền giá trị của A

Trường hợp biến ngẫu nhiên đa trị

Ở trên ta đã xét trường hợp biễn ngẫu nhiên có thể nhận một trong hai giá trị true hoặc false. Trong suy diễn nói chung, biễn ngẫu nhiên có thể nhận một trong nhiều hơn hai giá trị. Trong phần này ta sẽ xem xét các tính chất xác suất cho trường hợp biến ngẫu nhiên đa trị.

Giả sử A là biến ngẫu nhiên có thể nhận một trong n giá trị $\{v_1, v_2, \ldots, v_n\}$. Do A chỉ có thể nhận một trong các giá trị này nên hai sự kiện $(A = v_i)$ và $(A = v_j)$ không thể đồng thời đúng nếu $i \neq j$, và do vậy:

$$P(A = v_i \wedge A = v_j) = 0$$

Tiếp theo, do ít nhất một trong các sự kiện $(A = v_i)$, $i = 1,..,n$ phải đúng nên ta có:

$$P(A = v_1 \vee A = v_2 \vee \ldots \vee A = v_n) = 1$$

Sử dụng các tiên đề xác suất và hai tính chất trên, có thể chứng minh:

$$P(A = v_1 \vee A = v_2 \vee \ldots \vee A = v_k) = \sum_{i=1}^{k} P(A = v_i), \text{ với } k \leq n,$$

và do vậy có thể suy ra:

$$\sum_{i=1}^{n} P(A = v_i) = 1$$

Tiếp theo, giả sử B là biến ngẫu nhiên nhị phân. Sử dụng tiên đề xác suất và các tính chất trên, có thể chứng minh:

$$P(B \wedge [A = v_1 \vee A = v_2 \vee \ldots \vee A = v_k]) = \sum_{i=1}^{k} P(B \wedge A = v_i)$$

Từ đây suy ra:

$$P(B) = \sum_{i=1}^{n} P(B \wedge A = v_i)$$

Các tính chất nói trên sẽ được sử dụng khi trình bầy các vấn đề liên quan tới suy diễn cho trường hợp biến ngẫu nhiên đa trị trong các phần sau.

4.3.2. Xác suất đồng thời

Xác suất đồng thời của các sự kiện là xác suất quan sát thấy đồng thời xẩy ra các sự kiện đó. Xác suất đồng thời quan sát thấy các sự kiện $V_1 = v_1$, và $V_2 = v_2$, …, và $V_n = v_n$ được ký hiệu như sau:

$$P(V_1 = v_1 \wedge V_2 = v_2 \wedge \ldots \wedge V_n = v_n)$$

hoặc $P(V_1 = v_1, V_2 = v_2, \ldots, V_n = v_n)$

Một bài toán suy diễn xác suất gồm n biến ngẫu nhiên, mỗi biến có thể nhận một số giá trị rời rạc với những xác suất nhất định. Phân bố xác suất đồng thời xác định xác suất xẩy ra từng tổ hợp giá trị của tất cả n biến ngẫu nhiên.

Phân bố xác suất đồng thời đóng vai trò quan trọng trong suy diễn xác suất. Với một bài toán suy diễn xác suất, nếu chúng ta biết phân bố xác suất đồng thời tức là xác suất tất cả các tổ hợp giá trị của các biến liên quan thì ta có thể tính được xác suất mọi mệnh đề liên quan tới bài toán đang xét.

Ví dụ: Cho 3 biến nhị phân (True, False): "Chim", "Non", "Bay được". Ta có các xác suất đồng thời cho trong bảng 4.1. Mỗi dòng trong bảng này tương ứng với một tổ hợp giá trị của ba biến ngẫu nhiên đang xét. Số dòng trong bảng bằng tổng số tổ hợp có thể có, tức là 2^3 trong trường hợp này và 2^m trong trường hợp ta có m biến nhị phân. Cột ngoài cùng bên phải của bảng chứa xác suất xảy ra tổ hợp giá trị tương ứng. Lưu ý rằng, theo các tiên đề xác suất, tổng của tất cả các xác suất đồng thời trong bảng phải bằng 1.

Từ bảng xác suất này, ta có thể tính giá trị mọi xác suất liên quan tới 3 biến của bài toán. Sau đây là ví dụ tính một số xác suất:

Xác suất một vật nào đó là "chim":

P(Chim = T) = 0 + 0,2 + 0,04 + 0.01 = 0,25.

Xác suất "chim không biết bay":

P(Chim = T, Bay = F) = 0,04 + 0,01 = 0,05.

Trường hợp tổng quát

Trong trường hợp tổng quát, xác suất của bất cứ tổ hợp giá trị E nào của một hoặc nhiều trong số n biến có thể tính từ bảng xác suất đồng thời của n biến đó như sau:

$$P(E) = \sum_{các\_dòng\_trùng\_với\_E} P(các\_dòng)$$

Có thể dễ dàng kiểm tra các ví dụ trên là những trường hợp riêng của công thức này. Chẳng hạn, ở ví dụ "chim không biết bay", E là (Chim = T, Bay = F) và $P(E)$ bằng tổng xác suất ở các dòng có Chim = T và Bay = F.

Bảng 4.1. Ví dụ bảng xác suất đồng thời cho ba biến nhị phân

Chim	Bay được	Non	Xác suất
T	T	T	0
T	T	F	0,2
T	F	T	0,04
T	F	F	0,01
F	T	T	0,01
F	T	F	0,01
F	F	T	0,23
F	F	F	0,5

4.3.3. Xác suất điều kiện

Xác suất điều kiện $P(A|B)$ là xác suất xảy ra sự kiện A khi biết sự kiện B và không biết thêm gì khác. Xác suất điều kiện được định nghĩa như sau:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Ví dụ, xác suất P(cúm|đau đầu) = 0.5 có nghĩa là nếu ta thấy một người đau đầu, và ngoài ra không có thêm thông tin gì khác thì xác suất người đó bị cúm là 50%.

Xác suất điều kiện đóng vai trò quan trọng trong suy diễn xác suất và có ý nghĩa như sau:

- Khi $P(A \mid B) = 1$: B đúng thì chắc chắn suy ra A
- Khi $P(A \mid B) = 0$: B đúng thì suy ra $\neg A$.
- Có thể xem B là bằng chứng hoặc quan sát và A là kết luận. Khi đó, $P(A \mid B)$ là xác suất hoặc niềm tin A là đúng khi quan sát thấy B.

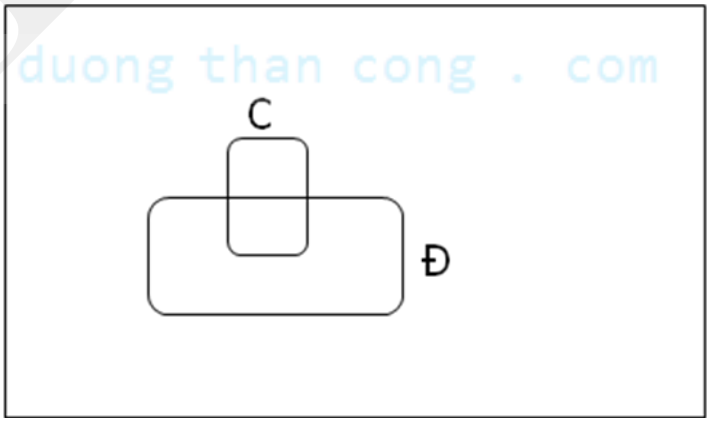
Như ta sẽ thấy ở dưới, quá trình suy diễn là quá trình tính xác suất điều kiện của kết luận khi biết bằng chứng. Cụ thể là khi cho biết các bằng chứng $E_1, \ldots, E_n$, suy diễn xác suất được thực hiện bằng cách tính xác suất điều kiện $P(Q \mid E_1, \ldots, E_n)$, tức là niềm tin kết luận Q đúng khi có các bằng chứng $E_1, \ldots, E_n$.

Để hiểu rõ hơn về ý nghĩa của xác suất điều kiện, ta hãy xét ví dụ với sơ đồ Venn được thể hiện trên hình vẽ dưới đây. Trong hình vẽ, hình chữ nhật bên ngoài thể hiện thế giới của bài toán, trong trường hợp này là toàn bộ người trên trái đất. Hình chữ nhật tròn C thể hiện tập hợp những người bị bệnh cúm và hình chữ nhật tròn Đ là tập hợp những người bị đau đầu. P(C) là xác suất một người bị cúm và tính bằng tỷ lệ hình chữ nhật C so với hình chữ nhật to. Tương tự P(Đ) là xác suất một người bị đau đầu. Phần giao của C và Đ là những người vừa bị cúm vừa đau đầu, hay C ∧ Đ. Theo hình vẽ, ta có:

P(C) = 1/40; P(Đ) = 1/10; P(C ∧ Đ) = 1/80

Theo hình vẽ, cả cúm và đau đầu đều có xác suất không lớn. Tuy nhiên, nếu ta biết một người bị cúm, tức là nằm trong hình chữ nhật C thì sẽ có 1/2 xác suất người đó nằm trong hình chữ nhật Đ, tức là 50% xác suất người đó bị đau đầu. Nói cách khác, ta có:

P(Đ|C) = 1/2 = P(C ∧ Đ)/P(C)



Các tính chất của xác suất điều kiện

- Qui tắc nhân

$$P(A, B) = P(A \mid B) * P(B)$$

- Qui tắc chuỗi (là mở rộng của quy tắc nhân)

$$P(A, B, C, D) = P(A \mid B, C, D) * P(B \mid C, D) * P(C \mid D) * P(D)$$

- Quy tắc chuỗi có điều kiện

$$P(A, B \mid C) = P(A \mid B, C) * P(B \mid C)$$

- Cộng

$$P(A) = \sum_b P(A|B = b) * P(B = b)$$

quy tắc cộng có thể chứng minh dễ dàng từ định nghĩa xác suất của xác suất điều kiện và tính chất xác suất biến đa trị đã trình bầy ở phần trên.

- $P(\neg B \mid A) = 1 - P(B \mid A)$
- Quy tắc Bayes

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

ta sẽ xem xét kỹ hơn quy tắc Bayes trong một phần sau.

Tính xác suất điều kiện từ bảng xác suất đồng thời

Trong trường hợp có đầy đủ xác suất đồng thời như đã xem xét ở trên, ta có thể thực hiện suy diễn thông qua tính xác suất điều kiện. Sau đây là một số ví dụ minh họa sử dụng bảng xác suất đồng thời trong phần trên với ký hiệu "Chim" là C, "Bay" là B và "Non" là N

Ví dụ 1. Giả sử các bằng chứng cho thấy một sinh vật biết bay, cần tính xác suất sinh vật đó là chim. Ta có:

$$P(\text{Chim} \mid \text{Bay}) = P(C \mid B)$$
$$= P(C, B) / P(B) \qquad \text{(theo định nghĩa)}$$
$$= \frac{P(C, B, N) + P(C, B, \sim N)}{P(C, B, N) + P(\sim C, B, N) + P(C, \sim B, N) + P(\sim C, \sim B, N)}$$
$$= (0 + 0{,}2) / (0 + 0{,}04 + 0{,}01 + 0{,}23)$$

Ví dụ 2. Ví dụ này minh họa cho việc kết hợp nhiều bằng chứng. Giả sử có bằng chứng sinh vật biết bay và không còn non, cần tính xác suất đây không phải là Chim. Ta có:

$$P(\neg\text{Chim} \mid \text{Bay}, \neg\text{Non}) = \frac{P(\sim C, B, \sim N)}{P(C, B, \sim N) + P(\sim C, B, \sim N)} = \frac{0{,}01}{0{,}01 + 0{,}2} = 0{,}048$$

Trong trường hợp tổng quát, khi cho bảng xác suất đồng thời của n biến V1, …, Vn, ta có thể tính xác suất của một số biến này khi biết giá trị một số biến khác như sau:

➔ **Công thức tổng quát**

$$P(V_1 = v_1, \ \ldots V_k = v_k \mid V_{k+i} = v_{k+i}, \ \ldots V_n = v_n)$$
$$= \frac{\text{tổng các dòng có } V_1 = v_1, \ldots, V_K = v_k, V_{k+i} = v_{k+i}, \ \ldots V_n = v_n}{\text{tổng các dòng có } V_{k+i} = v_{k+i}, \ \ldots V_n = v_n}$$

Một cách hình thức hơn, gọi các biến cần tính xác suất là Q, các biến đã biết là E, các biến còn lại (ngoài E và Q) là Y, ta có

$$P(Q \mid E) = \frac{\sum_{Y} P(Q, E, Y)}{\sum_{Q,Y} P(Q, E, Y)}$$

Mặc dù công thức trên tương đối đơn giản nhưng trên thực tế, khi số lượng biến tăng lên, số lượng các dòng chứa tổ hợp giá trị các biến ở tử số và mẫu số của công thức trên sẽ tăng theo hàm mũ, dẫn tới kích thước bảng xác suất đồng thời quá lớn nên không thể sử dụng cho suy diễn xác suất được. Trên thực tế, việc suy diễn với bảng xác suất đồng thời chỉ có thể thực hiện được với những bài toán có dưới 10 biến ngẫu nhiên nhị phân.

4.3.4. Tính độc lập xác suất

Tính độc lập xác suất là một trong những tính chất quan trọng, cho phép giảm số lượng xác suất cần biết khi xây dựng bảng xác suất đồng thời.

Tính độc lập xác suất được định nghĩa như sau: sự kiện A độc lập về xác suất với sự kiện B nếu:

$$P(A \mid B) = P(A)$$

Tức là biết giá trị của B không cho ta thêm thông tin gì về A.

Ví dụ, giả sử A là sự kiện một sinh viên thi môn trí tuệ nhân tạo được 10 điểm, và B là sự kiện năm nay là năm nhuận. Việc biết năm nay là năm nhuận hay không rõ ràng không làm tăng thêm xác suất sinh viên được 10 điểm trí tuệ nhân tạo và do vậy ta có $P(A \mid B) = P(A)$, tức là A độc lập xác suất với B.

Tương đương với công thức trên, khi A độc lập với B, ta có:

$$P(A, B) = P(A)\, P(B)$$

và

$$P(B \mid A) = P(B)$$

(người đọc có thể tự chứng minh hai công thức này bằng cách sử dụng định nghĩa xác suất điều kiện).

Công thức cuối cho thấy sự độc lập xác suất có tính đối xứng: nếu A độc lập xác suất với B thì ngược lại ta có B độc lập xác suất với A. Từ hai công thức trên cũng có thể nhận thấy từ hai xác suất $P(A)$ và $P(B)$, ta có thể tính được toàn bộ xác suất đồng thời của hai biến ngẫu nhiên A và B như sau:

$$P(A, B) = P(A)\, P(B)$$

$$P(A, \neg B) = P(A)\, P(\neg B) = P(A)\,(1 - P(B))$$

$$P(\neg A, B) = P(\neg A)\, P(B) = (1 - P(A))\, P(B)$$

$$P(\neg A, \neg B) = P(\neg A)\, P(\neg B) = (1 - P(A))\,(1 - P(B))$$

Như vậy, thay vì cần biết 4 xác suất đồng thời, nay ta chỉ cần biết 2 xác suất. Nhận xét này cho thấy tính độc lập xác suất cho phép biểu diễn rút gọn bảng xác suất đồng thời, làm cơ sở cho việc suy diễn trong những bài toán suy diễn xác suất với nhiều biến ngẫu nhiên.

Độc lập xác suất có điều kiện

Ở trên ta đã xét trường hợp độc lập xác suất không điều kiện. Mở rộng cho trường hợp xác suất điều kiện, ta nói rằng A độc lập có điều kiện với B khi biết C nếu:

$$P(A \mid B, C) = P(A \mid C)$$

hoặc $P(B \mid A, C) = P(B \mid C)$

khi đó ta có:

$$P(A, B \mid C) = P(A \mid C)\, P(B \mid C)$$

Ý nghĩa: nếu đã biết giá trị của C thì việc biết giá trị của B không cho ta thêm thông tin về A.

Ví dụ: giả sử ký hiệu A là sự kiện sinh viên đi học muộn, B là sự kiện sinh viên mặc áo mưa, C là trời mưa. Thông thường, nếu quan sát thấy sinh viên mặc áo mưa (B = true) thì xác suất có sinh viên đi học muộn sẽ tăng lên (do đường bị ngập, phương tiện di chuyển chậm hơn), tức là P(A|B) ≠ P(A). Tuy nhiên, nếu ta đã biết trời mưa (C = true) và biết xác suất đi học muộn tăng lên, thì việc nhìn thấy sinh viên mặc áo mưa không làm tăng tiếp đổi xác suất sinh viên đi học muộn nữa. Tức là $P(A \mid B, C) = P(A \mid C)$.

4.3.5. Quy tắc Bayes

Như đã nhắc tới ở trên, quy tắc Bayes có dạng sau:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Quy tắc Bayes đóng vai trò quan trọng trong suy diễn xác suất. Thông thường, bài toán suy diễn đòi hỏi tính $P(A \mid B)$, tuy nhiên trong nhiều trường hợp, việc tính $P(B \mid A)$ có thể dễ dàng hơn. Khi đó, quy tắc Bayes cho phép ta quy việc tính $P(A \mid B)$ về tính $P(B \mid A)$.

Ví dụ: Giả sử cần tính xác suất một người bị cúm khi biết người đó đau đầu, tức là tính

P (cúm | đau đầu)

Để tính xác suất này, cần xác định có bao nhiêu người bị đau đầu trong một cộng đồng dân cư, sau đó đếm xem có bao nhiêu người trong số người đau đầu bị cúm. Rõ ràng việc thống kê những người đau đầu tương đối khó khăn do không phải ai đau đầu cũng thông báo cho cơ sở y tế. Ngược lại, để tính P(đau đầu | cúm), ta cần đếm số người bị cúm sau đó xác định xem trong số này bao nhiêu người có triệu chứng đau đầu. Việc này được thực hiện tương đối dễ dàng hơn, chẳng hạn bằng cách phân tích bệnh án những người bị cúm.

Sau đây, ta xem xét một ví dụ sử dụng quy tắc Bayes cho suy diễn.

Ví dụ: Một người làm xét nghiệm về một căn bệnh hiếm gặp (ký hiệu HG) và nhận được kết quả dương tính. Biết rằng thiết bị xét nghiệm không chính xác hoàn toàn. Cụ thể, thiết bị cho kết quả dương tính đối với 98% người có bệnh. Trong trường hợp người không có bệnh vẫn có 3% khả năng xét nghiệm cho kết quả dương tính. Ngoài ra, ta biết rằng xác suất quan sát thấy bệnh này trong cộng đồng là 8 phần nghìn. Yêu cầu đặt ra là cần xác định xem người xét nghiệm với kết quả dương tính có bệnh không.

Ký hiệu sự kiện có bệnh là B, xét nghiệm dương tính là ⊕, như vậy kết quả âm tính sẽ là ¬⊕. Để kết luận người khám có bị bệnh không ta cần so sánh xác suất $P(B \mid \oplus)$ và $P(\neg B \mid \oplus)$.

Theo dữ kiện đã cho, ta có:

$P(B) = 0.008 \qquad P(\neg B) = 0.992$

$P(\oplus \mid B) = 0.98 \quad P(\neg \oplus \mid B) = 1\text{-}0.98 = 0.02$

$P(\neg\oplus \mid \neg B) = 0.97 \; P(\oplus \mid \neg B) = 0.03$

Sử dụng quy tắc Bayes, ta có:

$$P(B \mid \oplus) = \frac{P(\oplus \mid B)P(B)}{P(\oplus)} = \frac{0.98 * 0.008}{P(\oplus)} = \frac{0.00784}{P(\oplus)}$$

$$P(\neg B \mid \oplus) = \frac{P(\oplus \mid \sim B)P(\sim B)}{P(\oplus)} = \frac{0.03 * 0.992}{P(\oplus)} = \frac{0.02976}{P(\oplus)}$$

Như vậy, $P(\neg B \mid \oplus) > P(B \mid \oplus)$ và do đó ta kết luận người tới khám không bị bệnh. Sở dĩ xác suất không bệnh lớn hơn là do xác suất $P(B)$, còn gọi là xác suất tiền nghiệm, của bệnh rất nhỏ. Trong trường hợp như vậy, để khẳng định bị bệnh cần phải có khả năng xét nghiệm với đội chính xác rất cao.

Chuẩn tắc hóa (normalization)

Cần lưu ý thêm rằng, để so sánh $P(\neg B \mid \oplus)$ và $P(B \mid \oplus)$ ta không cần tính giá trị đúng của hai xác suất này do hai biểu thức có chung mẫu số $P(\oplus)$. Thay vào đó ta có thể tính

$$\frac{P(B \mid \oplus)}{P(\sim B \mid \oplus)} = \frac{0.00784}{0.02976}$$

và kết luận có bệnh hay không bệnh tùy vào giá trị trên lớn hơn hay nhỏ hơn 1 (hoặc một giá trị ngưỡng nào đó).

Tuy nhiên, trong một số trường hợp, ta có thể cần tính giá trị đúng của hai xác suất đó, thay vì so sánh với nhau. Để tính các xác suất này, ta có thể làm như sau:

do $\quad P(\neg B \mid \oplus) + P(B \mid \oplus) = 1$

nên

$$\frac{P(\oplus \mid \sim B)P(\sim B)}{P(\oplus)} + \frac{P(\oplus \mid B)P(B)}{P(\oplus)} = 1$$

Từ đây suy ra

$$P(\oplus) = P(\oplus \mid B)P(B) + P(\oplus \mid \sim B)P(\sim B)$$

hay: $\quad P(\oplus) = 0.00784 + 0.02976$

Thay vào các biểu thức trên, ta có $P(\neg B \mid \oplus)=0.79$ và $P(B \mid \oplus) = 0.21$

Quá trình tính toán như trên được gọi là chuẩn tắc hóa do ta đã nhân các giá trị $1/ P(\oplus)$ để chuẩn tắc hóa các xác suất điều kiện sao cho tổng các xác suất này bằng 1.

Kết hợp quy tắc Bayes với tính độc lập xác suất

Quy tắc Bayes cũng có thể kết hợp với tính độc lập về xác suất. Ví dụ tiếp theo sẽ minh họa cho việc kết hợp này.

Ví dụ 1: Tính $P(A| B, C)$, cho biết B độc lập với C nếu biết A:

$$P(A|B, C) = \frac{P(B,C| A) * P(A)}{P(B,C)} = \frac{P(B|A) * P(C|A) * P(A)}{P(B,C)}$$

Ví dụ 2: Cho 3 biến nhị phân: gan BG, vàng da VD, thiếu máu TM.

- Giả sử VD độc lập với TM
- Biết P(BG) = 10^{-17}
- Có người khám bị VD
- Biết P(VD) = 2^{-10} và P(VD|BG) = 2^{-3}

a. Xác suất người khám bị bệnh là bao nhiêu?

b. Cho biết thêm người đó bị thiếu máu và P(TM) = 2^{-6}, P(TM|BG) = 2^{-1}. Hãy tính xác suất người khám bị bệnh BG.

Giải:

a. $P(BG|VD) = \frac{P(VD|BG) * P(BG)}{P(VD)} = \frac{2^{-3} * 10^{-17}}{2^{-10}}$

b. Ta cần tính xác suất điều kiện

$$P(BG \mid VD, TM) = \frac{P(VD|BG) * P(TM|BG) * P(BG)}{P(TM,VD)}$$

Mà TM và VD độc lập nên ta có:

$$P(BG \mid VD, TM) = \frac{2^{-3} * (1 - 2^{-1}) * 10^{-17}}{(1 - 2^{-6}) * 2^{-10}}$$

Ví dụ trên cho thấy, việc sử dụng tính độc lập (có điều kiện) giữa các biến cho phép rút gọn đáng kể quá trình tính toán xác suất điều kiện khi thực hiện suy diễn.

4.4. MẠNG BAYES

Trong các phần trước ta đã làm quen với vấn đề suy diễn xác suất, trong đó cho trước một số bằng chứng $E_1, \ldots, E_n$, cần tính xác suất điều kiện $P(Q \mid E_1, \ldots, E_n)$ để kết luận về câu truy vấn Q.

Xác suất điều kiện trên có thể tính được nếu biết toàn bộ xác suất đồng thời của các biến ngẫu nhiên. Tuy nhiên, trên thực tế, các bài toán thường có số lượng biến ngẫu nhiên lớn, dẫn tới số lượng xác suất đồng thời tăng theo hàm mũ. Do vậy, liệt kê và sử dụng bảng xác suất đồng thời đầy đủ để suy diễn là không thực tế.

Để khắc phục khó khăn trên, trong phần này ta sẽ xem xét cách sử dụng mạng Bayes như một mô hình biểu diễn xác suất rút gọn và cách thực hiện suy diễn xác suất trên mạng Bayes.

4.4.1. Khái niệm mạng Bayes

Để tiện cho việc trình bày khái niệm mạng Bayes, xét một ví dụ sau[3].

Một người đi làm về và muốn dự đoán xem ở nhà có người không thông qua một số dấu hiệu có thể quan sát được. Cho biết một số dữ kiện sau:

- *Nếu cả nhà đi vắng thì thường bật đèn ngoài sân. Tuy nhiên, đèn ngoài sân có thể được cả trong một số trường hợp có người ở nhà, ví dụ khi có khách đến chơi.*
- *Nếu cả nhà đi vắng thì thường buộc chó ở sân sau.*

[3] Đây là một ví dụ được sử dụng trong bài báo "Bayesian networks without tears" đăng trên tạp chí AI Magazine năm 1991.

- *Tuy nhiên chó có thể được buộc ở sân sau cả khi có người ở nhà nếu như chó bị đau bụng.*
- *Nếu chó buộc ở ngoài thì có thể nghe tiếng sủa, tuy nhiên có thể nghe tiếng sủa (của chó hàng xóm) cả khi chó không buộc ở ngoài.*

Để thực hiện suy diễn xác suất cho bài toán trên, trước tiên cần xây dựng mô hình xác suất. Ta sẽ sử dụng năm biến ngẫu nhiên sau để thể hiện các dữ kiện liên quan tới bài toán.

O: không ai ở nhà

L: đèn sáng

D: chó ở ngoài

B: chó bị ốm.

H: nghe thấy tiếng sủa.

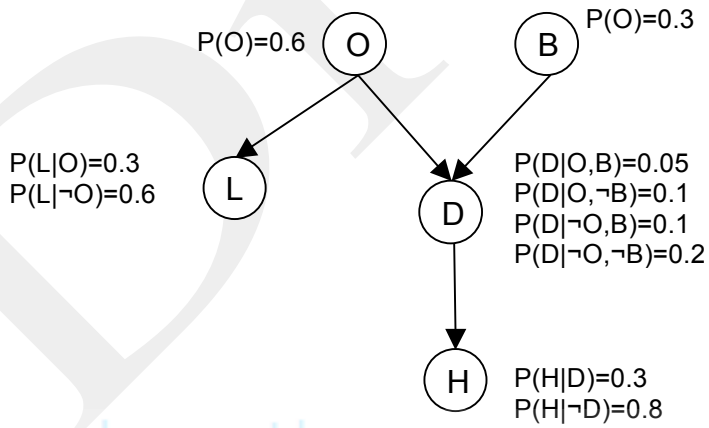
Việc phân tích bài toán cho thấy:

- Nếu biết D thì H không phụ thuộc vào O, L, B.

- Nếu biết B thì D độc lập với O.

- O và B độc lập với nhau

Tiếp theo, ta xây dựng một đồ thị, trong đó mỗi biến ngẫu nhiên ở trên được biểu diễn bởi một nút như trên hình vẽ dưới đây (hình 4.1). Các nút được nối với nhau bằng những cung có hướng sao cho hai hai nút có quan hệ phụ thuộc được nối bởi một cung và hướng của cung thể hiện chiều tác động của nút gốc tới nút đích. Với đồ thị có hướng, ta có thể xác định quan hệ giữa các nút như sau: nếu tồn tại cung có hướng từ nút A tới nút B thì nút A được gọi là nút cha (mẹ) và nút B là nút con.



Hình 4.1: Một ví dụ mạng Bayes

Sau khi có đồ thị, ta thêm vào *bảng xác suất điều kiện*. Bảng xác suất điều kiện thể hiện xác suất của biến khi biết giá trị cụ thể của các biến ở các nút cha mẹ. Trong trường hợp nút không có cha mẹ, xác suất trở thành xác suất tiền nghiệm. Để thuận tiện, bảng xác suất điều kiện được thể hiện ngay trên hình vẽ cùng với đồ thị.

Đồ thị vừa xây dựng cùng với các bảng xác suất điều kiện tạo thành mạng Bayes cho bài toán trong ví dụ trên.

Định nghĩa: Mạng Bayes là một mô hình xác suất bao gồm 2 phần

- Phần thứ nhất là một đồ thị có hướng không chứa chu trình, trong đó mỗi nút tương ứng với một biến ngẫu nhiên, các cung thể hiện mối quan hệ phụ thuộc giữa các biến.
- Phần thứ hai là các bảng xác suất điều kiện: mỗi nút có một bảng xác suất điều kiện cho biết xác suất các giá trị của biến khi biết giá trị các nút cha mẹ.

Cấu trúc của đồ thị trong mạng Bayes thể hiện mối quan hệ phụ thuộc hoặc độc lập giữa các biến ngẫu nhiên của bài toán. Hai nút được nối với nhau bởi một cung khi giữa hai nút có quan hệ trực tiếp với nhau, trong đó giá trị nút gốc ảnh hưởng tới giá trị nút đích.

Lưu ý rằng trong cấu trúc của mạng Bayes không cho phép có chu trình. Hạn chế này ảnh hưởng tới khả năng mô hình hóa của mạng Bayes trong một số trường hợp tuy nhiên cho phép đơn giản hóa việc xây dựng và suy diễn trên mạng Bayes.

Bảng xác suất điều kiện xác định cụ thể ảnh hưởng của các nút cha mẹ tới giá trị nút con. Ở đây ta chỉ xét trường hợp biến ngẫu nhiên có thể nhận giá trị rời rạc và bảng xác suất điều kiện được cho theo tổ hợp giá trị của các nút cha mẹ. Mỗi dòng trong bảng tương ứng với một điều kiện cụ thể, thực chất là một tổ hợp giá trị các nút cha. Ví dụ, trong mạng Bayes của ví dụ trên, dòng thứ nhất trong bảng xác suất của nút D ứng với điều kiện trong đó O = True và B = True. Nếu nút không có cha mẹ thì bảng xác suất chỉ gồm một dòng duy nhất như trường hợp với nút O và nút B.

4.4.2. Tính độc lập xác suất trong mạng Bayes

Mạng Bayes thể hiện hai thông tin chính.

Thứ nhất, đây là biểu diễn rút gọn của toàn bộ xác suất đồng thời. Trong ví dụ trên ta chỉ cần 10 xác suất thay vì 2^5-1 xác suất đồng thời. Tùy theo kích thước và đặc điểm cụ thể của bài toán, hiệu quả của việc rút gọn số lượng xác suất có thể lớn hơn rất nhiều. Chẳng hạn, với mạng gồm 30 nút nhị phân, mỗi nút có 5 nút cha, ta cần tất cả 960 xác suất điều kiện cho mạng Bayes, trong khi bảng xác suất đồng thời cho 30 biến như vậy phải có 2^{30}-1, tức là hơn một tỷ dòng.

Thứ hai, mạng Bayes cho thấy sự phụ thuộc hoặc độc lập xác suất có điều kiện giữa các biến. Về thực chất, chính việc độc lập về xác suất dẫn tới khả năng biểu diễn rút gọn các xác suất đồng thời.

Tính độc lập xác suất trong mạng Bayes thể hiện qua tính chất sau.

Tính chất:

- Mỗi nút trên mạng Bayes độc lập có điều kiện với tất cả các nút không phải là hậu duệ của nút đó nếu biết giá trị các nút cha.
- Mỗi nút độc lập có điều kiện với tất cả các nút khác trên mạng nếu biết giá trị tất cả nút cha, nút con và nút cha của các nút con.

Ví dụ: Theo mạng Bayes trong ví dụ trên H độc lập với O, L, B nếu biết giá trị của D.

Tính các xác suất đồng thời

Sử dụng tính độc lập xác suất vừa phát biểu ở trên, có thể tính xác suất đồng thời của tất cả các biến. Xét ví dụ sau

Ví dụ :cần tính $P(H, D, L, \neg O, B)$

Theo công thức chuỗi:

$P(H, D, L, \neg O, B) = P(H| D, L, \neg O, B) * P(D| L, \neg O, B) * P(L|\neg O, B) * P(\neg O|B) * P(B)$

Do tính độc lập xác suất (có điều kiện):

$P(H| B, D, \neg O, L) = P(H|D)$

$P(D| L, \neg O, B) = P(D|\neg O, B)$

$P(L| \neg O, B) = P(L|\neg O)$

$P(\neg O|B)=P(O)$

do vậy,

$P(H, D, L, \neg O, B) = P(H| D, L, \neg O, B) * P(D| L, \neg O, B) * P(L|\neg O, B) * P(\neg O|B) * P(B)$

$= P(H|\neg O) * P(D|\neg O, B) * P(L|\neg O) * P(\neg O) * P(B)$

Một cách tổng quát, giả sử mạng có n nút tương ứng với n biến ngẫu nhiên $X_1, \ldots, X_n$ của bài toán đang xét. Từ thông tin của mạng, có thể tính mọi xác suất đồng thời của n biến, trong đó mỗi xác suất đồng thời có dạng $P(X_1 = x_1 \wedge X_2 = x_2 \wedge \ldots \wedge X_n = x_n)$ hay viết gọn là $P(x_1, \ldots, x_n)$. Xác suất đồng thời được tính theo công thức tổng quát sau:

$$P(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i \mid cha\_me(X_i))$$

hay viết gọn là

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid cha\_me(X_i))$$

trong đó $cha\_me(X_i)$ là giá trị cụ thể các nút cha mẹ của nút X_i.

Để minh họa cho công thức trên, ta sẽ tính xác suất xảy ra tình huống ở nhà có người, chó bị ốm và được buộc ngoài sân, đồng thời đèn không sáng và nghe tiếng chó sủa. Xác suất tình huống này chính là $P(B, \neg O, D, \neg L, H)$ và được tính như sau:

$$P(B, \neg O, D, \neg L, H) = P(B) * P(\neg O) * P(D|\neg O, B) * P(H|D) * P(\neg L|\neg O)$$
$$= 0{,}3 * 0{,}4 * 0{,}05 * 0{,}7 * 0{,}3$$
$$= 0{,}00126$$

Trong một phần trên ta đã thấy rằng nếu có mọi xác suất đồng thời thì có thể thực hiện suy diễn xác suất cho mọi dạng câu truy vấn. Như vậy, với mạng Bayes ta có thể suy diễn bằng cách trước tiên tính ra mọi xác suất đồng thời cần thiết. Tuy nhiên, cách này đòi hỏi tính toán nhiều và vì vậy trên thực tế thường sử dụng một số phương pháp suy diễn khác hiệu quả hơn. Vấn đề này sẽ được nhắc tới trong một phần sau.

4.4.3. Cách xây dựng mạng Bayes

Để có thể sử dụng, trước tiên cần xây dựng ra mạng Bayes. Quá trình xây dựng mạng Bayes bao gồm việc xác định tất cả các biến ngẫu nhiên liên quan, xác định cấu trúc đồ thị của mạng, và cuối cùng là xác định giá trị cho các bảng xác suất điều kiện. Trong phần này, ta sẽ coi như đã có biến ngẫu nhiên, việc xây dựng mạng chỉ bao gồm xác định cấu trúc và bảng xác suất điều kiện.

Có hai cách tiếp cận chính để xây dựng mạng Bayes.

- Cách thứ nhất do con người (chuyên gia) thực hiện dựa trên hiểu biết của mình về bài toán đang xét. Việc xây dựng mạng được chia thành hai bước: xác định cấu trúc đồ thị và điền giá trị cho bảng xác suất điều kiện.
- Cách thứ hai là tự động xác định cấu trúc và xác suất điều kiện từ dữ liệu. Ở đây, dữ liệu có dạng giá trị các biến ghi nhận được trong quá khứ, ví dụ ta có thể ghi lại tổ hợp cá giá trị của năm biến trong ví dụ trên trong thời gian vài năm. Quá trình xây dựng mạng khi đó bao gồm xác định cấu trúc của đồ thị và bảng xác suất điều kiện sao cho phân bố xác suất do mạng thể hiện phù hợp nhất với tần suất xuất hiện các giá trị trong tập dữ liệu.

Phần này chỉ xem xét cách xây dựng mạng do con người thực hiện và mô tả một quy trình cụ thể cho việc xây dựng mạng.

Các bước xây dựng mạng được thực hiện như trên hình 4.2. Sau khi đã có cấu trúc mạng, chuyên gia sẽ xác định giá trị cho các bảng xác suất điều kiện. Thông thường, việc xác định giá trị xác suất điều kiện khó hơn nhiều so với việc xác định cấu trúc mạng, tức là xác định quan hệ giữa các nút.

B1: Xác định các biến ngẫu nhiên cho phép mô tả miền của bài toán.

B2: Sắp xếp các biến theo một thứ tự nào đó. Ví dụ theo thứ tự sau: X_1, X_2 … X_n.

B3: *For i = 1 to n do*

a. *Thêm một nút mới X_i vào mạng*

b. *Xác định tập Cha_Mẹ(X_i) là tập nhỏ nhất các nút đã có trước đó sao cho X_i độc lập có điều kiện với tất cả nút còn lại khi biết bố mẹ của X_i.*

c. *Với mỗi nút thuộc tập Cha_Mẹ(X_i). Ta thêm một cạnh có hướng từ nút đó tới X_i.*

d. *Xác định bảng xác suất điều kiện cho X_i the các giá trị của bố mẹ hoặc bằng xác suất tiền nghiệm nếu X_i không có bố mẹ.*

Hình 4.2: Phương pháp xây dựng mạng Bayes

Để minh họa, xét ví dụ sau. Một người vừa lắp hệ thống báo động chống trộm ở nhà. Hệ thống sẽ phát tiếng động khi có trộm. Tuy nhiên, hệ thống có thể báo động (sai) nếu có chấn động do động đất. Trong trường hợp nghe thấy hệ thống báo động, hai người hàng xóm tên làm Nam và Việt sẽ gọi điện cho chủ nhà. Do nhiều nguyên nhân khác nhau, Nam và Việt có thể thông báo sai, chẳng hạn do ồn nên không nghe thấy chuông báo động hoặc ngược lại, nhầm âm thanh khác là tiếng chuông.

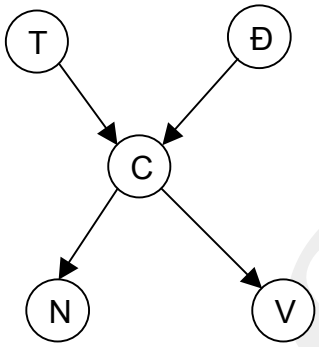
Theo phương pháp trên, các bước xây dựng mạng được thực hiện như sau.

B1: lựa chọn biến: sử dụng 5 biến sau

T (có trộm), Đ (động đất), B (chuông báo động), N (Nam gọi điện), V (Việt gọi điện)

B2: các biến được sắp xếp theo thứ tự T, Đ, B, N, V

B3: thực hiện như các bước ở hình vẽ, ta xây dựng được mạng thể hiện trên hình sau (để đơn giản, trên hình vẽ chỉ thể hiện cấu trúc và không có bảng xác suất điều kiện).



Hình 4.3.: Kết quả xây dựng mạng Bayes cho ví dụ chuông báo trộm

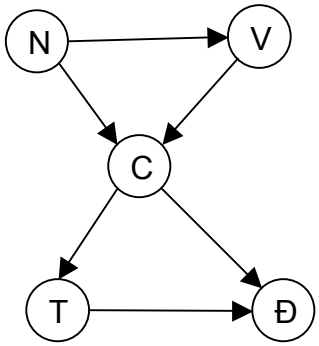
Ảnh hưởng của việc sắp xếp các nút tới kết quả xây dựng mạng.

Trên thực tế, việc xây dựng mạng Bayes không đơn giản, đặc biệt trong việc chọn thứ tự các nút đúng để từ đây chọn được tập nút cha có kích thước nhỏ. Để làm rõ điểm này, ta giả sử trong ví dụ trên, các biến được xếp theo thứ tự khác: N, V, C, T, Đ.

Các bước thêm nút sẽ thực hiện như sau:

- Thêm nút N: không có nút cha
- Thêm nút V: nếu Nam gọi điện, xác suất Việt gọi điện sẽ tăng lên do sự kiện Nam gọi điện nhiều khả năng do có báo động và do vậy xác suất Việt nghe thấy chuông và gọi điện tăng theo. Do vậy N có ảnh hưởng tới V và được thêm vào tập cha của V.
- Thêm C: Nếu Nam và Việt cùng gọi thì khả năng có chuông cao hơn, do vậy cần thêm cả N và V vào tập cha của C.
- Thêm T: Nếu đã biết trạng thái của chuông thì không cần quan tâm tới Nam và Việt nữa, do vậy chỉ có C là cha của T.
- Thêm Đ: nếu có chuông, khả năng động đất tăng lên. Tuy nhiên, nếu đồng thời ta biết có trộm thì việc có trộm giải thích phần nào nguyên nhân chuông kêu. Như vậy, cả chuông và có trộm ảnh hưởng tới xác suất động đất, tức là C và T đều là cha của Đ.

Kết quả của mạng Bayes xây dựng theo thứ tự mới được thể hiện trên hình dưới. So sánh với kết quả ở trên, mạng Bayes mới phức tạp hơn, theo nghĩa có nhiều cung hơn hay trung bình các nút có nhiều nút cha hơn. Ngoài ra, ý nghĩa một số quan hệ trên mạng rất không trực quan và khó giải thích, chẳng hạn việc xác suất động đất phục thuộc vào chuông báo động và có trộm. Như vậy, mặc dù cả hai mạng Bayes xây dựng ở trên đều đúng theo nghĩa đảm bảo các ràng buộc về xác suất và đều cho phép tính ra các xác suất đồng thời, việc lựa chọn không đúng thứ tự nút sẽ làm mạng khó hiểu và phức tạp hơn.



Hình 4.4: Kết quả xây dựng mạng Bayes khi sử dụng thứ tự các nút khác

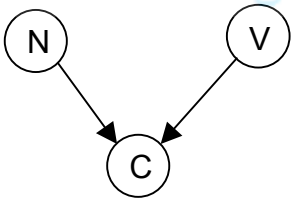
Từ ví dụ trên ta có thể đưa ra một số nhận xét về kết quả xây dựng mạng Bayes.

Nhận xét:

- Cùng một tập hợp biến có thể xây dựng nhiều mạng Bayes khác nhau.
- Thứ tự sắp xếp có ảnh hưởng tới mạng Bayes. Nên sắp xếp sao cho các nút đóng vai trò nguyên nhân đứng trước nút hệ quả.
- Tất cả các mạng được xây dưng như trên đều hợp lệ, theo nghĩa không vi phạm các ràng buộc về xác suất và đều cho phép thực hiện suy diễn.

4.4.4. Tính độc lập xác suất tổng quát: khái niệm *d*-phân cách

Trong phần 4.4.2, ta đã xem xét khả năng biểu diễn tính độc lập xác suất của mạng Bayes, ví dụ, mỗi x nút độc lập với các nút không phải hậu duệ nếu biết giá trị tất cả nút cha của x. Tuy nhiên, đây mới là các trường hợp riêng, trong trường hợp tổng quát cần có khả năng xác định liệu một tập hợp các nút X có độc lập với tập hợp các nút Y khi biết các nút E không. Các tính chất độc lập xác suất đã trình bầy trong phần 4.4.2 không cho phép trả lời tất cả các câu hỏi tổng quát dạng này. Chẳng hạn, trong ví dụ mạng Bayes trên hình 4.5 dưới đây, nếu không biết giá trị của nút C thì theo tính chất của mạng Bayes, N và V độc lập (không điều kiện) với nhau do V không phải hậu duệ của N và N không có cha. Tuy nhiên, nếu đã biết giá trị của C thì N và V còn độc lập với nhau không? Hai tính chất trình bầy trong phần 4.4.2 không cho phép trả lời câu hỏi này.



Hình 4.5. Ví dụ mạng Bayes

Trong phần này, ta sẽ xem xét cách trả lời câu hỏi về tính độc lập của tập các nút X với tập nút Y khi biết tập nút E trên một mạng Bayes bằng cách sử dụng khái niệm *d-phân cách* (*d*-separation).

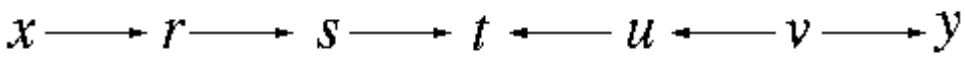
Nguyên lý chung của *d*-phân cách là gắn khái niệm phụ thuộc xác suất với tính kết nối (tức là có đường đi giữa các nút), và khái niệm độc lập xác suất với tính không kết nối, hay chia cắt, trên đồ thị có hướng khi ta biết giá trị một số nút *E*. Chữ "*d* " ở đây là viết tắt của từ "directional" tức là "có hướng". Theo đó, các nút *X* và các nút *Y* là *d*-kết nối với nhau nếu chúng không bị *d*-phân cách. *Nếu các nút X và các nút Y bị d-phân cách bởi các nút E thì X và Y là độc lập xác suất với nhau khi biết E*.

Để xác định tính *d*-phân cách của tập X và Y, trước tiên ta cần xác định tính *d*-phân cách giữa hai nút đơn *x* thuộc *X* và *y* thuộc *Y*. Từ đây, hai tập nút sẽ độc lập với nhau nếu mỗi nút trong tập này độc lập với tất cả các nút trong tập kia. Sau đây là các quy tắc cho phép xác định tính *d*-phân cách hay tính độc lập xác suất của hai biến *x* và *y*.

Quy tắc 1: nút *x* và *y* được gọi là *d*-kết nối nếu tồn tại đường đi không bị phong tỏa giữa hai nút. Ngược lại, nếu không tồn tại đường đi như vậy thì *x* và *y* là *d*-phân cách.

Trong quy tắc này, đường đi là một chuỗi các cung nằm liền nhau, không tính tới hướng của các cung đó. Đường đi không bị phong tỏa là đường đi mà trên đó không có hai cung liền kề hướng vào nhau. Trong trường hợp tồn tại hai cung như vậy thì thông tin sẽ không thể đi qua được và do vậy các nút không thể kết nối với nhau. Nút có hai cung hướng vào như vậy gọi là *nút xung đột*.

Ví dụ, trong trường hợp sau:



giữa *x* và *y* tồn tại đường đi *x* - *r* - *s* - *t* - *u* - *v* - *y*, tuy nhiên *t* là nút xung đột do hai cung *st* và *ut* hướng vào nhau. Đường đi *x* - *r* - *s* - *t* và *t* - *u* - *v* - *y* là các đường đi không bị phong tỏa, do vậy *x* và *t* là d-kết nối, *t* và *y* cũng vậy. Tuy vậy, *x* và *y* không phải là *d*-kết nối do không tồn tại đường đi nào không qua nút xung đột *t*. Như vậy, *x* và *y* là *d*-phân cách trên mạng này và do vậy độc lập xác suất (không điều kiện) với nhau.

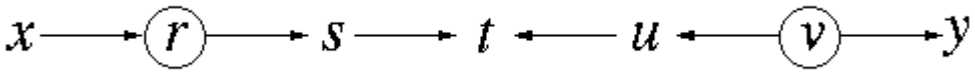
Tính kết nối và phân cách xác định theo quy tắc 1 là không điều kiện và do vậy tính độc lập xác suất được xác định theo quy tắc 1 là *độc lập không điều kiện*.

Quy tắc 2: nút *x* và *y* là *d*-kết nối có điều kiện khi biết tập nút *E* nếu tồn tại đường đi không bị phong tỏa (không chứa nút xung đột) và không đi qua bất cứ nút nào thuộc *E*. Ngược lại, nếu không tồn tại đường đi như vậy thì ta nói rằng *x* và *y* là *d*-phân cách bởi *E*. Nói cách khác, mọi đường đi giữa *x* và *y* (nếu có) đều bị *E* phong tỏa.

Quy tắc 2 là cần thiết do khi ta biết giá trị một số nút (tập nút *E*), tính chất độc lập hay phụ thuộc giữa các nút còn lại có thể thay đổi: một số nút độc lập trở nên phụ thuộc, và ngược lại, một ss nút phụ thuộc trở thành độc lập. Tính độc lập hay phụ thuộc trong trường hợp này được gọi là *d*-phân cách có điều kiện theo tập biến *E*.

Ví dụ: trên hình sau, giả sử tập *E* gồm hai nút *r* và *v* được khoanh tròn. Theo quy tắc 2, không tồn tại đường đi không bị phong tỏa nào giữa *x* và *y* mà không đi qua *E*, do đó *x* và *y* là *d*-phân cách khi biết *E*. Tương tự như vậy: *x* và *s*, *u* và *y*, *s* và *u* là *d*-phân cách khi biết *E* do

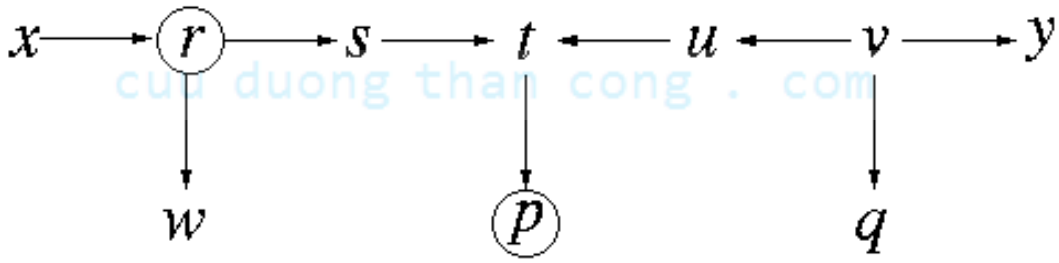
đường đi $s - r - t$ đi qua nút r thuộc E, đường đi $y - v - s$ đi qua nút v thuộc E, còn đường đi $s - t - u$ là đường đi bị phong tỏa tại nút xung đột t theo quy tắc 1. Chỉ có các cặp nút s và t, t và u là không bị phong tỏa bởi E.



Quy tắc 3: nếu một nút xung đột là thành viên của tập E, hoặc có hậu duệ thuộc tập E, thì nút đó không còn phong tỏa các đường đi qua nó nữa.

Quy tắc này được sử dụng cho trường hợp ta biết một sự kiện được gây ra bởi hai hay nhiều nguyên nhân. Khi ta đã biết một nguyên nhân là đúng thì xác suất những nguyên nhân còn lại giảm đi, và ngược lại nếu ta biết một nguyên nhân là sai thì xác suất những nguyên nhân còn lại tăng lên. Chẳng hạn, xẩy ra tai nạn máy bay với hai nguyên nhân là trục trặc kỹ thuật hoặc lỗi của con người. Nếu ta đã xác định được xẩy ra trục trặc kỹ thuật thì xác suất lỗi con người sẽ bị giảm đi (mặc dù không loại trừ hoàn toàn).

Ví dụ: trên ví dụ ở hình sau, giả sử tập E gồm các nút r và p được đánh dấu bằng cách khoanh tròn. Theo quy tắc 3, nút s và y là d-kết nối do nút xung đột t có hậu duệ là nút p thuộc E, do vậy đã giải tỏa đường đi s – t – u – v – y. Trong khi đó x và u vẫn là d-phân cách do mặc dù t đã được giải tỏa nhưng nút r vẫn bị phong tỏa theo quy tắc 2.



4.5. SUY DIỄN VỚI MẠNG BAYES

Sau khi đã xem xét khái niệm mạng Bayes và khả năng biểu diễn ngắn gọn quan hệ độc lập xác suất, cũng như khả năng tính xác suất đồng thời trên mạng, trong phần này ta sẽ xem xét cách thực hiện suy diễn xác suất trên mạng. Sau khi nhắc lại khái niệm suy diễn xác suất, ta sẽ xem xét một số kỹ thuật suy diễn cho những trường hợp đơn giản. Các kỹ thuật phức tạp hơn sẽ được trình bầy ở cuối chương. Các kỹ thuật suy diễn trình bầy trong phần này đều dựa trên việc tận dụng các cấu trúc của mạng Bayes để đơn giản hoá quá trình tính toán.

Suy diễn xác suất là quá trình tìm xác suất hậu nghiệm hay xác suất điều kiện của một số biến khi biết giá trị một số biến khác. Thông thường, suy diễn xác suất được thực hiện bằng cách tính

$$P(Q \mid E = e)$$

trong đó:

Q là danh sách các biến truy vấn, thường chỉ gồm một biến;

E là danh sách các biến quan sát được, còn gọi là các bằng chứng;

e là danh sách các giá trị cụ thể của các biến thuộc tập E.

Sử dụng định nghĩa xác suất điều kiện, giá trị trên có thể viết lại như sau:

$$P(Q \mid E = e) = \frac{P(Q, E = e)}{P(E = e)}$$

Chẳng hạn, với mạng Bayes cho ở ví dụ đầu tiên, ta có thể quan sát thấy đèn sáng và không nghe tiếng chó sủa, từ đây cần xác định trong nhà có người hay không bằng cách tính xác suất điều kiện P(¬ O | L, H).

Lưu ý: Trong công thức này, cũng như trong các phần tiếp theo, các chữ hoa như *E* được dùng để ký hiệu các biến, các chữ thường như *e* để ký hiệu giá trị cụ thể mà các biến đó có thể nhận. Trong các ví dụ với biến ngẫu nhiên nhị phân, ta dùng ký hiệu chữ hoa như *L* và chữ hoa với dấu phủ định ¬*L* để thể hiện biến đó nhận giá trị true hay false.

Tiếp sau đây, ta sẽ xem xét kỹ thuật suy diễn đơn giản nhất.

4.5.1. Suy diễn dựa trên xác suất đồng thời

Trong phần này, ta sẽ xem xét một phương pháp suy diễn được thực hiện bằng cách liệt kê và tính tổng các xác suất đồng thời có liên quan tới *Q* và *E*. Do phương pháp này cho phép tính chính xác xác suất $P(Q \mid E = e)$ nên được gọi là suy *diễn xác chính xác* (exact inference). Ngoài ra, do phương pháp này đòi hỏi liệt kê và tính các xác suất đồng thời nên còn gọi là phương pháp liệt kê (enumeration).

Phần giới thiệu về xác suất điều kiện ở trên đã giải thích cách tính xác suất điều kiện khi biết đầy đủ xác suất đồng thời. Cụ thể là, với các biến truy vấn *Q*, biến bằng chứng *E*, và các biến còn lại *Y* (tức là các biến không thuộc *Q* và *E*, còn được gọi là các nút ẩn), xác suất điều kiện được tính như sau:

$$P(Q \mid E = e) = \frac{P(Q, E = e)}{P(E)} = \frac{\sum_{Y} P(Q, e, Y)}{\sum_{Q,Y} P(Q, e, Y)}$$

Do mạng Bayes cho phép xác định toàn bộ xác suất đồng thời cần thiết, việc suy diễn có thể thực hiện bằng cách tính xác suất đồng thời, sau đó thay vào công thức này để tính xác suất $P(Q \mid E)$.

Ví dụ. Xét ví dụ dự đoán ở nhà có người hay không ở trên, giả sử ta biết rằng chó đang bị đau bụng và không nghe tiếng sủa, cần xác định xác suất đèn sáng, tức là xác suất P(L | B, ¬H). Theo công thức trên ta có:

$$P(L \mid B, \sim H) = \frac{\sum_{O,D} P(L, B, \sim H, O, D)}{\sum_{L,O,D} P(L, B, \sim H, O, D)}$$

trong đó

$$\sum_{O,D} P(L, B, \sim H, O, D) = P(L, B, \sim H, O, D) + P(L, B, \sim H, O, \sim D) + P(L, B, \sim H, \sim O, D) + P(L, B, \sim H, \sim O, \sim D)$$

và

$$\sum_{L,O,D} P(L,B,\sim H,O,D) = P(L,B,\sim H,O,D) + P(L,B,\sim H,O,\sim D)$$
$$+ P(L,B,\sim H,\sim O,D) + P(L,B,\sim H,\sim O,\sim D)$$
$$+ P(\sim L,B,\sim H,O,D) + P(L,B,\sim H,O,\sim D)$$
$$+ P(\sim L,B,\sim H,\sim O,D) + P(L,B,\sim H,\sim O,\sim D)$$

là các xác suất đồng thời được tính theo công thức xác suất đồng thời cho mạng Bayes như trong phần trước.

Độ phức tạp tính toán. Cách suy diễn thông qua xác suất đồng thời hay suy diễn bằng cách liệt kê có nguyên tắc đơn giản tuy nhiên đòi hỏi tính tổng của rất nhiều xác suất đồng thời. Với số lượng biến ngẫu nhiên lớn, số lượng xác suất đồng thời cần tính tăng theo hàm mũ. Cụ thể là với mạng Bayes gồm n nút tương ứng với n biến nhị phân và trong trường hợp Y gồm hầu hết các nút, độ phức tạp tính toán sẽ là $O(n2^n)$ do ta cần tính tổng của gần n số, mỗi số là tích của gần 2^n xác suất điều kiện. Độ phức tạp như vậy là rất lớn khi n lớn và do vậy không thực tế.

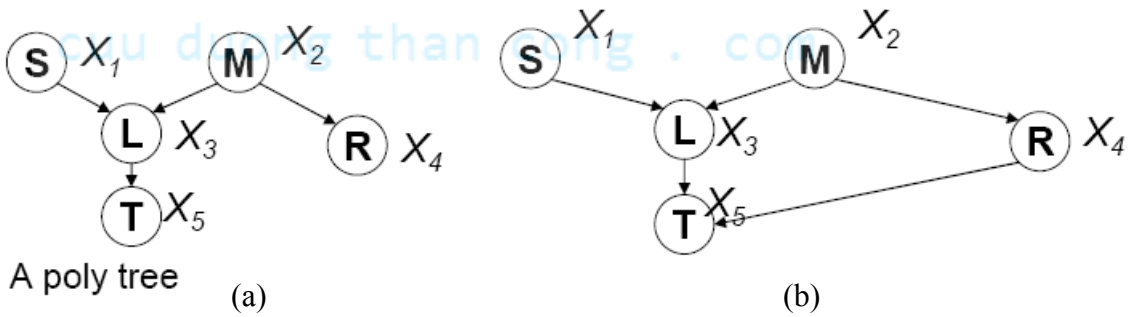
4.5.2. Độ phức tạp của suy diễn trên mạng Bayes

Ta đã thấy, việc tính chính xác giá trị $P(Q \mid E)$ thông qua xác suất đồng thời đòi hỏi độ phức tạp tính toán hàm mũ. Vậy việc suy diễn thông qua tính chính xác xác suất điều kiện nói chung (tạm gọi là suy diễn chính xác) có độ phức tạp tính toán ra sao khi sử dụng những phương pháp khác?

Đối với mạng có dạng liên kết đơn hay còn gọi là polytree.

Mạng liên kết đơn hay mạng polytree là mạng trong đó giữa hai nút bất kỳ chỉ tồn tại duy nhất một đường đi (hình 4.5.a). Trong trường hợp này, tồn tại thuật toán cho phép suy diễn chính xác với độ phức tạp tuyến tính (tỷ lệ thuận) với kích thước của mạng. Ở đây, kích thước mạng được tính bằng tổng số phần tử của các bảng xác suất điều kiện. Trong trường hợp số nút cha được giới hạn bởi hằng số, độ phức tạp sẽ tỷ lệ tuyến tính với số nút.

Trong phần cuối của chương (phần 4.5.5), ta sẽ xem xét thuật toán loại trừ biến (variable elimination), một thuật toán cho phép giảm đáng kể độ phức tạp tính toán so với phương pháp liệt kê, đặc biệt đối với mạng có cấu trúc polytree.



Hình 4.6: Mạng Bayes dạng liên kết đơn (a) và không phải liên kết đơn (b) với hai đường đi từ M tới T

Đối với mạng không kiên kết đơn hay mạng đa liên kết.

Đây là những mạng mà giữa hai nút có thể có nhiều hơn một đường đi (hình 4.5.b). Trong trường hợp này, việc tính chính xác xác suất điều kiện là bài toán NP-đầy đủ và do vậy không thể thực hiện khi mạng có kích thước lớn.

Để thực hiện suy diễn trong trường hợp mạng đa liên kết, có thể sử dụng các thuật toán cho phép tính xác suất điều kiện một cách xấp xỉ, chẳng hạn bằng cách lấy mẫu thống kê, hay sử dụng các thuật toán lan tỏa (propagation). Một kỹ thuật khác cũng có thể sử dụng là biến đổi mạng đa liên kết thành mạng liên kết đơn bằng cách chấp nhận mất một số thông tin. Bản thân thuật toán biến đổi này cũng có độ phức tạp tính toán khá lớn.

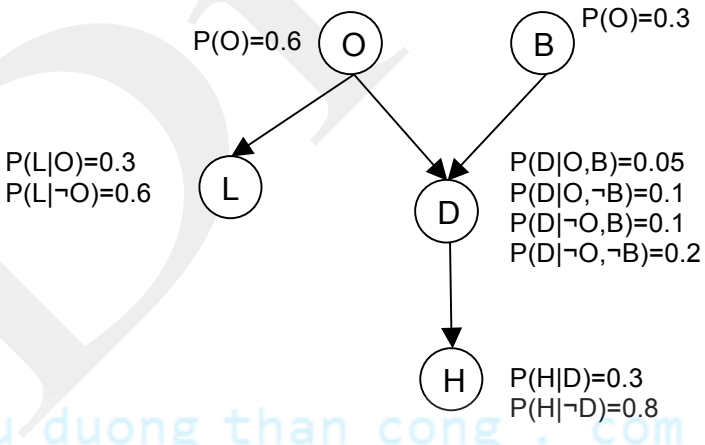
4.5.3. Suy diễn cho trường hợp riêng đơn giản

Phần này sẽ giới thiệu cách suy diễn cho trường hợp riêng đơn giản nhất, đó là trường hợp mạng liên kết đơn, suy diễn chỉ thực hiện cho các nút có liên kết trực tiếp với nhau. Có thể phân biệt hai loại suy diễn cho trường hợp đơn giản như vậy.

Trong trường hợp thứ nhất, biết giá trị một nút cha, yêu cầu tính xác suất quan sát thấy giá trị nút con. Suy diễn dạng này gọi là suy diễn *nhân quả* hoặc suy diễn từ trên xuống.

Trường hợp thứ hai ngược lại, tức là cho biết giá trị nút con và cần tính xác suất quan sát thấy giá trị nút cha. Trường hợp này được gọi là suy diễn *chẩn đoán* hoặc suy diễn dưới lên. Suy diễn chẩn đoán thường gặp trong những bài toán cần xác định hay chẩn đoán nguyên nhân của dấu hiệu nào đó.

Để tiện trình bày, các suy diễn sẽ được minh họa trên ví dụ đã sử dụng ở phần đầu bài. Như đã nói ở trên, ta chỉ xét trường hợp nút truy vấn và nút bằng chứng có liên kết trực tiếp với nhau.



Hình 4.7: Mạng Bayes

Suy diễn nhân quả hay suy diễn trên xuống

Trong suy diễn nhân quả, cho biết E, cần tính P(Q|E), trong đó E là một nút cha của Q.

Ví dụ: cần tính P(D|B)

$$\begin{aligned} P(D|B) &= P(D,O|B) + P(D,\neg O|B) \\ &= P(D|O,B)P(O|B) + P(D|\neg O,B)P(\neg O|B) \\ &= P(D|O,B)P(O) + P(D|\neg O,B)P(\neg O) \end{aligned}$$

$$= (.05)(.6) + (.1)(1 - .6) = 0.07$$

Ví dụ trên cho thấy, suy diễn nhân quả được thực hiện theo 3 bước.

- Bước 1: dòng 1. Trong bước này, xác suất Q cần tính được viết lại dưới dạng xác suất điều kiện của Q và cha của Q (không thuộc E) điều kiện theo E.
- Bước 2: dòng 2. Viết lại các xác suất đồng thời dưới dạng xác suất của Q khi biết các giá trị bố mẹ.
- Bước 3: dòng 3,4. Sử dụng các giá trị xác suất từ bảng xác suất điều kiện và thay vào công thức nhận được ở bước 2 để tính ra kết quả.

Suy diễn chẩn đoán hay suy diễn dưới lên

Đây là dạng suy diễn yêu cầu tính P(Q|E), trong đó Q là một nút cha của E.

Ví dụ: cần tính P(¬B|¬D)

Sử dụng quy tắc Bayes, ta viết lại giá trị cần tính như sau

$$P(\neg B|\neg D) = P(\neg D|\neg B)P(\neg B)/P(\neg D)$$

Tiếp theo, tính P(¬D|¬B) như cách suy diễn nhân quả ở trên

$$P(\neg D|\neg B) = P(\neg D|O,\neg B)P(O) + P(\neg D|\neg O,\neg B)P(\neg O)$$
$$= (.9)(.6) + (.8)(.4)$$
$$= 0.86$$

Thay giá trị P(¬D|¬B) vừa tính, ta được:

$$P(\neg B|\neg D) = (.86)(.7)/P(\neg D) = .602/P(\neg D)$$

Để tính P(¬D), có thể tính P(B|¬D), sau đó dùng P(¬B|¬D)+P(B|¬D) = 1

$$P(B|\neg D) = (.93)(.3)/P(\neg D) = .279/P(\neg D)$$

do đó:

$$0.602/P(\neg D) + 0.279/P(\neg D) = 1$$

suy ra:

$$P(\neg D) = 0.881$$

thay vào trên:

$$P(\neg B|\neg D) = 0.602 / 0.881 = 0.683$$

Trong trường hợp chung, suy diễn chẩn đoán có thể thực hiện qua hai bước như sau:

- Bước 1: Biến đổi về suy diễn nhân quả sử dụng quy tắc Bayes
- Bước 2: Thực hiện giống suy diễn nhân quả như đã trình bầy ở phần trước.

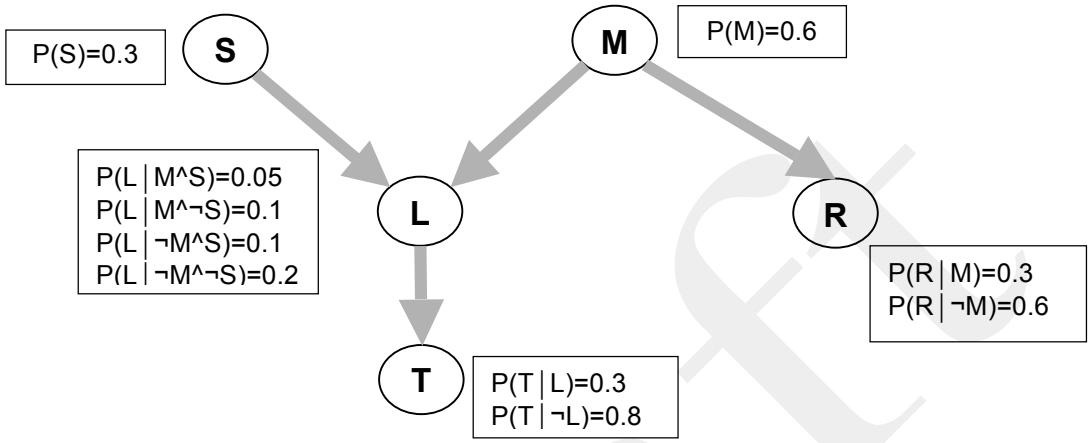
4.5.4. Suy diễn bằng phương pháp lấy mẫu

Trong trường hợp mạng Bayes có cấu trúc phức tạp, việc tính chính xác các xác suất điều kiện là không thực tế do độ phức tạp tính toán lớn. Một phương pháp suy diễn thường được sử dụng là việc tính toán xấp xỉ xác suất điều kiện bằng cách lấy mẫu. Thực chất quá trình này là sinh ra các bộ giá trị của các biến theo phân bố xác suất trên mạng, sau đó cộng lại và tính tần suất quan sát được, dùng giá trị tần suất này làm giá trị (xấp xỉ) của xác suất cần tính. Phương pháp suy diễn bằng lấy mẫu còn được gọi là suy diễn xấp xỉ do xác suất tính được là giá trị gần đúng của xác suất thực.

a) Phương pháp lấy mẫu đơn giản

Phương pháp lấy mẫu đơn giản nhất cho mạng Bayes được thực hiện bằng cách sinh ra các bộ giá trị cho các biến bằng cách lần lượt lấy mẫu các biến theo thứ tự được quy định trên mạng, tức là các nút cha mẹ được lấy mẫu trước, nút con được lấy mẫu sau. Mỗi biến được lấy mẫu theo xác suất điều kiện tùy thuộc vào giá trị nút cha mẹ đã được gán nhờ bước lấy mẫu trước đó.

Việc lấy mẫu được minh họa qua ví dụ mạng Bayes trên hình dưới. Quá trình lấy mẫu và tính xác suất điều kiện dựa trên các mẫu sinh ra được trình bầy lần lượt dưới đây.



Hình 4.8. Ví dụ mạng Bayes

Lấy mẫu

Từ mô hình mạng, ta có thể sắp xếp các nút theo thứ tự [*S, M, L, R, T*]. Thứ tự này đảm bảo các nút cha luôn nằm trước nút con (có thể chọn thứ tự khác miễn là đảm bảo điều kiện nút cha nằm trước nút con).

Tại mỗi bước lấy mẫu, một bộ giá trị của các biến được xác định bằng cách lần lượt lấy mẫu các biến theo thứ tự đã xác định như sau.

- Chọn ngẫu nhiên giá trị của S theo xác suất $P(S$=true) = 0.3. Giả sử được giá trị S = false.
- Chọn ngẫu nhiên giá trị M theo xác suất $P(M$ = true) = 0.6. Giả sử được giá trị M = true.
- Chọn ngẫu nhiên giá trị L theo xác suất $P(L|S$ = false, M = true) = 0.1. Giá trị của S và M đã được chọn ở trên. Giả sử được giá trị L là false.
- Chọn ngẫu nhiên R theo xác suất $P(R \mid M$ = true) = 0.3. Giả sử được R = true.
- Chọn ngẫu nhiên T theo xác suất $P(T \mid L$=false) = 0.8. Giả sử được T = true.

Như vậy, sau một lần lấy mẫu, ta được bộ giá trị (¬S, M, ¬L, R, T).

Cách lấy mẫu tại mỗi bước là trước tiên lấy mẫu các nút không có nút cha. Sau đó lấy mẫu tới các nút con. Khi lấy mẫu nút con sẽ sử dụng phần tử trong bảng xác suất điều kiện tương ứng với các giá trị nút cha đã lựa chọn được trước đó. Mô tả thuật toán lấy mẫu thể hiện trên hình sau.

- Giả sử ta có các biến/nút là X_1, … X_n sao cho Cha_Mẹ(X_i) là tập con của {X_1, …, X_{i-1}} // tức nút cha luôn đứng trước con
- For i = 1 to n do
    1. Tìm Cha_Mẹ(X_i) (nếu có). Giả sử có n(i) nút cha của Xi. Gọi các nút là $X_{p(i,1)}$, …, $X_{p(i,n(i))}$)
    2. Giả sử giá trị các nút cha đã sinh ra trước đó là:$x_{p\,(i,1)}$, …, $x_{p(i,n(i))}$)
    3. Tìm trong bảng xác xuất giá trị xác suất:

       $P(X_i = true \mid X_{p(i,1)} = x_{p(i,1)} \ldots)$
    4. Chọn X_i = true với xác xuất trên

  Kết quả x_1, … x_n là một mẫu sinh ra từ các biến X_1, …, X_n

Hình 4.9: Thuật toán lấy mẫu từ mạng Bayes

Tính xác suất điều kiện

Tiếp theo, giả sử cần tính $P(R = \text{True} \mid T = \text{True}, S = \text{False})$.

Trước hết, ta thực hiện lấy mẫu nhiều lần theo cách ở trên, mỗi bộ giá trị sinh ra được gọi là một mẫu. Nếu số lượng mẫu đủ lớn thì tần suất xuất hiện mỗi bộ giá trị sẽ xấp xỉ xác suất đồng thời của các giá trị đó. Xác suất đồng thời này sẽ được sử dụng để tính xác suất điều kiện như dưới đây.

Tính số lần xảy ra những sự kiện sau:

- N_c: số mẫu có T = True và S = False
- N_s: số mẫu có R = True, T = True và S = False
- N: tổng số mẫu

Nếu N đủ lớn:

- N_c / N: (xấp xỉ) xác suất P (T = True, S = False)
- N_s / N: (xấp xỉ) xác suất P (R = True , T = True , S = False)
- $P(R \mid T, \neg S) = \dfrac{\mathrm{P(R, T, \sim S)}}{\mathrm{P(T, \sim S)}} \approx \dfrac{N_s}{N_c}$

Như vậy, bằng cách lấy mẫu và đếm số lần các tổ hợp giá trị xuất hiện, ta có thể tính được xác suất điều kiện.

Phương pháp lấy mẫu

Từ ví dụ minh họa ở trên, có thể mô tả cách suy diễn xấp xỉ bằng cách lấy mẫu như sau. Giả sử cho mạng Bayes và cần tính $P(Q \mid E)$. Thực hiện các bước dưới đây:

- Lấy mẫu số lượng đủ lớn
- Tính số lượng:

- N_c: số mẫu có E
- N_s: số mẫu có Q và E
- N: Tổng số mẫu

• Nếu N đủ lớn, ta có: $P(Q \mid E) = N_s / N_c$

Hình 4.10: Thuật toán suy diễn bằng cách lấy mẫu trên mạng Bayes

Vấn đề với phương pháp lấy mẫu đơn giản

Phương pháp lấy mẫu đơn giản trình bầy ở trên có một nhược điểm là tính hiệu quả không cao. Trong trường hợp tập E gồm nhiều biến hoặc tập E gồm những giá trị ít xẩy ra thì N_c rất nhỏ so với N và do vậy phương pháp sẽ sinh ra rất nhiều mẫu mà không dùng tới. Tiếp theo đây, ta sẽ xem xét một phương pháp lấy mẫu cho phép cải thiện tính hiệu quả của lấy mẫu đơn giản.

b) Lấy mẫu với trọng số theo khả năng (likelihood weighting)

Lấy mẫu với trọng số theo khả năng cho phép cải thiện hiệu quả lấy mẫu bằng cách chỉ sinh ra những mẫu có chứa các giá trị của tập quan sát E. Phương pháp này cố định các giá trị của các biến trong tập E và chỉ lấy mẫu các biến không thuộc E. Như vậy, các mẫu sinh ra luôn chứa các giá trị cần thiết của E và do vậy không bị bỏ phí khi tính xác suất, tức là $N_c = N$ theo ký hiệu như ở phần trên.

Phương pháp được thực hiện như sau. Giả sử trong tập E ta có $X_i = v$ (X_i là một biến ngẫu nhiên).

Theo cách lấy mẫu đơn giản, ta sẽ sinh ra giá trị của X_i một cách ngẫu nhiên với xác suất $P(X_i = v \mid$ cha_me_ $X_i) = p$. Như vậy, theo phương pháp lấy mẫu đơn giản, ta sẽ sinh ra $X_i = v$ với tần suất hay tỷ lệ là p và sinh ra $X_i \neq v$ với tần suất hay tỷ lệ $(1 - p)$. Rõ ràng, các mẫu có $X_i \neq v$ bị bỏ phí. Theo lấy mẫu với trọng số theo khả năng, ta sẽ luôn sinh ra $X_i = v$, tuy nhiên sẽ nhân trọng số của mẫu được sinh ra với p để bù lại.

Lấy mẫu

Phương pháp lấy mẫu này được minh họa với ví dụ mạng Bayes trên hình 4.10. Giả sử cần tính $P(R =$ true $\mid T =$ true , $S =$ false), và các nút được sắp xếp theo thứ tự [*S, M, L, R, T*]. Trước tiên ta đặt giá trị cho trọng số $w = 1$. Quá trình lấy mẫu sau đó diễn ra như sau:

- S là biến ngẫu nhiên thuộc E với giá trị S = false, do vậy ta thay đổi trọng số
  $w \leftarrow w * P(S =$ false) = 0.7. //ở đây, ký hiệu a ← b có nghĩa là gán giá trị b cho a
- M không thuộc E, do vậy chọn ngẫu nhiên M với xác suất P(M = true) = 0.6. Giả sử ta được M = true.
- Tương tự như vậy, chọn ngẫu nhiên L với xác suất P(L = true|S = false, M = true) = 0.1. Giả sử ta được L = false.
- Tương tự, chọn ngẫu nhiên R với xác suất P(R = true | M = true) = 0.3. Giả sử R = true.
- T là biến thuộc E với giá trị true, do vậy ta thay đổi trọng số
  $w \leftarrow w * P(T =$ true | L = false) = 0.7 * 0.8 = 0.56

Như vậy, trong ví dụ lần lấy mẫu này trả về bộ giá trị [S = false, M = true, L = false, R = 0.3, T = true] với trọng số 0.56.

Quá trình lấy mẫu này được tóm tắt trong thuật toán sau:

Lấy mẫu có trọng số (Mạng Bayes, tập nút E và giá trị e)

Input:

- Mạng Bayes
- Các biến/nút là $X_1, \ldots X_n$ sao cho Cha_Mẹ(X_i) là tập con của $\{X_1, \ldots, X_{i-1}\}$ // tức nút cha luôn đứng trước con
- Tập các nút E và giá trị quan sát được **e** của các nút đó

Khởi tạo:

w←1; **x** ← bộ giá trị cho n biến với giá trị được khởi tạo từ **e**

For i = 1 **to** n **do**

If Xi là biến thuộc tập E với giá trị x[i] được khởi tạo tự e **then**

$w \leftarrow w * P(X_i = \mathbf{x}[i] \mid \text{Cha\_mẹ}(X_i))$

Else **x**[i] ← chọn ngẫu nhiên giá trị từ phân bố $P(X_i \mid \text{Cha\_mẹ}(X_i))$

Endif

Return: **x**, w.

Hình 4.11. Thuật toán lấy mẫu có trọng số

Tính xác suất điều kiện

Sau khi thực hiện lấy mẫu với trọng số với số mẫu đủ lớn, ta có thể tính xác suất điều kiện theo công thức $P(Q \mid E) = N_s / N_c$ tương tự trong trường hợp lấy mẫu đơn giản. Tuy nhiên, cách tính giá trị N_s và N_c không còn giống trường hợp lấy mẫu đơn giản nữa. Nếu như trong trường hợp lấy mẫu đơn giản, hai giá trị này được tính bằng tổng số lần các bộ giá trị tương ứng xuất hiện, thì trong trường hợp lấy mẫu với trọng số, ta cần tính tổng trọng số cho các bộ giá trị. Cụ thể, xác suất được tính như sau:

Input:

- Mạng Bayes
- Tập các nút E và giá trị quan sát được **e** của các nút đó
- Tập biến truy vấn Q và các giá trị của chúng.

Khởi tạo:

$N_s \leftarrow 0$; $N_c \leftarrow 0$

1. Sinh ra bộ giá trị **x** và trọng số w sử dụng thuật toán "**Lấy mẫu có trọng số**"
2. $N_c \leftarrow N_c + w$
3. **If** bộ giá trị **x** có chứa các giá trị của Q **then** cập nhật $N_s \leftarrow N_s + w$
4. **Goto 1**

Return: $P(Q \mid E) = N_s / N_c$

Hình 4.12. Tính xác suất điều kiện bằng lấy mẫu có trọng số

Phương pháp lấy mẫu với trọng số theo khả năng có hiệu quả tốt hơn so với lấy mẫu đơn giản do không sinh ra các mẫu thừa. Tuy nhiên, hiệu quả của phương pháp này giảm khi số lượng biến trọng tập E tăng lên hoặc các biến này xuất hiện ở cuối trong thứ tự các biến trên mạng.

4.5.5. Phương pháp loại trừ biến

Loại trừ biến (variable elimination) là phương pháp suy diễn chính xác, tức là cho phép tính chính xác xác suất $P(Q \mid E)$, tuy nhiên đòi hỏi tính toán ít hơn nhiều so với thuật toán suy diễn bằng cách liệt kê đã trình bầy ở phần 4.5.1. Phương pháp loại trừ biến cho phép giảm khối lượng tính toán nhờ sử dụng kỹ thuật quy hoạch động và khai thác đặc điểm cấu trúc của mạng Bayes.

a) Ví dụ minh hoạ.

Để nắm được nguyên lý cơ bản của loại trừ biến, trước khi đi vào tìm hiểu chi tiết về phương pháp ta sẽ xem xét ví dụ suy diễn cho một mạng Bayes đơn giản với 4 biến ngẫu nhiên được kết nối tuần tự với nhau như minh hoạ trên hình 4.13. Ví dụ sẽ cho thấy, ngoài việc biểu diễn ngắn gọn các xác suất đồng thời, kiến trúc mạng cũng giúp tiết kiệm các thao tác tính toán được sử dụng khi suy diễn.

Hình 4.13. Ví dụ mạng Bayes đơn giản

Để đơn giản, ta sẽ giả sử các biến ngẫu nhiên trong ví dụ đang xét là các biến nhị phân, có thể nhận giá trị True hoặc False. Theo công thức tính xác suất đồng thời, xác suất $P(A, B, C, D)$ được tính như sau:

$$P(A)P(B \mid A)P(C \mid B)P(D \mid C)$$

Giả sử cần tính xác suất $P(D)$. Xác suất này được tính riêng cho hai trường hợp D = True (viết tắt là D), và D = False (viết tắt là ¬D) bằng cách tính tổng các xác suất đồng thời với tất cả các tổ hợp giá trị của A, B, và C cho từng giá trị của D (đây là trường hợp biến nhị phân do vậy chỉ cần tính P(D) rồi tính P(¬D) = 1 – P(D), tuy nhiên với biến nhiều hơn hai giá trị thì cần tính riêng xác suất biến đó nhận mỗi giá trị). Ví dụ, với D = True, ta có:

$$P(D) = \sum_A \sum_B \sum_C P(A)P(B \mid A)P(C \mid B)P(D \mid C)$$

Viết chi tiết vế phải, ta được:

$$P(A)P(B|A)P(C|B)P(D|C)+$$
$$P(A)P(B|A)P(\neg C|B)P(D|\neg C)+$$
$$P(A)P(\neg B|A)P(C|\neg B)P(D|C)+$$
$$P(A)P(\neg B|A)P(\neg C|\neg B)P(D|\neg C)+$$
$$P(\neg A)P(B|\neg A)P(C|B)P(D|C)+$$
$$P(\neg A)P(B|\neg A)P(\neg C|B)P(D|\neg C)+$$
$$P(\neg A)P(\neg B|\neg A)P(C|\neg B)P(D|C)+$$
$$P(\neg A)P(\neg B|\neg A)P(\neg C|\neg B)P(D|\neg C)$$

Như vậy, để tính được $P(D)$ theo cách này, cần thực hiện tất cả 24 phép nhân và 7 phép cộng. Trong trường hợp tổng quát với n biến, mỗi biến có thể nhận k giá trị, cần thực hiện $(n-1)k^{(n-1)}$ phép nhân và $k^{(n-1)} - 1$ phép cộng.

Tuy nhiên, biểu thức trên chứa nhiều thành phần lặp lại. Chẳng hạn, dòng thứ nhất và dòng thứ năm đều chứa $P(C|B)P(D|C)$. Do vậy, thay vì tính đầy đủ dòng thứ nhất và thứ năm, ta có thể tính tổng $P(A)P(B|A)+P(\neg A)P(B|\neg A)$, sau đó nhân kết quả tính được với thành phần chung. Bằng cách này có thể giảm số phép tính cần thực hiện cho dòng thứ nhất và thứ năm xuống còn 4 phép nhân và 1 phép cộng thay vì 6 phép nhân và 1 phép cộng như trước. Tương tự như vậy, rất nhiều thành phần khác được lặp lại trong biểu thức và có thể tranh thủ các cấu trúc như vậy để giảm số phép tính.

Quay về biểu thức ban đầu, ý tưởng rút gọn việc tính toán được thể hiện bằng cách viết lại biểu thức như sau:

$$P(D) = \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C)$$
$$= \sum_C P(D|C) \sum_B P(C|B) \sum_A P(B|A)P(A)$$

Dễ dàng nhận thấy, biểu thức dưới cho giá trị giống hệt biểu thức ban đầu nhưng đòi hỏi thực hiện ít phép nhân hơn.

Tiếp theo, có thể rút gọn hơn nữa số phép tính cần thực hiện nếu để ý rằng phần tính $\Sigma_A P(A)P(B|A)$ đang được thực hiện lặp lại với từng giá trị của C. Việc lặp lại như vậy là thừa do biểu thức này không phụ thuộc vào C. Để rút gọn, có thể thực hiện phép nhân một lần và lưu kết quả lại. Cụ thể, với mỗi giá trị của A, ta thực hiện phép nhân và nhận được ma trận sau:

$$\begin{bmatrix} P(B|A)P(A) & P(B|\neg A)P(\neg A) \\ P(\neg B|A)P(A) & P(\neg B|\neg A)P(\neg A) \end{bmatrix}$$

tiếp theo, cộng giá trị theo mỗi dòng, ta được mảng sau cho từng giá trị của B:

$$\begin{bmatrix} P(B) \\ (\neg B) \end{bmatrix}$$

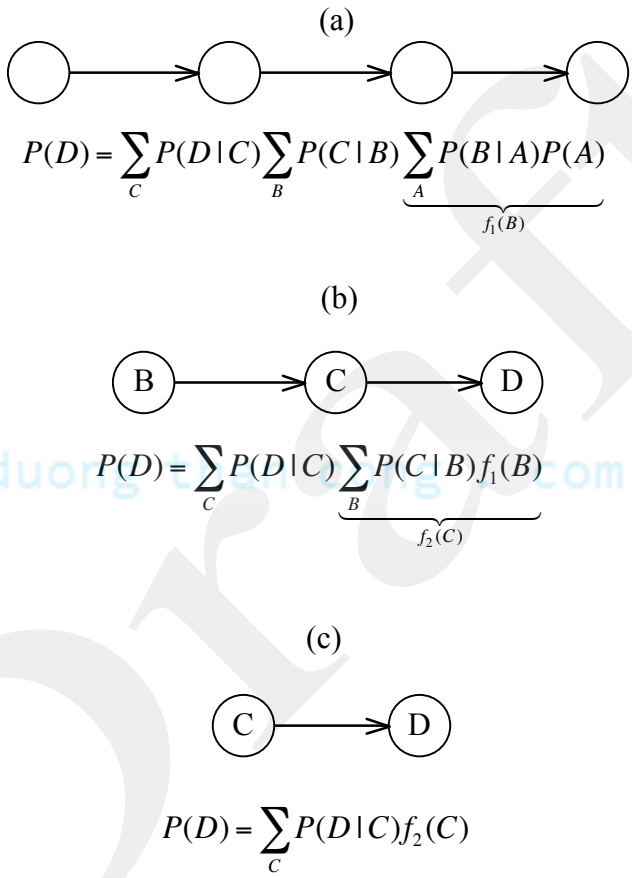
Tập các giá trị này chỉ phụ thuộc vào giá trị của B và được gọi là **thừa số** theo B. Ký hiệu thừa số này là $f_1(B)$, ta có (hình 4.14.a):

$$f_1(B) = \sum_A P(B \mid A)P(A)$$

Sau khi tính được $f_1(B)$, ta có thể thay thế giá trị này vào chỗ của $\Sigma_A P(A)P(B|A)$, và loại bỏ nút A khỏi mạng ban đầu như thể hiện trên hình 4.14.b. Như vậy, $f_1(B)$ biểu diễn phần đóng góp của B, đã tính tới đóng góp của A vào công thức tính xác suất.

Tiếp theo, ta có thể thực hiện rút gọn tương tự như trên. Do phần cộng theo giá trị của C chỉ phụ thuộc vào B, có thể cộng các giá trị đó để tạo ra các thừa số chỉ phụ thuộc vào C, ký hiệu $f_2(C)$ như minh hoạ trên hình 4.14.b.

Tiếp theo, thay thế $f_2(C)$ vào biểu thức, xoá nút B khỏi mạng, ta còn lại biểu thức đơn giản, từ đó có thể tính giá trị của D bằng cách lấy tổng theo các giá trị của C (hình 4.14.c).



Hình 4.14. Ví dụ minh hoạ cách loại trừ biến cho trường hợp đơn giản.

b) Thuật toán loại trừ biến

Ví dụ trên cho thấy nguyên tắc cơ bản của phương pháp loại trừ biến trên một mạng Bayes đơn giản. Ý tưởng là tính toán sẵn và lưu các thành phần lặp lại, sau đó sử dụng các giá trị đã lưu để tiết kiệm số phép tính cần thực hiện theo nguyên tắc quy hoạch động. Tiếp theo, ta sẽ xem xét thuật toán loại trừ biến cho trường hợp tổng quát.

Để trình bầy thuật toán, trước hết cần biết khái niệm thừa số và thừa số hoá.

Thừa số hoá một phân phối xác suất đồng thời là phân tích xác suất đó thành các thành phần, mỗi thành phần phụ thuộc vào một hoặc một số biến ngẫu nhiên, sao cho tích các thành phần đó là giá trị xác suất đồng thời. Có nhiều cách để thừa số hoá một phân bố xác suất. Như trong ví dụ ở trên, một trong các cách thừa số hoá cho ta các thành phần sau

$$F = \{P(A), P(B \mid A), P(C \mid B), P(D \mid C)\}$$

Mỗi thành phần sau khi thừa số hoá gọi là một **thừa số**. Như trong ví dụ trên, ta có $P(A)$, $P(B \mid A)$ ... $P(D \mid C)$ là các thừa số. Lưu ý rằng, mỗi thừa số là hàm của một số biến ngẫu nhiên, chẳng hạn P(A) là hàm phụ thuộc vào A, P(D | C) là hàm và phụ thuộc vào D, C. Giá trị của mỗi thừa số là một ma trận, chẳng hạn nếu các biến là nhị phân thì P(A) là ma trận kích thước 2x1, trong khi P(D | C) là ma trận kích thước 2x2 do mỗi biến D và C có thể nhận 2 giá trị.

Thuật toán loại trừ biến được thực hiện bằng cách loại trừ dần các biến khỏi biểu thức tính phân bố xác suất đồng thời (từ đây mà có tên gọi của phương pháp là "loại trừ biến"). Việc loại trừ một biến khỏi biểu thức tính xác suất đồng thời được thực hiện bằng cách loại trừ các thừa số chứa biến đó khỏi biểu thức như đã thấy trong ví dụ ở phần trên. Lưu ý rằng, sau khi loại từ một biến, ta được một danh sách các thừa số khác sao cho tích các thừa số đó bằng giá trị xác suất đồng thời nhưng chứa ít biến ngẫu nhiên hơn.

Trước hết ta sẽ xem xét thủ tục *Eliminate*() cho phép loại trừ *một biến*, trước khi sử dụng thủ tục này trong thuật toán loại trừ biến tổng quát.

Giả sử xác suất đồng thời được thừa số hoá thành:

$$P(X_1, X_2, ..., X_m) = f_1.f_2....f_n,$$

tương ứng với danh sách các thừa số

$$F = \{f_1, f_2, ..., f_n\}$$

Việc loại trừ một biến được thực hiện bằng cách loại bỏ tất cả các thừa số chứa biến đó khỏi F, nhân giá trị các thừa số đó với nhau, sau đó cộng các tích nhận được theo các giá trị khác nhau của biến cần loại trừ. Thủ tục loại trừ một biến *Eliminate*() được thể hiện trên hình dưới đây.

Eliminate(F, X_i)

Input:

F: danh sách các thừa số

X_i: biến cần loại trừ

Output:

Danh sách mới các thừa số, danh sách này không chứa X_i

---

1. Loại khỏi F tất cả các thừa số chứa X_i. Không mất tính tổng quát, giả sử các thừa số đó là $f_1, f_2, ..., f_k$.
2. Tính tích của các thừa số trên: $g = f_1.f_2....f_k$
3. Tính tổng của g theo các giá trị của X_i: $h = \sum_{X_i} g$
4. Thêm h như một thừa số mới vào F

Return F (bây giờ đã không còn thừa số nào chứa X_i)

Hình 4.15: Thủ tục Eliminate cho phép loại trừ một biến khỏi xác suất đồng thời

Trong thủ tục Eliminate() ở hình trên, g là mảng gồm $K + 1$ chiều, trong đó một chiều ứng với X_i và K chiều còn lại ứng với các biến ngẫu nhiên khác chứa trong $f_1, f_2,...,f_k$. Sau khi thực hiện bước 3, ta được hàm h có dạng một bảng với K chiều do đã giảm được một chiều tương ứng với $f_1, f_2,...,f_k$.

Với thủ tục *Eliminate*() vừa trình bầy, ta có thể xây dựng thuật toán loại trừ biến, theo đó các biến không phải biến truy vấn Q và không phải biến bằng chứng E sẽ bị loại trừ dần bằng cách gọi Eliminate(). Sau khi đã loại trừ hết các biến không thuộc Q và E, có thể tính xác suất điều kiện $P(Q \mid E = e)$. Để thực hiện dần việc loại trừ các biến, ta cần sắp xếp các biến cần loại trừ theo một thứ tự nào đó. Các biến sẽ lần lượt được loại trừ theo đúng thứ tự này. Sau khi đã loại trừ các biến, các thừa số còn lại chỉ còn chứa Q và E, từ đó có thể tính ra xác suất $P(Q \mid E = e)$.

Thuật toán loại trừ biến VE() được trình bầy trên hình 4.16. Thuật toán nhận đầu vào là biến truy vấn, các biến bằng chứng và giá trị, danh sách các thừa số ban đầu và thứ tự các biến cần loại trừ. Lưu ý rằng danh sách thừa số ban đầu chính là các bảng xác suất điều kiện cho cùng mạng Bayes.

VE(Q, E, e, F, $\wp$)

Input:

Q: các biến truy vấn

E: các biến bằng chứng

e: giá trị các biến bằng chứng

F: danh sách các thừa số ban đầu, chính là danh sách các bảng xác suất điều kiện

$\wp$: các biến không thuộc Q và E được sắp theo thứ tự (thứ tự loại trừ các biến)

Output: $P(Q \mid E = e)$

---

1. Thay thế các biến bằng chứng bằng giá trị quan sát được e trong tất cả thừa số
2. While $\wp$ không rỗng
   a. Lấy biến đầu tiên X từ $\wp$
   b. Thực hiện Eliminate (F, X)
3. End While
4. Tính h = tích các thừa số còn lại trong F
5. Tính $P(Q \mid E = e) = \dfrac{h(Q)}{\sum_{Q} h(Q)}$

Return $P(Q \mid E = e)$

Hình 4.16. Thuật toán Loại trừ biến dùng cho suy diễn trên mạng Bayes

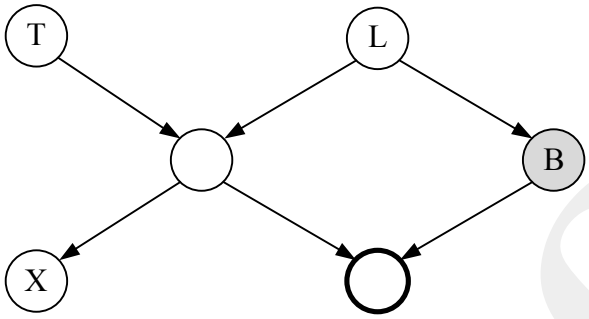
Trước hết, ở bước 1, thuật toán thay thế các biến bằng chứng E bằng giá trị quan sát được e. Phần chính của thuật toán là vòng lặp ở bước 2. Tại mỗi bước lặp, thuật toán gọi thủ

tục *Eliminate*() để loại bỏ một biến theo đúng thứ tự quy định trong tham số đầu vào. Bước 4 tính tích các thừa số còn lại và chỉ chứa các biến truy vấn. Bước cuối cùng thực hiện chuẩn tắc hoá để tính ra xác suất truy vấn.

c) Ví dụ minh hoạ thuật toán

Để minh hoạ thuật toán, ta xét một ví dụ khác phức tạp hơn ví dụ ở trên. Cho mạng Bayes như trên hình 4.17.a với các biến nhị phân. Giả sử cần tính phân phối xác suất cho D khi biết B = false, tức là P(D | B = false). Giả sử thứ tự loại trừ biến là T, X, L, A.

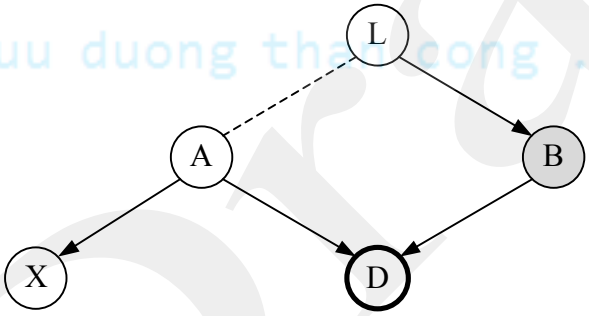
(a)



$$F = \{P(T), P(X \mid A), P(L), P(A \mid T, L), P(B \mid L), P(D \mid A, B)\}$$

(b) $$F = \{P(T), P(X \mid A), P(L), P(A \mid T, L), P(B = false \mid L), P(D \mid A, B = false)\}$$

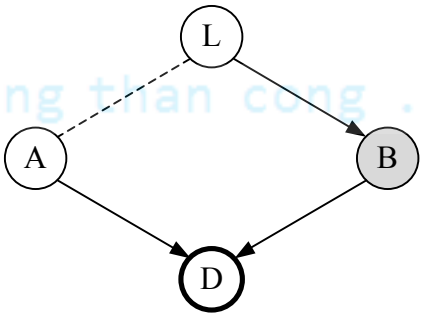
(c)



$$F = \{P(X \mid A), P(L), P(B = false \mid L), P(D \mid A, B = false), f_1(A, L)\}$$

$$f_1(A, L) = \sum_T P(T) P(A \mid T, L)$$

(d)



$$F = \{P(L), P(B = false \mid L), P(D \mid A, B = false), f_1(A, L)\}$$

do $\sum_X P(X \mid A) = 1$

Hình 4.17: Ví dụ các bước thực hiện thuật toán loại trừ biến

Trước hết, xác suất đồng thời có thể thừa số hoá thành

$$F = \{P(T), P(X \mid A), P(L), P(A \mid T, L), P(B \mid L), P(D \mid A, B)\}$$

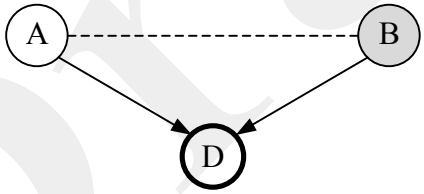
Thực hiện bước 1 của thuật toán, ta thay giá trị B = true và được các thừa số như trên hình 4.17.b.

Tiếp theo, thực hiện vòng lặp chính ở bước 2, ta loại trừ dần các biến theo thứ tự đã chọn. Trước hết, theo thứ tự, ta thực hiện loại trừ biến T. Trong danh sách F hiện thời có hai thừa số chứa T là $P(T)$ và $P(A \mid T, L)$. Các thừa số này chứa A và L nên ta tính tích cho mỗi bộ giá trị của A và L sau đó cộng theo các giá trị của T để tạo ra thừa số mới $f_1(A, L)$ phụ thuộc vào A và L như minh hoạ trên hình 4.17.c. Sau đó, ta có thể thay thế thừa số mới vào F và xoá nút T khỏi mạng. Lưu ý rằng bây giờ A và L phụ thuộc vào nhau thông qua thừa số mới $f_1(A, L)$ thay vì xác quan hệ phụ thuộc như lúc đầu. Để thể hiện quan hệ mới, cạnh $L \rightarrow A$ được thay bằng đường đứt đoạn như trên hình vẽ.

Biến tiếp theo cần loại trừ là X. Trong danh sách F hiện thời có một thừa số $P(X \mid A)$ chứa X và cả A. Do vậy cần cộng $P(X \mid A)$ theo các giá trị của X. Do tổng các xác suất như vậy có giá trị là 1 nên ta có thể đơn giả bỏ thừa số này khỏi F và xoá nút X khỏi mạng mà không cần thêm thừa số mới (hình 4.17.d).

Tiếp theo, ta loại trừ biến L và thêm thừa số mới $f_2(A, B = false)$ như trên hình 4.17.e, đồng thời xoá nút L và vẽ đường đứt đoạn thể hiện quan hệ giữa A và B thông qua thừa số mới. Cuối cùng, loại trừ nút A, ta được danh sách thừa số với một thừa số f_3 duy nhất chứa D và B như trên hình 4.17.f. Thực hiện bước 4 và 5 của thuật toán, ta được kết quả như trên hình 4.17.g.

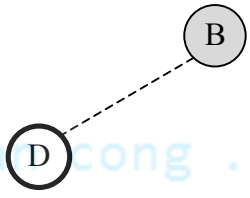
(e)



$$F = \{P(D \mid A, B = false), f_2(A, B = false)\}$$

trong đó: $$f_2(A, B = false) = \sum_L P(L) P(B = false \mid L) f_1(A, L)$$

(f)



$$F = \{f_3(D, B = false)\}$$

trong đó: $$f_3(D, B = false) = \sum_A P(D \mid A, B = false) f_2(A, B = false)$$

(g)

$$h(D) = f_3(D, B = false)$$

$$P(D \mid B = false) = \frac{h(D)}{\sum_D h(D)}$$

Hình 4.17 (tiếp theo)

d) Đặc điểm của thuật toán loại trừ biến

Độ phức tạp tính toán. Độ phức tạp tính toán của thuật toán loại trừ biến tăng theo hàm mũ đối với kích thước của thừa số lớn nhất, tức là số biến của thừa số đó. Thật vậy, nếu thừa số có k biến thì ta cần tạo ra bảng (k+1) chiều, trong đó k chiều cho k biến và chiều còn lại cho biến cần loại trừ. Nếu mỗi biến có thể nhận b giá trị thì bảng gồm $b^{(k+1)}$ phần tử. Để tính mỗi phần tử của bảng cần thực hiện tối đa n phép nhân với n là số nút trên mạng. Các thao tác này được lặp lại với mỗi biến cần loại trừ và số biến cần loại trừ thường gần bằng n. Như vậy, độ phức tạp tính toán của thuật toán loại trừ biến là $O(n^2 b^{(k+1)})$.

Kích thước của thừa số phụ thuộc vào thứ tự loại trừ biến. Kích thước các thừa số sinh ra trong quá trình loại trừ biến phụ thuộc vào thứ tự loại trừ. Việc lựa chọn thứ tự không tốt có thể sinh ra thừa số có kích thước lớn hơn nhiều so với thứ tự tối ưu. Như vậy, việc lựa chọn thứ tự không tốt làm tăng độ phức tạp tính toán của thuật toán.

Bài toán tìm thứ tự loại trừ tốt nhất là bài toán NP khó. Từ đặc điểm trên, ta cần tìm thứ tự biến cho các thừa số nhỏ nhất. Tuy nhiên, bản thân bài toán tìm thứ tự này là bài toán NP khó và không thể giải trong thời gian đa thức. Ngoài ra, kể cả với thứ tự loại trừ tốt nhất vẫn có thể sinh ra các thừa số kích thước lớn nếu mạng có cấu trúc phức tạp.

Có thể tìm ra thứ tự loại trừ tốt phương pháp heuristics. Có thể sử dụng heuristic để tìm thứ tự loại trừ biến tốt. Thay vì sắp xếp toàn bộ biến cần loại trừ ngay từ đầu, ta có thể lựa chọn biến tiếp theo để loại trừ trong khi thực hiện thuật toán. Một heuristics thường được dùng là lựa chọn biến tiếp theo để loại trừ sao cho tạo ra thừa số có kích thước nhỏ nhất. Thực tế cho thấy, heuristic đơn giản này cho kết quả khá tốt.

Với mạng dạng polytree, thuật toán loại trừ biến có độ phức tạp tuyến tính theo kích thước mạng. Nhắc lại, mạng polytree là mạng trong đó giữa hai nút bất kỳ chỉ có một đường đi. Ở đây, kích thước mạng được tính bằng kích thước bảng xác suất điều kiện lớn nhất trong mạng. Với mạng dạng polytree, thứ tự loại trừ tối ưu là bắt đầu từ các nút gốc và chuyển động dần về phía nút con, luôn loại trừ các nút không còn nút cha mẹ trước.

Trong trường hợp loại trừ biến tạo ra thừa số lớn, dẫn tới độ phức tạp tính toán lớn, phương pháp loại trừ biến có thể không áp dụng được, và cần dùng suy diễn xấp xỉ bằng cách lấy mẫu như đã trình bầy ở trên.

4.6. ỨNG DỤNG SUY DIỄN XÁC SUẤT

Có rất nhiều ứng dụng khác nhau của suy diễn xác suất, cụ thể là ứng dụng suy diễn trên mạng Bayes được sử dụng trong thực tế. Có thể kể ra một số ứng dụng tiêu biểu như chẩn đoán bệnh (hệ thống Pathfinder), hệ trợ giúp thông minh (Microsoft Office assistant, có tên là Clippy), phương pháp xác định chức năng gen và protein, hệ thống giúp phát hiện và khắc phục lỗi. Dưới đây ta sẽ xem xét một ví dụ ứng dụng mạng Bayes trong chẩn đoán và khắc phục lỗi (troubleshooting) và một ví dụ chuẩn đoán y tế đơn giản.

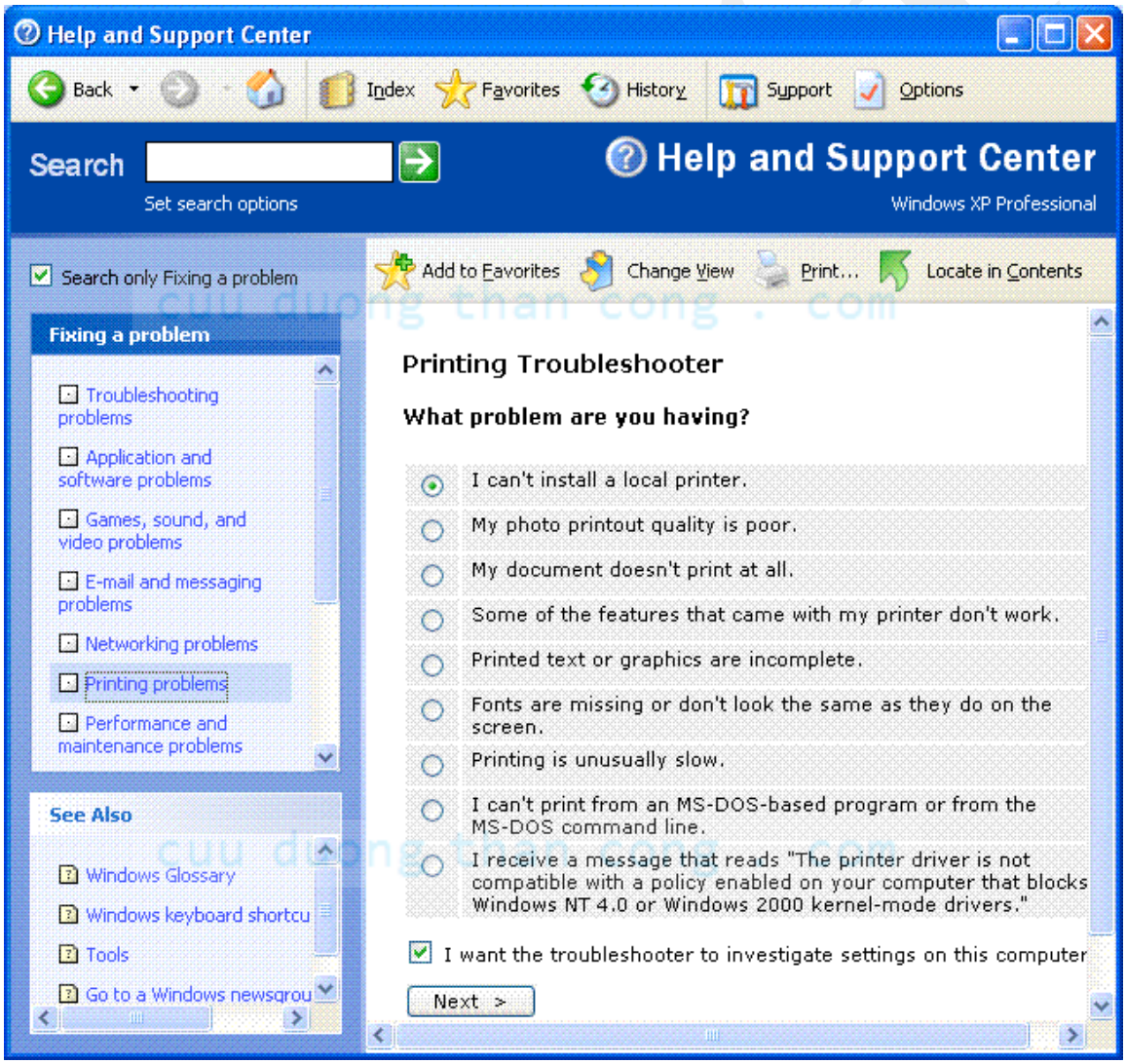
a) Chẩn đoán và khắc phục lỗi

Chẩn đoán và khắc phục lỗi là công việc tương đối phức tạp, cần có sự trợ giúp thông minh. Một hệ trợ giúp thường gặp dạng này là hệ thống troubleshooting của hệ điều hành Windows. Trên hình dưới đây là một màn hình giao diện của hệ thống này. Khi gặp sự cố,

chẳng hạn sự cố với máy in, người dùng khởi động hệ thống troubleshooting. Hệ thống đặt ra một số câu hỏi và yêu cầu trả lời. Tùy theo câu trả lời nhận được, hệ thống tìm cách xác định nguyên nhân và cách giải quyết. Câu trả lời càng cụ thể và càng chính xác thì nguyên nhân cũng được xác định càng cụ thể.

Mạng Bayes có thể được sử dụng cho phần suy diễn của hệ thống troubleshooting. Cụ thể, mạng Bayes cho phép biểu diễn quan hệ giữa ba dạng biến ngẫu nhiên: các dạng hỏng hóc F của thiết bị, hành động A cho phép khắc phục hỏng hóc, và câu hỏi Q cho phép thu thập thông tin về sự cố.

Xét ví dụ cụ thể sau: máy in in quá mờ. Có thể có rất nhiều hỏng hóc hay nguyên nhân dẫn tới in mờ. Để đơn giản, giả sử có bốn nguyên nhân sau: F1 – lỗi hộp mực, F2 – còn ít mực, F3 - luồng dữ liệu bị hỏng, F4 – chọn driver không đúng. Một số hành động cho phép khắc phục (một phần) nguyên nhân sau là: A1 – lắc ống mực, A2 – dùng ống mực khác, A3 – tắt và bật lại máy in.

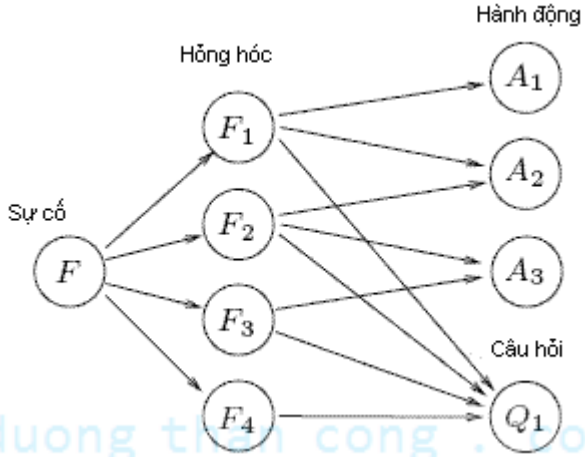


Hình 4.18. Màn hình hệ thống troubleshooting của Windows

Mỗi hành động cho phép khắc phục một hỏng hóc với một xác suất nào đó, ví dụ $P(A_2 = yes \mid F_1) = P(A_2 = yes \mid F_2) = 0,9$, tức là hành động dùng ống mực khác cho phép khắc phục lỗi hộp mực hay còn ít mực với xác suất 90%, trong khi $P(A_2 = yes \mid F_4) = 0$.

Trong quá trình khắc phục lỗi, hệ thống đặt câu hỏi cho người dùng để có thêm thông tin về hỏng hóc. Ví dụ, nếu người dùng trả lời "không" cho câu hỏi "Trang in thử có bị mờ không?" thì hệ thống có thể loại bỏ nguyên nhân hỏng hóc liên quan tới hộp mực. Hệ thống sử dụng các bảng xác suất điều kiện $P(Q_i = yes \mid F_j)$ cho phần suy diễn này.

Giữa các biến F, A, và Q có một số quan hệ độc lập xác suất. Cụ thể, hành động A và câu hỏi Q độc lập với nhau khi đã biết F. Ngoài ra, có thể giả thiết tại mỗi thời điểm chỉ xảy ra một hỏng hóc duy nhất. Quan hệ độc lập xác suất giữa các biến được mô hình hóa bằng mạng Bayes trên hình 4.19.



Hình 4.19. Mạng Bayes dùng cho khắc phục sự cố

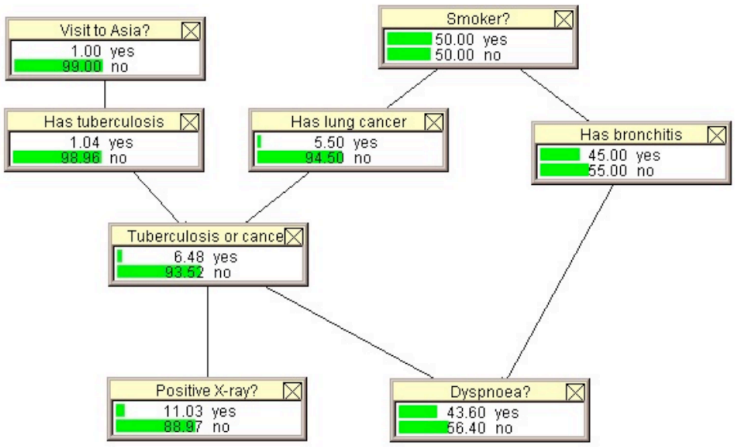
Tùy thông tin có được, hệ thống có thể thực hiện suy diễn để tính xác suất lựa chọn một hành động cụ thể nào đó.

b) Chẩn đoán y tế

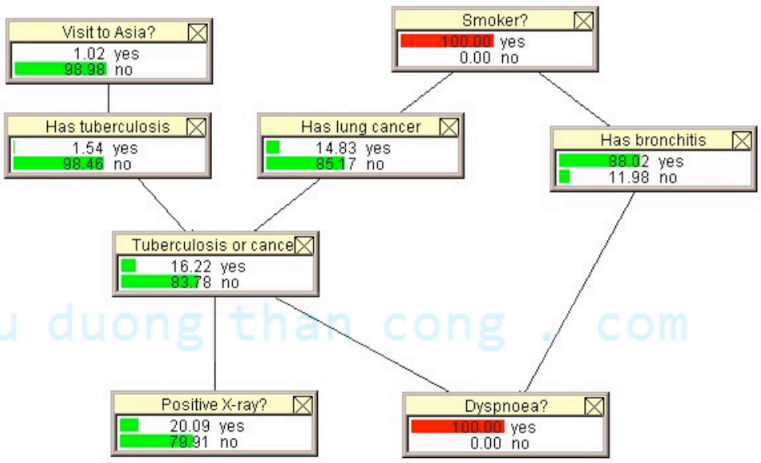
Một nhóm ứng dụng thường được nhắc đến khác của mạng Bayes là chẩn đoán y tế. Sau đây là một ví dụ đơn giản minh hoạ cho việc chẩn đoán một số bệnh về phổi và hô hấp. Trong ví dụ đơn giản này cần đưa ra các chẩn đoán liên quan tới ba bệnh: viêm phế quản, lao, và ung thư phổi. Các dữ kiện có liên quan bao gồm: đã tới Châu Á (bệnh nhân là người Âu, Mỹ), hút thuốc, Ảnh X-quang, khó thở. Mối liên hệ giữa các yếu tố này và các bệnh được mã hoá bằng mạng Bayes như trên hình 4.20.

Các xác suất trên hình 4.20 là xác suất cho trường hợp ta không có thêm bất cứ thông tin nào về người bệnh. Thông tin từ mạng cho thấy, xác suất một người bị lao hoặc ung thư phổi là nhỏ (1.04 và 5.5%), trong khi xác suất viêm phế quản là 45%.

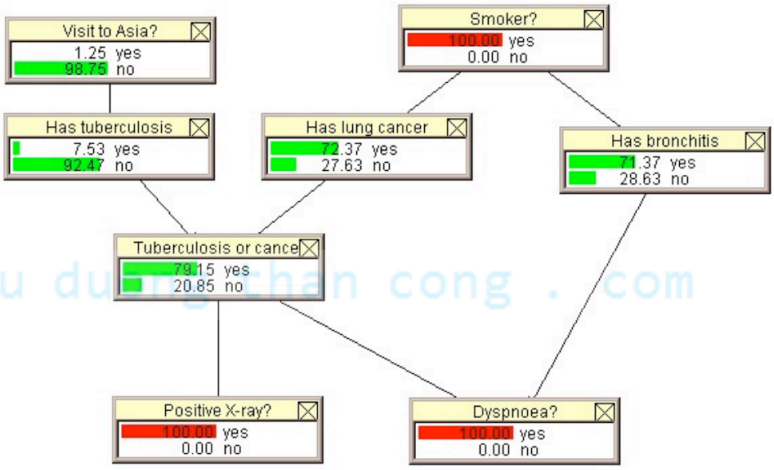
Lấy ví dụ một trường hợp cụ thể. Giả sử một bệnh nhân có biểu hiện khó thở đồng thời nghiện thuốc. Hình 4.21 thể hiện xác suất hai sự kiện hút thuốc và khó thở là 100% đồng thời cập nhật xác suất các bệnh trong trường hợp có thêm hai thông tin này. Mặc dù xác suất cả ba bệnh đều tăng, nhưng tăng nhiều nhất là xác suất viêm phế quản, từ 45% lên 88.02%. Như vậy, nếu không có thêm thông tin gì, có thể kết luận bệnh nhân viêm phế quản.



Hình 4.20. Mạng Bayes chẩn đoán bệnh hô hấp



Hình 4.21. Trường hợp có triệu chứng khó thở và bệnh nhân hút thuốc



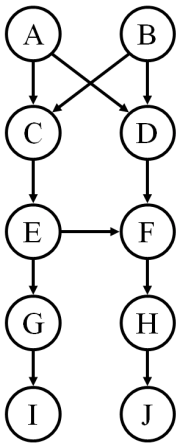
Hình 4.22. Trường hợp khó thở, hút thuốc, và ảnh chụp X-quang dương tính

Tiếp theo, để có thêm thông tin, bệnh nhân được chụp X-quang và cho kết quả không bình thường. Hình 4.22 thể hiện xác suất được cập nhật khi có thêm thông tin này. Có thể thấy xác suất ung thư phổi tăng mạnh và có giá trị xấp xỉ xác suất viêm phế quản.

Như vậy, qua ví dụ đơn giản này, ta có thể thấy khả năng mã hoá thông tin về quan hệ giữa các bệnh với các triệu chứng và yếu tố khác của mạng Bayes, cũng như sự thay đổi xác suất bệnh theo mức độ xuất hiện các bằng chứng.

4.7. CÂU HỎI VÀ BÀI TẬP CHƯƠNG

1. Cho cấu trúc mạng Bayes như sau:



Hãy xác định các phát biểu sau đúng hay sai:

a) C và D bị *d*-phân cách
b) C và D bị *d*-phân cách bởi A
c) C và D bị *d*-phân cách bởi {A, B}
d) C và D bị *d*-phân cách bởi {A, B, J}
e) C và D bị *d*-phân cách bởi {A, B, E, J}

2. Giả sử có hai phương pháp xét nghiệm A và B độc lập với nhau và cho phép phát hiện cùng một loại virus. Phương pháp A cho phép phát hiện 95% trường hợp nhiễm virus thật nhưng lại cho kết quả dương tính đối với 10% số người không có virus. Phương pháp B chỉ phát hiện được 90% trường hợp nhiễm virus thật nhưng chỉ cho kết quả dương tính sai với 5% số người không nhiễm virus. Biết rằng xác suất nhiễm virus trong cộng đồng dân cư là 1%. Giả sử một người xét nghiệm bằng một trong hai phương pháp trên và có kết quả dương tính với virus. Trong trường hợp sử dụng phương pháp nào thì kết quả đáng tin cậy hơn (xác suất người đó nhiễm virus thật cao hơn) ? Trình bầy phương pháp tính xác suất cụ thể cho câu trả lời.
3. Nam báo cáo cô giáo đã làm bài tập nhưng quên vở ở nhà. Từ kinh nghiệm giảng dậy của mình, cô giáo biết rằng chỉ 1% số sinh viên đã làm bài tập quên vở và báo cáo với cô giáo như vậy. Trong khi đó, một nửa số sinh viên chưa làm bài tập sẽ báo cáo quên vở. Thống kê cũng cho thấy số sinh viên làm bài tập chiếm 90% sinh viên cả lớp. Hãy tính xác suất Nam nói thật.

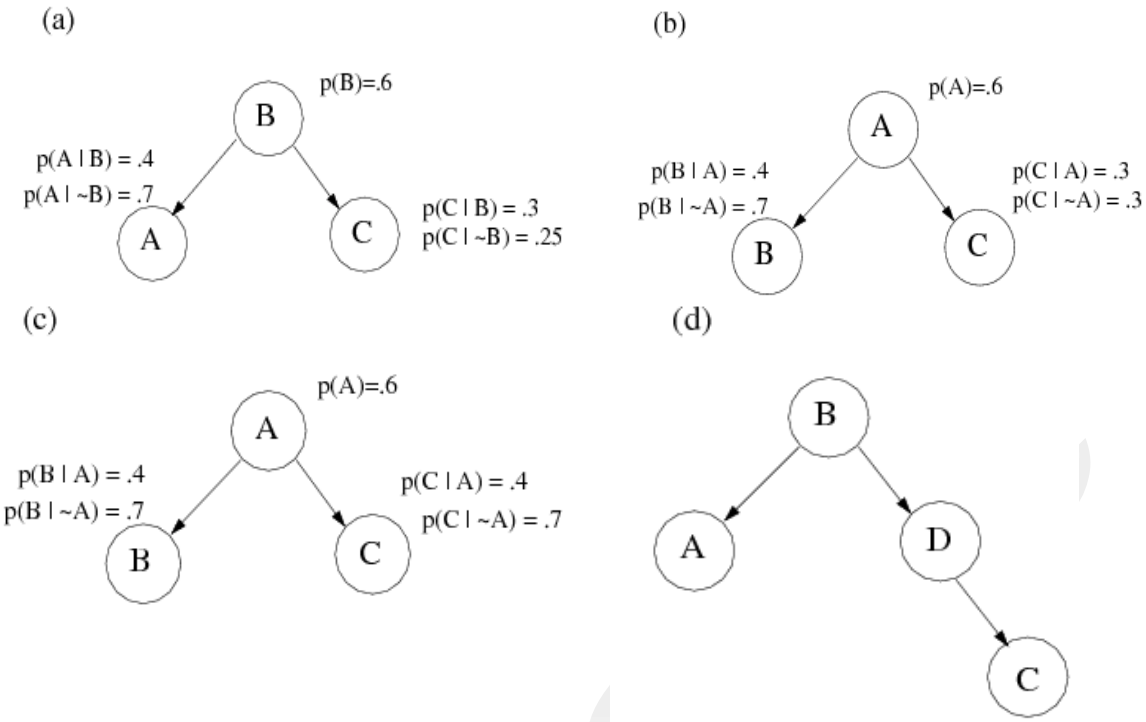
4. Hãy chứng minh công thức

   $P(X, Y \mid Z) = P(X \mid Z)\, P(Y \mid Z)$

   tương đương với mỗi công thức sau

   $P(X \mid Y, Z) = P(X \mid Z)$ và $P(Y \mid X, Z) = P(Y \mid Z)$

5. Sử dụng thông tin cho trên mạng Bayes, xác định xem A và C có độc lập với nhau không trong 4 trường hợp sau

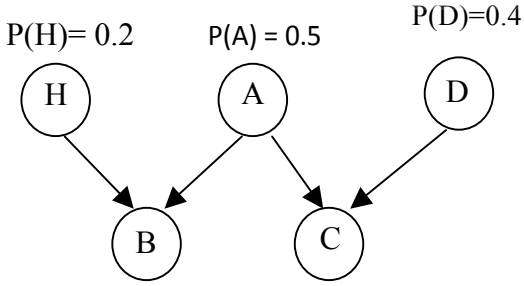


6. Giả sử cần suy diễn về quan hệ giữa thời tiết và giao thông. Cho ba biến ngẫu nhiên W, A, C biểu diễn cho ba tình huống sau: "thời tiết xấu" (W), "Chuyến bay Hà nội – HCM bị chậm" (A), "Quốc lộ 1 bị tắc" (C). Tiếp theo, giả sử chuyến bay chậm và đường tắc không ảnh hưởng đến nhau trong bất cứ thời tiết nào. Quan sát cho thấy, khi thời tiết xấu có 80% chuyến bay bị chậm và khi thời tiết tốt chỉ có 40% bị chậm. Tương tự, tần suất tắc đường 1 khi thời tiết xấu là 30% và khi thời tiết tốt là 10%. Xác suất thời tiết xấu tại Việt nam là 20%.

   a) Vẽ mạng Bayes và bảng xác suất điều kiện cho ví dụ này.

   b) Tính các xác suất $P(\neg A, W, C)$; $P(A, C)$

7. Cho mạng Bayes sau, các biến có thể nhận giá trị {T,F} ({true, false})



H	A	P(B=T|A,H)
F	F	0.7

A	D	P(C=T|A,D)
F	F	0.8

F	T	0.2
T	F	0.1
T	T	0.5

F	T	0.3
T	F	0.4
T	T	0.2

a) Tính xác suất cả năm biến cùng nhận giá trị F.

b) Tính P(A|C).

c) Theo mạng đã cho H và B có độc lập xác suất với nhau không ?

CHƯƠNG 5: HỌC MÁY

5.1. KHÁI NIỆM HỌC MÁY

5.1.1. Học máy là gì

Trong những chương trước ta đã xem xét một số kỹ thuật giải quyết vấn đề và suy diễn. Đặc điểm chung của những kỹ thuật này là phương pháp giải quyết vấn đề cùng với tri thức được cung cấp cho chương trình máy tính từ trước và chương trình không thể thay đổi hoặc cải thiện hoạt động của mình dựa trên kinh nghiệm thu được trong quá trình làm việc sau đó. Trong chương này, ta sẽ xem xét các kỹ thuật *học máy* (machine learning), là kỹ thuật cho phép giải quyết vấn đề hoặc ra quyết định dựa trên dữ liệu và kinh nghiệm.

Học máy là khả năng của chương trình máy tính sử dụng kinh nghiệm, quan sát, hoặc dữ liệu trong quá khứ để cải thiện công việc của mình trong tương lai thay vì chỉ thực hiện theo đúng các quy tắc đã được lập trình sẵn. Chẳng hạn, máy tính có thể học cách dự đoán dựa trên các ví dụ, hay học cách tạo ra các hành vi phù hợp dựa trên quan sát trong quá khứ.

Ví dụ. Xét một số ví dụ sau. Ví dụ thứ nhất là học cách đánh cờ. Chương trình có thể quan sát các ván cờ cùng với kết quả (thắng hay thua) để cải thiện khả năng chơi cờ và tăng số ván thắng trong tương lai. Trong trường hợp này, kinh nghiệm là các ván cờ trong quá khứ (có thể là ván cờ chương trình tự chơi với chính mình), được sử dụng để học cách làm tốt hơn công việc chơi cờ với tiêu chí đánh giá là số ván thắng.

Ví dụ thứ hai là học nhận dạng các ký tự. Chương trình được cung cấp dữ liệu dưới dạng ảnh chụp các ký tự (chữ cái) cùng mã UNICODE của ký tự đó. Sau khi học, chương trình cần có khả năng nhận dạng các ảnh chụp ký tự mới, tức là xác định được mã UNICODE của các ảnh mới chụp ký tự đã được học.

Tương tự quá trình học thông thường, một hệ thống học máy cần có khả năng ghi nhớ, thích nghi, và đặc biệt là tổng quát hóa. Tổng quát hóa là khả năng của hệ thống học máy ra quyết định chính xác trong các trường hợp mới, chưa gặp, dựa trên kinh nghiệm học được từ dữ liệu hoặc các quan sát trước đó.

Lý do cần tới học máy

Học máy là một nhánh nghiên cứu rất quan trọng của trí tuệ nhân tạo với khá nhiều ứng dụng thành công trong thực tế. Hiện nay, học máy là một trong những lĩnh vực phát triển mạnh nhất của trí tuệ nhân tạo. Có một số lý do giải thích cho sự cần thiết và phát triển của học máy:

- Thứ nhất, rất khó xây dựng hệ thống thông minh có thể thực hiện các công việc liên quan đến trí tuệ như thị giác máy, xử lý ngôn ngữ tự nhiên mà không sử dụng tới kinh nghiệm và quá trình học. Thông thường, khi viết chương trình, cần có thuật toán rõ ràng để chuyển đổi đầu vào thành đầu ra. Tuy nhiên, trong nhiều bài toán, rất khó để xây dựng được thuật toán như vậy. Như trong ví dụ về nhận dạng chữ ở trên, người bình thường có khả năng nhận dạng các chữ rất tốt nhưng rất khó để giải thích vì sao từ đầu vào là ảnh lại kết luận được đây là ký tự cụ thể nào. Học máy cho phép tìm ra giải pháp cho những trường hợp như vậy dựa trên dữ liệu, chẳng hạn bằng cách tìm ra điểm chung và riêng giữa rất nhiều ảnh chụp các ký tự.

- Thứ hai, nhiều ứng dụng đòi hỏi chương trình máy tính phải có khả năng thích nghi. Ví dụ, hành vi mua sắm của khách hàng có thể thay đổi theo thời điểm cụ thể trong ngày, trong năm, hoặc theo tuổi tác. Việc xây dựng thuật toán cố định cho những ứng dụng cần thích nghi và thay đổi là không phù hợp. Học máy mang lại khả năng thích nghi nhờ phân tích dữ liệu thu thập được.
- Thứ ba, việc tìm được chuyên gia và thu thập được tri thức cần thiết cho việc thiết kế thuật toán để giải quyết các vấn đề tương đối khó, trong khi dữ liệu ngày càng nhiều và có thể thu thập dễ dàng hơn. Khả năng lưu trữ và tính toán của máy tính cũng ngày càng tăng, cho phép thực hiện thuật toán học máy trên dữ liệu có kích thước lớn.
- Cuối cùng, bản thân khả năng học là một hoạt động trí tuệ quan trọng của con người, do vậy học tự động hay học máy luôn thu hút được sự quan tâm khi xây dựng hệ thống thông minh.

5.1.2. Ứng dụng của học máy

Có rất nhiều ứng dụng thực tế khác nhau của học máy. Hai lĩnh vực ứng dụng lớn nhất của học máy là *khai phá dữ liệu* (data mining) và *nhận dạng mẫu* (pattern recognition).

Khai phá dữ liệu là ứng dụng kỹ thuật học máy vào các cơ sở dữ liệu hoặc các tập dữ liệu lớn để phát hiện quy luật hay tri thức trong dữ liệu đó hoặc để dự đoán các thông tin quan tâm trong tương lai. Ví dụ, từ tập hợp hóa đơn bán hàng có thể phát hiện ra quy luật "những người mua bánh mì thường mua bơ".

Nhận dạng mẫu là ứng dụng các kỹ thuật học máy để phát hiện các mẫu có tính quy luật trong dữ liệu, thường là dữ liệu hình ảnh, âm thanh. Bài toán nhận dạng mẫu cụ thể thường là xác định nhãn cho đầu vào cụ thể, ví dụ cho ảnh chụp mặt người, cần xác định đó là ai.

Cần lưu ý, khai phá dữ liệu và nhận dạng mẫu có nhiều điểm trùng nhau cả trong phạm vi nghiên cứu và ứng dụng. Điểm khác nhau chủ yếu liên quan tới lĩnh vực ứng dụng và kỹ thuật sử dụng, theo đó khai phá dữ liệu liên quan tới dữ liệu thương mại trong khi nhận dạng mẫu liên quan nhiều tới dữ liệu âm thanh, hình ảnh và được dùng nhiều trong kỹ thuật.

Ứng dụng cụ thể

Sau đây là một số ví dụ ứng dụng cụ thể của học máy:

- Nhận dạng ký tự: phân loại hình chụp ký tự thành các loại, mỗi loại ứng với một ký tự tương ứng.
- Phát hiện và nhận dạng mặt người: phát hiện vùng có chứa mặt người trong ảnh, xác định đó là mặt người nào trong số những người đã có ảnh trước đó, tức là phân chia ảnh thành những loại tương ứng với những người khác nhau.
- Lọc thư rác, phân loại văn bản: dựa trên nội dung thư điện tử, chia thư thành loại "thư rác" hay "thư bình thường"; hoặc phân chia tin tức thành các thể loại khác nhau như "xã hội", "kinh tế", "thể thao".v.v.
- Dịch tự động: dựa trên dữ liệu huấn luyện dưới dạng các văn bản song ngữ, hệ thống dịch tự động học cách dịch từ ngôn ngữ này sang ngôn ngữ khác. Hệ thống dịch tự động tiêu biểu dạng này là Google Translate.
- Chẩn đoán y tế: học cách dự đoán người bệnh có mắc hay không mắc một số bệnh nào đó dựa trên triệu chứng quan sát được.

- Phân loại khách hàng và dự đoán sở thích: sắp xếp khách hàng vào một số loại, từ đây dự đoán sở thích tiêu dùng của khách hàng.
- Dự đoán chỉ số thí trường: căn cứ giá trị một số tham số hiện thời và trong lịch sử, đưa ra dự đoán, chẳng hạn dự đoán giá chứng khoán, giá vàng.v.v.
- Các hệ khuyến nghị, hay hệ tư vấn lựa chọn: cung cấp một danh sách ngắn các loại hàng hóa, phim, video, tin tức v.v. mà người dùng nhiều khả năng quan tâm. Ví dụ ứng dụng loại này là phần khuyến nghị trên Youtube hay trên trang mua bán trực tuyến Amazon.
- Ứng dụng lái xe tự động: dựa trên các mẫu học chứa thông tin về các tình huống trên đường, hệ thống học máy cho phép tự ra quyết định điều khiển xe, và do vậy không cần người lái. Hiện Google đã có kế hoạch thương mại hóa xe ôtô tự động lái như vậy.

5.1.3. Các dạng học máy

Khi thiết kế và xây dựng hệ thống học máy cần quan tâm tới những yếu tố sau.

- Thứ nhất, kinh nghiệm hoặc dữ liệu cho học máy được cho dưới dạng nào?
- Thứ hai, lựa chọn biểu diễn cho hàm đích ra sao? Hàm đích có thể biểu diễn dưới dạng hàm đại số thông thường nhưng cũng có thể biểu diễn dưới những dạng khác như dạng cây, dạng mạng nơ ron, công thức xác suất .v.v.

Việc sử dụng những dạng kinh nghiệm và dạng biểu diễn khác nhau dẫn tới những dạng học máy khác nhau. Có ba dạng học máy chính như sau:

- **Học có giám sát** (supervised learning). Là dạng học máy trong đó cho trước tập dữ liệu huấn luyện dưới dạng các ví dụ cùng với *giá trị đầu ra* hay *giá trị đích*. Dựa trên dữ liệu huấn luyện, thuật toán học cần xây dựng mô hình hay hàm đích để dự đoán giá trị đầu ra (giá trị đích) cho các trường hợp mới.
  - Nếu giá trị đầu ra là *rời rạc* thì học có giám sát được gọi là *phân loại* hay *phân lớp* (classification).
  - Nếu đầu ra nhận giá trị *liên tục*, tức đầu ra là số thực, thì học có giám sát được gọi là *hồi quy* (regression). Trong phần tiếp theo, ta sẽ xem xét chi tiết hơn về học có giám sát.
- **Học không giám sát** (un-supervised learning). Là dạng học máy trong đó các ví dụ được cung cấp nhưng không có giá trị đầu ra hay giá trị đích.
  - Thay vì xác định giá trị đích, thuật toán học máy dựa trên độ tương tự giữa các ví dụ để xếp chúng thành những nhóm, mỗi nhóm gồm các ví dụ tương tự nhau. Hình thức học không giám sát như vậy gọi là *phân cụm* (clustering). Ví dụ, chỉ bằng cách quan sát hoặc đo chiều cao của mọi người, dần dần ta học được khái niệm "người cao" và "người thấp", và có thể xếp mọi người vào hai cụm tương ứng.
  - Ngoài phân cụm, một dạng học không giám sát phổ biến khác là phát hiện *luật kết hợp* (association rule). Luật kết hợp có dạng $P(A \mid B)$, cho thấy xác suất hai tính chất A và B xuất hiện cùng với nhau. Ví dụ, qua phân tích dữ

liệu mua hàng ở siêu thị, ta có luật P(Bơ | Bánh mỳ) =80%, có nghĩa là 80% những người mua bánh mỳ cũng mua bơ.

- **Học tăng cường** (reinforcement learning)[4]. Đối với dạng học này, kinh nghiệm không được cho trực tiếp dưới dạng đầu vào/đầu ra cho mỗi trạng thái hoặc mỗi hành động. Thay vào đó, hệ thống nhận được một giá trị khuyến khích (reward) là kết quả cho một chuỗi hành động nào đó. Thuật toán cần học cách hành động để cực đại hóa giá trị khuyến khích. Ví dụ của học khuyến khích là học đánh cờ, trong đó hệ thống không được chỉ dẫn nước đi nào là hợp lý cho từng tình huống mà chỉ biết kết quả toàn ván cờ. Như vậy, các chỉ dẫn về nước đi được cho một cách gián tiến và có độ trễ dưới dạng giá trị thưởng. Nước đi tốt là nước đi nằm trong một chuỗi các nước đi dẫn tới kết quả thắng toàn bộ ván cờ.

Trong các dạng học máy, học có giám sát là dạng phổ biến, có nhiều thuật toán liên quan và nhiều ứng dụng nhất. Phần còn lại của chương sẽ tập trung chủ yếu vào dạng học máy này.

5.1.4. Học có giám sát

Như đã nói ở trên, học có giám sát là trường hợp dữ liệu huấn luyện được cho một cách tường minh dưới dạng đầu vào và đầu ra của hàm đích, ví dụ, cho trước tập các mẫu cùng nhãn phân loại tương ứng. Học máy khi đó được gọi là *có giám sát* để thể hiện việc thuật toán nhận được chỉ dẫn trực tiếp về lời giải cho từng trường hợp. Học có giám sát bao gồm *phân loại* và *hồi quy*. Phân loại là dạng học có giám sát với hàm đích nhận giá trị rời rạc và hồi quy là học có giám sát với hàm đích nhận giá trị liên tục. Trong phần này ta sẽ xem xét kỹ hơn về dạng học thông dụng này.

Phân loại. Giả sử cho dữ liệu về một số loại ô tô con như trong bảng trên hình 5.1. Mỗi dòng trong bảng tương ứng với một loại xe cụ thể gồm hai thông số về xe là dung tích động cơ và loại xe. Ngoài ra, mỗi xe còn được xếp loại "cao cấp" hay "trung bình". Đây là dữ liệu huấn luyện tiêu biểu cho bài toán phân loại, trong đó mỗi ví dụ được gán một giá trị đích có thể nhận giá trị từ một trong hai giá trị "cao cấp" hoặc "trung bình". Trong trường hợp nói chung, số giá trị đích có thể nhiều hơn hai, nhưng giá trị đích vẫn là rời rạc và hữu hạn. Chẳng hạn, mỗi xe có thể xếp vào một trong ba loại "cao cấp", "trung bình", "bình dân" Nhiệm vụ của bài toán phân loại khi đó là tổng quát hóa trên dữ liệu huấn luyện đã cho. Từ đó xác định phân khúc xe cho các mẫu xe mới nếu biết dung tích động cơ và loại xe.

Dung tích động cơ	Loại xe	Phân khúc
3200	Sedan	Cao cấp
2500	Sedan	Cao cấp

[4] Khái niệm "reinforcement" được dùng trong tâm lý học với nghĩa khuyến khích, kích thích, thưởng. Chúng tôi tạm dịch sang tiếng Việt là tăng cường nhưng cũng có thể dịch là thưởng hoặc có tài liệu dịch reinforcement learning là "học khuyến khích"

2500	SUV	Trung bình
2000	Sedan	Trung bình
3500	SUV	Cao cấp
1800	Sedan	Trung bình

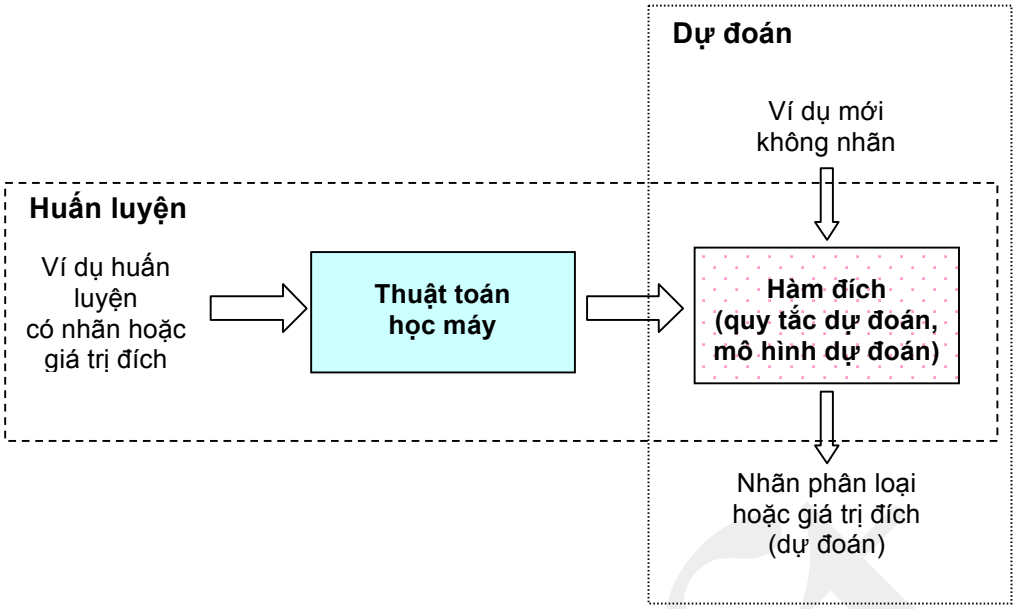
Hình 5.1. Dữ liệu huấn luyện cho bài toán phân loại

Hồi quy. Trong trường hợp bài toán hồi quy, dữ liệu huấn luyện thường có dạng như trong bảng trên hình 5.2, trong đó mỗi dòng ứng với một chiếc xe cũ, giá trị đích là giá bán xe được cho trong cột cuối cùng. Khác biệt chính so với trường hợp phân loại là giá trị đích ở đây (giá bán) là số thực và có thể nhận các giá trị liên tục. Nhiệm vụ của bài toán hồi quy là sử dụng dữ liệu huấn luyện để xây dựng mô hình hồi quy và dùng mô hình này để dự đoán giá bán cho các xe khác nếu biết dung tích động cơ và tuổi của xe.

Dung tích động cơ	Tuổi của xe	Giá bán (triệu đồng)
3200	1	2500
2500	2	1600
2500	4	1300
2000	5	600
1800	1	915
1800	3	725

Hình 5.2. Dữ liệu huấn luyện cho trường hợp hồi quy

Huấn luyện và dự đoán. Hoạt động của một hệ thống học máy điển hình cho trường hợp học có giám sát bao gồm hai giai đoạn: giai đoạn *huấn luyện* và giai đoạn *dự đoán*, như thể hiện trên hình 5.3. Trong giai đoạn huấn luyện, dữ liệu huấn luyện được sử dụng để huấn luyện các mô hình dự đoán, còn gọi là các hàm đích. Việc huấn luyện mô hình được thực hiện nhờ thuật toán học máy bao gồm các thuật toán phân loại hoặc hồi quy. Mô hình dự đoán sau đó được lưu lại. Trong giai đoạn dự đoán, hệ thống nhận các ví dụ mới chưa có giá trị đích và sử dụng mô hình dự đoán đã huấn luyện để xác định giá trị đích cho ví dụ mới.



Hình 5.3. Một hệ thống học máy tiểu biểu cho trường hợp học có giám sát với hai giai đoạn: 1) Huấn luyện; 2) Dự đoán

Sau đây là một số khái niệm được sử dụng khi trình bày về học có giám sát.

Mẫu hay *ví dụ* là từ dùng để chỉ đối tượng cần phân loại. Chẳng hạn, khi lọc thư rác, mỗi thư được gọi là một mẫu hay một ví dụ. Trong các ví dụ ở hình 5.2, mỗi xe ô tô là một ví dụ.

Ví dụ được mô tả bằng một tập các *thuộc tính*, còn được gọi là *đặc trưng* hay *biến*. Ví dụ, trong bài toán chẩn đoán bệnh, thuộc tính là những triệu chứng của người bệnh và các tham số khác: cân nặng, huyết áp, v.v. Trong ví dụ trên hình 5.1, ta có hai thuộc tính là dung tích động cơ và loại xe. Thuộc tính có thể nhận giá trị là số (như dung tích động cơ) hoặc giá trị rời rạc (như loại xe) trong ví dụ trên hình 5.1.

Nhãn phân loại thể hiện loại của đối tượng mà ta cần dự đoán. Đối với trường hợp phân loại thư rác, nhãn phân loại có thể là "rác" hay "bình thường". Trong ví dụ trên hình 5.1, nhãn phân loại cho biết phân khúc của xe và có thể nhận giá trị "cao cấp" hay "trung bình". Trong giai đoạn học hay còn gọi là giai đoạn huấn luyện, thuật toán học được cung cấp cả nhãn phân loại của mẫu, trong giai đoạn dự đoán, thuật toán chỉ nhận được các mẫu không nhãn và cần xác định nhãn cho những mẫu này.

Trong bài toán phân loại, nếu nhãn phân loại chỉ có thể nhận một trong hai giá trị thì bài toán gọi là *phân loại hai lớp* hay *phân loại nhị phân*. Nếu có nhiều hơn hai loại thì gọi là bài toán *phân loại đa lớp*.

Dữ liệu huấn luyện. Dữ liệu huấn luyện được cho nhưng trong hình 5.1 đối với phân loại, hay trên hình 5.2 đối với hồi quy. Mỗi tập dữ liệu huấn luyện gồm nhiều ví dụ, mỗi ví dụ được mô tả trong một dòng. Mỗi ví dụ huấn luyện được ký hiệu bằng một cặp $(\mathbf{x}_i, y_i)$, trong đó $\mathbf{x}_i$ là vec tơ các đặc trưng hay thuộc tính, y_i là giá trị nhãn phân loại hay giá trị đích trong hồi quy. Như vậy $\mathbf{x}_i$ là đầu vào, còn y_i là đầu ra cần dự đoán. Trong giai đoạn huấn luyện, thuật toán được cung cấp giá trị y_i ; trong giai đoạn dự đoán, mô hình được sử dụng để dự đoán giá trị này.

Kết quả học thường được thể hiện dưới dạng một ánh xạ từ mẫu sang nhãn phân loại gọi là *mô hình* học. Mô hình này được thể hiện dưới dạng một hàm gọi là *hàm đích* (target function) có dạng $f : X \to C$, trong đó X là không gian các ví dụ và C là tập các nhãn phân loại khác nhau. Tùy vào phương pháp cụ thể, hàm đích có thể rất khác nhau. Ví dụ, hàm đích có thể làm một hàm tuyến tính như trong hồi quy tuyến tính, là mô hình dạng cây trong học cây quyết định, là mạng nơ ron trong mạng nơ ron nhân tạo .v.v.

5.2. HỌC CÂY QUYẾT ĐỊNH

Sau khi làm quen với các khái niệm chung về học máy ở trên, trong phần này, ta sẽ làm quen với một kỹ thuật học máy cụ thể: *học cây quyết định*. Học cây quyết định là một trong phương pháp học máy tiêu biểu có nhiều ứng dụng trong phân loại và dự đoán. Mặc dù độ chính xác của phương pháp này không thật cao so với những phương pháp được nghiên cứu gần đây, học cây quyết định vẫn có nhiều ưu điểm như đơn giản, dễ lập trình, và cho phép biểu diễn hàm phân loại dưới dạng dễ hiểu, dễ giải thích cho con người. Phương pháp này thường được dùng như phương pháp mở đầu để minh họa cho kỹ thuật học bộ phân loại từ dữ liệu.

Phương pháp học cây quyết định được sử dụng cho việc học các hàm phân loại từ dữ liệu huấn luyện, trong đó cây quyết định được sử dụng làm biểu diễn xấp xỉ của hàm phân loại, tức là hàm có đầu ra là các giá trị rời rạc. Như đã nói ở trên, phương pháp học này thuộc loại học có giám sát.

Phần này sẽ giúp người đọc làm quen với khái niệm cây quyết định, đồng thời giới thiệu một số thuật toán học cây quyết định bao gồm ID3 và C4.5.

5.2.1. Khái niệm cây quyết định

Cây quyết định là một cấu trúc ra quyết định có dạng cây (xem hình 5.4). Cây quyết định nhận đầu vào là một bộ giá trị *thuộc tính* mô tả một đối tượng hay một tình huống và trả về một giá trị rời rạc. Mỗi bộ thuộc tính đầu vào được gọi là một *mẫu* hay một *ví dụ*, đầu ra gọi là *loại* hay *nhãn phân loại*. Thuộc tính đầu vào còn được gọi là đặc trưng và có thể nhận giá trị rời rạc hoặc liên tục. Để cho đơn giản, trước tiên ta sẽ xem xét thuộc tính rời rạc, sau đó sẽ mở rộng cho trường hợp thuộc tính nhận giá trị liên tục.

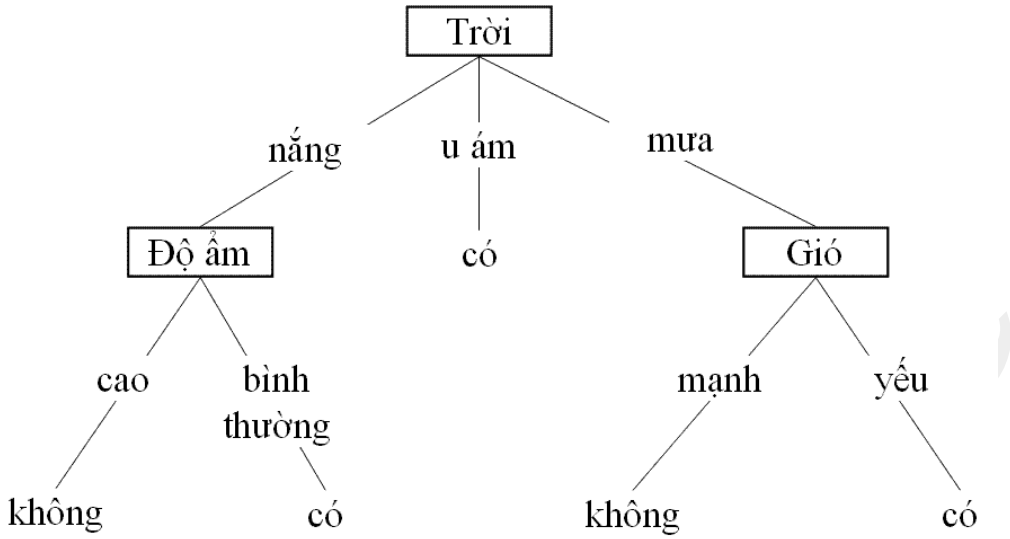
Trong các trình bày tiếp theo, tập thuộc tính đầu vào được cho dưới dạng véc tơ **x**, nhãn phân loại đầu ra được ký hiệu là y, cây quyết định là hàm $f(\mathbf{x})$ trả lại giá trị y. Để tiện minh họa, trong phần này sẽ sử dụng bộ dữ liệu được cho trên hình 5.5.

Cây quyết định được biểu diễn dưới dạng một cấu trúc cây (xem ví dụ trên hình 5.4). Mỗi nút trung gian, tức là nút không phải nút lá, tương ứng với phép kiểm tra một thuộc tính. Mỗi nhánh phía dưới của nút đó tương ứng với một giá trị của thuộc tính hay một kết quả của phép thử. Khác với nút trung gian, nút lá không chứa thuộc tính mà chứa nhãn phân loại.

Để xác định nhãn phân loại cho một ví dụ nào đó, ta cho ví dụ chuyển động từ gốc cây về phía nút lá. Tại mỗi nút, thuộc tính tương ứng với nút được kiểm tra, tùy theo giá trị của

thuộc tính đó mà ví dụ được chuyển xuống nhánh tương ứng bên dưới. Quá trình này lặp lại cho đến khi ví dụ tới được nút lá và được nhận nhãn phân loại là nhãn của nút lá tương ứng.

Xét ví dụ cây quyết định trên hình 5.4 được xây dựng cho bộ dữ liệu trong hình 5.5. Cây quyết định cho phép xác định (phân loại) các buổi sáng thành có (phù hợp) và không (phù hợp) cho việc chơi tennis tùy theo thời tiết trong ngày đó. Thời tiết mỗi ngày được mô tả thông qua bốn thuộc tính: Trời, Độ ẩm, Nhiệt độ, Gió.



Hình 5.4. Một ví dụ cây quyết định cho bài toán "Chơi tennis". Nút lá chứa nhãn phân loại "có chơi" hoặc "không chơi". Nút trung gian chứa thuộc tính thời tiết.

Giả sử ta có ví dụ < Trời = nắng, Nhiệt độ = cao, Gió = mạnh, Độ ẩm = cao>. Ví dụ sẽ được cây quyết định xếp xuống nút ngoài cùng bên trái và do vậy được xác định là "không chơi".

Có thể thấy cách biểu diễn hàm quyết định dưới dạng cây rất trực quan, dễ hiểu, dễ giải thích lý do ra quyết định về nhãn cho một ví dụ cụ thể nào đó.

Biểu diễn tương đương dưới dạng biểu thức logic

Cây quyết định có thể biểu diễn tương đương dưới dạng các quy tắc hay biểu thức logic.

Cây_quyết_định(**x**) $\Leftrightarrow$ (P_1(**x**) v P_2(**x**) v…v P_n(**x**))

trong đó mỗi P_i(**x**) là hội các phép thử thuộc tính theo đường đi từ gốc tới nút lá có giá trị dương (true). Cụ thể, cây quyết định trên hình 4.1 có thể biểu diễn tương đương dưới dạng:

(Trời = nắng ^ Độ ẩm = bình_thường)

v (Trời = u_ám)

v (Trời = mưa ^ Gió = yếu)

Hoặc biểu diễn dưới dạng các luật suy diễn Nếu … Thì… như sau:

Nếu (Trời = nắng) và (Độ ẩm = bình thường) Thì "có chơi"

Nếu (Trời = nắng) và (Độ ẩm = cao) Thì "không chơi"

Nếu (Trời = u ám) Thì "có chơi"

Nếu (Trời = mưa) và (Gió = mạnh) Thì "không chơi"

Nếu (Trời = mưa) và (Gió = yếu) Thì "có chơi".

Các luật dạng này rất gần gũi với cách diễn đạt của con người khi ra quyết định hay thực hiện phân loại.

5.2.2. Thuật toán học cây quyết định

Trước khi sử dụng cây quyết định, ta cần xây dựng hay "học" cây quyết định từ dữ liệu huấn luyện. Có nhiều thuật toán khác nhau được đề xuất và sử dụng để học cây quyết định từ dữ liệu, trong đó đa số dựa trên nguyên tắc chung là xây dựng cây theo kiểu tìm kiếm tham lam từ cây đơn giản tới cây phức tạp hơn. Phần này sẽ giới thiệu **thuật toán học cây ID3**, một thuật toán đơn giản nhưng có tính đại diện cho cách xây dựng cây như vậy. ID3 là viết tắt của từ Iterative Dichotomiser 3 (tạm dịch là phân chia lặp hay phân chia tuần tự). Thuật toán này do Ross Quinlan phát triển và có một phiên bản cải tiến gọi là **C4.5**.

Dữ liệu huấn luyện

Dữ liệu huấn luyện được cho dưới dạng n mẫu hay n ví dụ huấn luyện, mỗi ví dụ có dạng $(\mathbf{x}_i, y_i)$, trong đó $\mathbf{x}_i$ là véc tơ các thuộc tính và y_i là giá trị nhãn phân loại, chẳng hạn như ví dụ trong hình 5.5 với 14 ví dụ tương ứng với 14 dòng. Cột đầu tiên trong bảng chứa số thứ tự và không tham gia vào cây quyết định. Bốn cột tiếp theo chứa giá trị bốn thuộc tính. Cột ngoài cùng bên phải chứa nhãn phân loại. Đối với dữ liệu đang xét, nhãn phân loại là nhãn nhị phân, có thể nhận một trong hai giá trị "có" hoặc "không".

Ngày	Trời	Nhiệt độ	Độ ẩm	Gió	Chơi tennis
D1	nắng	Cao	Cao	yếu	không
D2	nắng	Cao	Cao	mạnh	không
D3	u ám	Cao	Cao	yếu	có
D4	Mưa	trung bình	cao	yếu	có
D5	Mưa	thấp	bình thường	yếu	có
D6	Mưa	thấp	bình thường	mạnh	không
D7	u ám	thấp	bình thường	mạnh	có
D8	nắng	trung bình	cao	yếu	không
D9	nắng	thấp	bình thường	yếu	có
D10	mưa	trung bình	bình thường	yếu	có
D11	nắng	trung bình	bình thường	mạnh	có

D12	u ám	trung bình	cao	mạnh	có
D13	u ám	Cao	bình thường	yếu	có
D14	mưa	trung bình	cao	mạnh	không

Hình 5.5. Bộ dữ liệu huấn luyện cho bài toán phân loại "Chơi tennis".

Thuật toán học cây ID3

Nhiệm vụ của thuật toán học là xây dựng cây quyết định phù hợp với tập dữ liệu huấn luyện, tức là cây quyết định có đầu ra giống (nhiều nhất) với nhãn phân loại cho trong tập mẫu. Trong trường hợp số thuộc tính nhỏ, việc xây dựng cây quyết định như vậy có thể thực hiện bằng cách liệt kê tất các cây quyết định hợp lệ và kiểm tra để chọn ra cây phù hợp với dữ liệu. Với số lượng thuộc tính lớn, số cây quyết định như vậy là rất lớn và không thể tìm kiếm theo kiểu vét cạn như vậy. Do đó, thuật toán học cây thường dựa trên nguyên tắc tham lam, xây dựng dần các nút từ trên xuống.

Quá trình xây dựng cây: Để bắt đầu, thuật toán học lựa chọn thuộc tính cho nút gốc. Thuộc tính được lựa chọn là thuộc tính cho phép *phân chia tốt nhất* các ví dụ thành những tập con, sao cho mỗi tập con càng đồng nhất càng tốt. Ở đây, đồng nhất được hiểu là các ví dụ có cùng nhãn phân loại. Sau khi lựa chọn được thuộc tính cho nút gốc, tập dữ liệu ban đầu sẽ được chia xuống các nhánh con do kết quả phép kiểm tra thuộc tính ở gốc. Với mỗi tập con dữ liệu, ta lại có một bài toán học cây dữ liệu mới và do vậy có thể lặp lại thủ tục ở trên với ít dữ liệu hơn và bớt đi một thuộc tính đã được sử dụng ở gốc.

Quá trình xây dựng cây quyết định được lặp đệ quy như vậy cho tới khi xẩy ra những tình huống sau:

- Sau khi phân chia tại một nút, tập dữ liệu con chỉ chứa các mẫu có cùng nhãn phân loại (chẳng hạn cùng dương hoặc cùng âm). Trong trường hợp này ta dừng quá trình phân chia ở đây, tạo một nút là và gán cho nút nhãn phân loại trùng với nhãn của các ví dụ tại nút đó. Trong ví dụ trên hình 5.4, nhánh giữa của nút gốc bao gồm các mẫu có nhãn "có" tạo thành nút lá.
- Tất cả các thuộc tính đã được sử dụng ở phía trên, trong khi tập dữ liệu con còn chứa cả nhãn dương và nhãn âm. Đây là trường hợp các ví dụ có cùng giá trị thuộc tính nhưng lại khác nhãn phân loại và xẩy ra do dữ liệu huấn luyện có chứa nhiễu hoặc do các thuộc tính hiện có không cung cấp đủ thông tin để xác định đúng nhãn phân loại. Trong trường hợp này, thuật toán sẽ chọn nhãn chiếm đa số trong tập con để gán cho nút.

Thuật toán học cây quyết định được cho trên hình 5.6.

Input: tập dữ liệu huấn luyện

Output: Cây quyết định

Khởi đầu: nút hiện thời là nút gốc chứa toàn bộ tập dữ liệu huấn luyện

a. Tại nút hiện thời n, lựa chọn thuộc tính:
   - Chưa được sử dụng ở nút tổ tiên (tức là nút nằm trên đường đi từ gốc tới nút hiện thời)
   - Cho phép phân chia tập dữ liệu hiện thời thành các tập con **một cách tốt nhất**

b. Với mỗi giá trị thuộc tính được chọn:
   - Thêm một nút con bên dưới
   - Chia các ví dụ ở nút hiện thời về các nút con theo giá trị thuộc tính được chọn

c. Lặp (đệ quy) với mỗi nút con cho tới khi:
   - Tất cả các thuộc tính đã được sử dụng ở các nút phía trên, hoặc
   - Tất cả ví dụ tại nút hiện thời có cùng nhãn phân loại
   - Nhãn của nút được lấy theo đa số nhãn của ví dụ tại nút hiện thời

Hình 5.6. Thuật toán xây dựng cây quyết định từ dữ liệu huấn luyện

Thuật toán được lặp đệ quy qua nhiều vòng. Tại mỗi bước lặp, thuật toán tìm cách lựa chọn một thuộc tính cho phép phân chia tập dữ liệu tại nút hiện thời một cách tốt nhất. Quá trình này được tiến hành cho tới khi đạt tới nút lá, tức là khi xảy ra một trong hai tình huống như đã nhắc tới ở trên.

Lựa chọn thuộc tính tốt nhất

Một điểm quan trọng trong thuật toán xây dựng cây quyết định là lựa chọn thuộc tính tốt nhất tại mỗi nút. Trong trường hợp lý tưởng, thuộc tính lựa chọn là thuộc tính cho phép chia tập dữ liệu thành các tập con có cùng một nhãn, và do vậy chỉ cần một phép kiểm tra thuộc tính khi phân loại. Trong trường hợp nói chung, thuộc tính lựa chọn cần cho phép tạo ta những tập con có *độ đồng nhất* cao nhất. Yêu cầu đặt ra là cần có cách đo độ đồng nhất của tập dữ liệu và mức tăng độ đồng nhất khi sử dụng một thuộc tính nào đó.

Thuật toán xây dựng cây ID3 sử dụng *entropy* làm mức đo độ đồng nhất của tập dữ liệu. Trên cơ sở entropy, thuật toán tính *độ tăng thông tin* như mức tăng độ đồng nhất, từ đây xác định thuộc tính tốt nhất tại mỗi nút.

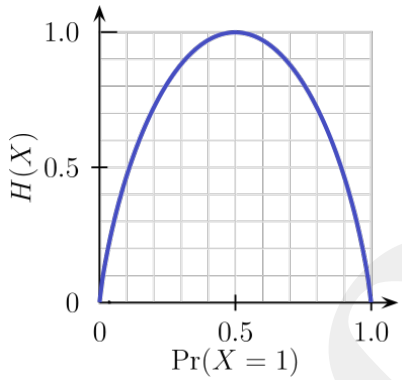
Trong trường hợp chỉ có hai nhãn phân loại, ký hiệu là + và -, entropy H(S) của tập dữ liệu S được tính như sau:

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

trong đó p+ và p- là xác suất quan sát thấy nhãn phân loại + và -, được tính bằng tần suất quan sát thấy + và – trong tập dữ liệu. Trong tập dữ liệu trong bảng trên hình 5.5, với 9 nhãn dương và 5 nhãn âm, ký hiệu [9+, 5-], ta có:

$$H([9+,5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$$

Trên hình 5.7 là đồ thị thể hiện sự phụ thuộc của giá trị entropy H vào giá trị xác suất trong trường hợp nhãn nhị phân. Có thể nhận thấy, trong trường hợp nhãn nhị phân, entropy đạt giá trị tối đa bằng 1 khi xác suất hai nhãn bằng nhau và bằng 0.5, entropy đạt giá trị nhỏ nhất bằng 0 khi xác suất một nhãn là 1 và nhãn còn lại là 0. Nói chung, entropy càng gần 0 thì tập đối tượng càng đồng nhất.



Hình 5.7: Giá trị của entropy $H(X)$ trong trường hợp X có thể nhận một trong hai giá trị {0,1}

Trong trường hợp tổng quát với C nhãn phân loại có xác suất lần lượt là p_1, p_2, …,p_C. entropy được tính như sau:

$$H(S) = -\sum_{i=1}^{C} p_i \log_2 p_i$$

Giá trị cực đại của entropy khi đó sẽ bằng $\log_2 C$ khi các nhãn có xác suất như nhau và giá trị nhỏ nhất của entropy bằng 0 khi tất cả đối tượng có chung một nhãn. Tương tự trường hợp nhãn nhị phân, giá trị entropy càng nhỏ thì tập dữ liệu càng đồng nhất.

Sử dụng entropy như độ đo mức đồng nhất của tập mẫu, ta có thể đánh giá độ tốt của thuộc tính bằng cách so sánh entropy trước và sau khi tập mẫu được phân chia thành tập con theo giá trị của thuộc tính. Sự chênh lệch entropy trước và sau khi phân chia một tập dữ liệu bằng một thuộc tính nào đó được gọi là độ tăng thông tin và được định nghĩa dưới đây.

Độ tăng thông tin (Information Gain), ký hiệu *IG*, là chỉ số đánh giá độ tốt của thuộc tính trong việc phân chia tập dữ liệu thành những tập con đồng nhất. IG được tính dựa trên entropy theo công thức sau:

$$IG(S,A) = H(S) - \sum_{v \in values(A)} \frac{|S_V|}{|S|} H(S_v)$$

trong đó:

S là tập dữ liệu ở nút hiện tại

A là thuộc tính

$values(A)$ là tập các giá trị của thuộc tính A.

S_v là tập các mẫu có giá trị thuộc tính A bằng v.

$|S|$ và $|S_v|$ là lực lượng của các tập hợp tương ứng.

Về bản chất, IG là độ chênh lệch giữa entropy của tập S và tổng entropy của các tập con S_v được tạo ra do phân chia S bằng cách sử dụng thuộc tính A. Do các tập con có thể có kích thước không bằng nhau nên entropy của tập con được nhân với một trọng số $|S_v| / |S|$, tức là tập con có kích thước lớn hơn sẽ đóng góp nhiều hơn vào tổng entropy.

Giá trị của IG được sử dụng làm tiêu chí để lựa chọn thuộc tính tốt nhất tại mỗi nút. Theo thuật toán ID3, thuộc tính được lựa chọn là *thuộc tính có giá trị IG lớn nhất*.

Ví dụ minh họa

Để minh hoạ cho cách tính độ tăng thông tin IG và lựa chọn thuộc tính tốt nhất, ta sẽ sử dụng dữ liệu huấn luyện cho trong bảng trên hình 5.5. Cụ thể, cần xác định thuộc tính tốt nhất tại nút gốc cho dữ liệu trong bảng 5.5 bằng cách tính IG cho các thuộc tính.

Với thuộc tính Gió:

$values$(Gió) = {yếu, mạnh}

S = [9+, 5-], $H(S) = 0.94$

$S_{yếu}$ = [6+, 2-] , $H(S_{yếu}) = 0.811$

$S_{mạnh}$ = [3+, 3-], $H(S_{mạnh}) = 1$

Do vậy:

$IG(S, Gió) = H(S) - (8/14)\, H(S_{yếu}) - (6/14)\, H(S_{mạnh})$

$H = 0.94 - (8/14) * 0.811 - (6/14) * 1$

$= 0.048$

Tính tương tự với ba thuộc tính còn lại, ta được:

IG (S, Trời) = 0.246

IG (S, Độ ẩm) = 0.151

IG (S, Gió) = 0.048

IG (S, Nhiệt độ) = 0.029

Trong bốn thuộc tính có thể sử dụng ở nút gốc, IG (S, Trời) có giá trị lớn nhất. Như vậy, thuộc tính tốt nhất là Trời được sử dụng cho nút gốc. Sau khi chọn nút gốc là Trời, ta được các bộ dữ liệu con ở ba nhánh tương ứng ba giá trị của thuộc tính "Trời" như trên hình 5.8. Để tiện cho việc trình bầy, số thứ tự các ví dụ được cho ngay tại các nút.

Đối với nhánh giữa, toàn bộ mẫu có nhãn dương, do vậy quá trình học cây cho nhánh này dừng lại, thuật toán tạo nút lá với nhãn "có". Đối với nhánh bên trái và bên phải, quá trình học cây được tiếp tục với tập dữ liệu con tại nhánh đó. Dưới đây là minh họa việc tính IG và chọn thuộc tính cho nút tiếp theo bên trái.

Tính IG cho Độ ẩm:

$S_{nắng}$ = [2+, 3–], $H(S_{nắng}) = 0.97$

$S_{nắng,\ cao}$ = [0+, 3–], $H(S_{nắng,\ cao}) = 0$

$S_{nắng,\ bình\ thường}$ = [2+, 0–], $H(S_{nắng,\ bình\ thường}) = 0$

Do vậy

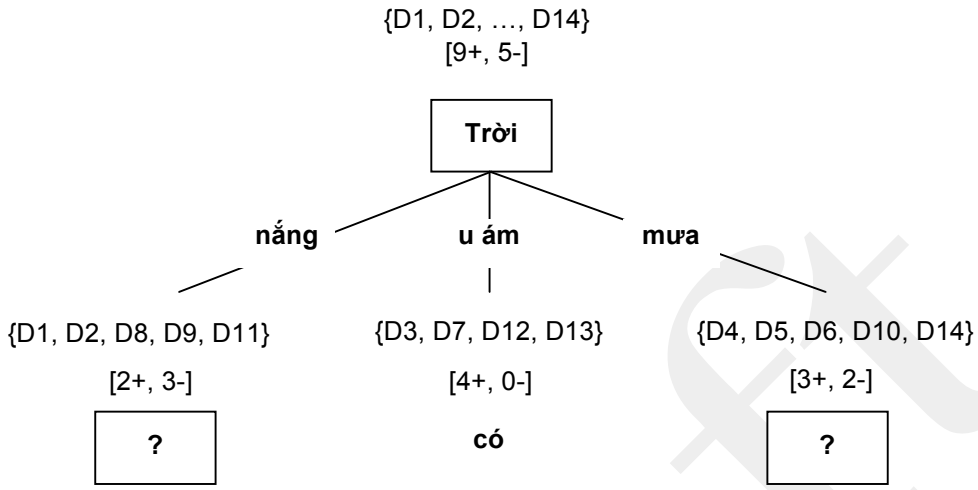
$IG(S_{nắng}, Độ\ ẩm) = 0.97 - (3/5)*0 - (2/5)*0 = 0.97$

Tính tương tự với hai thuộc tính còn lại, ta có:

$IG(S_{nắng}, \text{Nhiệt độ}) = 0.97 - (2/5)*0 - (2/5)*1 - (1/5)*0 = 0.57$

$IG(S_{nắng}, \text{Gió}) = 0.97 - (2/5)*1 - (3/5)*0.918 = 0.019$

Thuộc tính Độ ẩm có IG lớn nhất và được chọn cho nút này. Kết quả học cây đầy đủ được thể hiện trên hình 5.4.



Hình 5.8. Ví dụ xây dựng cây quyết định

5.2.3. Các đặc điểm thuật toán học cây quyết định

Thuật toán học cây quyết định ID3 ở trên có một số đặc điểm sau:

d. ID3 là thuật toán tìm kiếm cây quyết định phù hợp với dữ liệu huấn luyện.

e. Đây là phương pháp tìm kiếm theo kiểu tham lam, từ trên xuống, bắt đầu từ cây rỗng. Hàm đánh giá là độ tăng thông tin. Tính chất tham lam của thuật toán thể hiện ở chỗ tại mỗi nút, thuộc tính được chọn là thuộc tính có hàm mục tiêu lớn nhất, thuật toán không nhìn xa hơn nút hiện tại khi quyết định chọn thuộc tính. Không gian tìm kiếm là đầy đủ, nghĩa là theo cách xây dựng cây như vậy, thuật toán có thể di chuyển tới mọi cây hợp lệ.

f. ID3 có khuynh hướng lựa chọn cây quyết định đơn giản tức là cây có ít nút, trong đó những nút tương ứng với thuộc tính có độ tăng thông tin lớn được xếp ở gần gốc hơn.

Lưu ý: Ở trên vừa nhắc tới "khuynh hướng" (bias) của thuật toán. Trong học máy, từ khuynh hướng dùng để chỉ tính chất thuật toán ưu tiên một phương án này hơn một phương án khác trong khi cả hai phương án đều thỏa mãn yêu cầu đặt ra. Trong trường hợp cây quyết định, nếu hai cây cùng phù hợp với dữ liệu thì thuật toán có khuynh hướng lựa chọn cây ít nút hơn.

Việc lựa chọn cây quyết định đơn giản phù hợp với một nguyên tắc trong triết học được gọi là *Occam's razor* (Occam là tên một nhà triết học) theo đó nếu có nhiều giả thiết cho phép giải thích một số quan sát nào đó thì ưu tiên chọn giả thiết đơn giản hơn.

5.2.4. Vấn đề quá vừa dữ liệu

Quá vừa dữ liệu (data overfitting hay đơn giản là overfitting) là một vấn đề thường gặp trong học máy và có ảnh hưởng nhiều tới độ chính xác của các kỹ thuật học máy.

Trong khi xây dựng cây quyết định (hay bộ phân loại nói chung), thuật toán học máy thường cố gắng để cây phù hợp với dữ liệu, tức là phân loại đúng các mẫu huấn luyện, ở mức tối đa. Tuy nhiên, mục đích học cây quyết định không phải để phân loại dữ liệu mẫu, mà để phân loại dữ liệu nói chung, tức là dữ liệu mà thuật toán chưa biết trong thời gian học. Có thể xẩy ra tình huống cây quyết định có độ chính xác tốt trên dữ liệu huấn luyện nhưng lại cho độ chính xác không tốt trên dữ liệu nói chung. Khi đó ta nói cây quyết định quá vừa với dữ liệu huấn luyện.

Ta nói rằng cây quyết định t *quá vừa dữ liệu huấn luyện nếu tồn tại cây quyết định* t' *sao cho* t *phân loại chính xác hơn* t' *trên dữ liệu huấn luyện nhưng kém chính xác hơn* t' *trên dữ liệu nói chung*.

Lý do bộ phân loại làm việc tốt trên dữ liệu huấn luyện nhưng không tốt trên dữ liệu nói chung là do các mẫu huấn luyện thường không đủ và không mang tính đại diện cho phân bố của dữ liệu nói chung. Chẳng hạn, khi số lượng mẫu tại nút lá ít, việc xuất hiện tương quan giữa giá trị thuộc tính và nhãn phân loại có thể xẩy ra do trùng hợp ngẫu nhiên dẫn tới một số thuộc tính phân chia các mẫu rất tốt trong khi thực tế thuộc tính không có quan hệ với nhãn phân loại.

Một lý do khác dẫn tới quá vừa dữ liệu là do dữ liệu huấn luyện có nhiễu trong khi thuật toán cố gắng xây dựng cây để phân loại đúng các ví dụ nhiễu này.

Đối với cây quyết định, thuật toán ID3 mô tả ở trên phát triển nhánh cây rất sâu cho đến khi phân loại đúng toàn bộ mẫu, hoặc khi đã hết thuộc tính. Nghiên cứu cho thấy, việc phát triển cây phức tạp với nhiều nút là nguyên nhân chính dẫn tới quá vừa dữ liệu. Từ đây, có hai nhóm giải pháp chính để hạn chế quá vừa dữ liệu cho thuật toán học cây quyết định:

g. Giải pháp dừng việc dựng cây quyết định sớm, trước khi cây đủ phức tạp để phân loại đúng mẫu huấn luyện.
h. Giải pháp xây dựng cây đầy đủ, sau đó thực hiện "tỉa" cây để có cây đơn giản hơn.

Trong hai nhóm trên, nhóm giải pháp thứ hai được sử dụng thành công hơn trên thực tế và do vậy sẽ được trình bày tiếp theo.

Chống quá vừa dữ liệu bằng cách tỉa cây.

Trước tiên, để thực hiện tỉa cây, cần có cách xác định độ chính xác phân loại của cây. Do mục đích của cây là phân loại những mẫu chưa biết trong quá trình huấn luyện nên cách tính độ chính xác thông dụng nhất là sử dụng tập huấn luyện và tập kiểm tra riêng như sau:

i. Toàn bộ mẫu được chia thành hai tập: tập thứ nhất gọi là tập huấn luyện, tập thứ hai gọi là tập kiểm tra, thường với tỷ lệ 2:1.
j. Sử dụng tập huấn luyện để xây dựng cây, sử dụng tập kiểm tra để tính độ chính xác của cây, tức là xác định xem kết quả phân loại của cây phù hợp đến mức nào với mẫu trong tập kiểm tra.

Trong trường hợp ít dữ liệu, một phương pháp hay được sử dụng là *kiểm tra chéo*. Dữ liệu được chia ngẫu nhiên thành *n* phần bằng nhau. Thuật toán lần lượt sử dụng *n*-1 phần làm tập huấn luyện và phần còn lại làm tập kiểm tra. Độ chính xác được tính bằng độ chính xác trung bình cho *n* lần. Chi tiết về kỹ thuật kiểm tra chéo được trình bầy trong phần 5.6.2.

Thủ tục tỉa cây thực hiện như sau. Trước tiên sử dụng tập huấn luyện để xây dựng cây đầy đủ. Sau đó xem xét để tỉa dần các nút. Khi tỉa nút, toàn bộ các nhánh bên dưới nút bị bỏ, nút trở thành nút lá với nhãn phân loại lấy theo đa số nhãn của các ví dụ tại nút đó. Nút sẽ được tỉa nếu độ chính xác sau khi tỉa không giảm so với trước khi tỉa. Lưu ý rằng độ chính xác được tính trên tập kiểm tra.

Quá trình tỉa được lặp lại, tại mỗi bước chọn nút để tỉa là nút cho phép tăng độ chính xác phân loại nhiều nhất. Thủ tục tỉa nút dừng lại khi việc bỏ đi bất cứ nút nào cũng làm giảm độ chính xác.

Chống quá vừa bằng cách tỉa luật

Thay vì tỉa các nút như thuật toán ID3 ở trên, một cải tiến của thuật toán ID3 là thuật toán C4.5 thực hiện tỉa các luật như sau:

1. Xây dựng cây quyết định cho phép phân loại đúng tối đa tập huấn luyện.
2. Biến đổi cây thành luật suy diễn sao cho mỗi nhánh từ gốc đến lá tương ứng một luật. Ví dụ, luật có dạng "Nếu Trời = nắng và Độ ẩm = cao Thì không chơi"
3. Tỉa từng luật bằng cách bỏ bớt các điều kiện thành phần nếu sau khi bỏ độ chính xác tăng lên.
4. Sắp xếp các luật đã được tỉa theo độ chính xác trên tập kiểm tra. Sử dụng luật theo thứ tự đó để phân loại ví dụ mới.

Ví dụ: luật "Nếu Trời = nắng và Độ ẩm = cao Thì không chơi" có thể tạo ra hai luật bằng cách bỏ các điều kiện "Trời = nắng" và "Độ ẩm = cao".

Cần lưu ý rằng, sau khi tỉa các luật, cùng một ví dụ có thể trùng với vế trái của nhiều hơn một luật (điều này không xảy ra với cây do các nhánh cây loại trừ lẫn nhau). Trong trường hợp này cần xác định luật nào sẽ được sử dụng. Đó là lý do cần sắp xếp các luật và sử dụng theo thứ tự đó.

5.2.5. Sử dụng thuộc tính có giá trị liên tục

Thuật toán trình bày ở trên yêu cầu thuộc tính nhận giá trị rời rạc trong một tập hữu hạn. Trong nhiều trường hợp, thuộc tính có thể nhận giá trị liên tục dưới dạng số thực. Chẳng hạn, nhiệt độ được cho dưới dạng số đo thực như trong ví dụ sau (ở đây nhiệt độ tính bằng độ F):

Nhiệt độ	45	56	60	74	80	90
Chơi tennis	không	không	có	có	có	không

Để sử dụng thuộc tính như vậy cần tạo ra những thuộc tính rời rạc mới cho phép phân chia thuộc tính rời rạc thành các khoảng giá trị. Với mỗi thuộc tính liên tục *A*, cách thường

được sử dụng là tạo ra thuộc tính rời rạc *Ac* sao cho *Ac* = true nếu *A* > *c* và *Ac* = false nếu *A* ≤ *c*, trong đó c là giá trị ngưỡng.

Vấn đề đặt ra là xác định ngưỡng *c* như thế nào. Trước hết, *c* cần được chọn sao cho *Ac* đem lại độ tăng thông tin lớn nhất. Để tìm được *c* như vậy, ta sắp xếp các ví dụ theo thứ tự tăng dần của thuộc tính *A*, sau đó xác định những trường hợp hai ví dụ nằm cạnh nhau có nhãn khác nhau. Giá trị trung bình của thuộc tính *A* của hai thuộc tính như vậy sẽ được sử dụng làm giá trị dự kiến của *c*, trong ví dụ trên là (56+60)/2 = 58 và (80+90)/2 = 85 . Sau đó tính độ tăng thông tin cho từng giá trị dự kiến và chọn *c* đem lại độ thăng thông tin lớn nhất, trong ví dụ trên là độ thăng thông tin của Nhiệt_độ_{58} và Nhiệt_độ_{85}.

Phương pháp trên có thể mở rộng bằng cách cha giá trị thuộc tính thành nhiều khoảng với nhiều ngưỡng, thay vì chỉ sử dụng một ngường như ta vừa thấy.

5.2.6. Sử dụng cách đánh giá thuộc tính khác

Cách đánh giá thuộc tính bằng độ tăng thông tin IG ở trên không phải duy nhất và có thể cho kết quả không tốt trong một số trường hợp. Cụ thể, việc đánh giá dựa trên entropy thuần túy dẫn tới việc ưu tiên những thuộc tính có nhiều giá trị và do vậy tạo ra nhiều tập con.

Trong ví dụ ở hình 5.5, nếu sử dụng cả Ngày như một thuộc tính, thuộc tính này sẽ có tới 14 giá trị khác nhau, chia tập huấn luyện thành 14 tập con với entropy = 0 và do vậy có Ngày có IG cao nhất. Việc chọn ngày như vậy dẫn tới cây quyết định không có khả năng phân loại những ngày tiếp theo. Như vậy, thuộc tính Ngày mặc dầu có IG rất tốt nhưng cần tránh ví có nhiều giá trị.

Để giải quyết vấn đề này, ta có thể thêm thành phần vào công thức tính IG để phạt những thuộc tính nhiều giá trị. Thành phần đó gọi là thông tin chia (Split Informatio – SI) và được tính như sau:

$$SI(S, A) = -\sum_{i=1}^{C} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Tiêu chuẩn đánh giá thuộc tính (ký hiệu là GR – Gain Ratio) khi đó được tạo thành bằng cách chia IG cho SI:

$$GR = IG(S, A) / SI(S, A)$$

Một vấn đề cần quan tâm khi sử dụng GR là giá trị SI có thể bằng không khi $|S_i| = |S|$. Để xử lý trường hợp này cần có những quy tắc riêng, chẳng hạn chỉ tính GR khi IG lớn tới một mức nào đó.

Bên cạnh GR còn có nhiều độ đo khác được nghiên cứu và đề xuất sử dụng khi xây dựng cây quyết định.

5.3. PHÂN LOẠI BAYES ĐƠN GIẢN

Phần này sẽ đề cập tới phân loại Bayes đơn giản (Naïve Bayes), một phương pháp phân loại đơn giản nhưng có nhiều ứng dụng trong thực tế như phân loại văn bản, lọc thư rác. Phân loại Bayes đơn giản là trường hợp riêng của kỹ thuật học máy Bayes, trong đó các giả thiết về độc lập xác suất được sử dụng để đơn giản hóa việc tính xác suất.

5.3.1. Phương pháp phân loại Bayes đơn giản

Tương tự như học cây quyết định ở trên, phân loại Bayes đơn giản sử dụng trong trường hợp mỗi ví dụ được cho bằng tập các thuộc tính $<x_1, x_2, \ldots, x_n>$ và cần xác định nhãn phân loại y, y có thể nhận giá trị từ một tập nhãn hữu hạn C.

Trong giai đoạn huấn luyện, dữ liệu huấn luyện được cung cấp dưới dạng các mẫu $<\mathbf{x}_i, y_i>$. Sau khi huấn luyện xong, bộ phân loại cần dự đoán nhãn cho mẫu mới **x**.

Theo lý thuyết học Bayes, nhãn phân loại được xác định bằng cách tính xác suất điều kiện của nhãn khi quan sát thấy tổ hợp giá trị thuộc tính $<x_1, x_2, \ldots, x_n>$. Thuộc tính được chọn, ký hiệu c_{MAP} là thuộc tính có xác suất điều kiện cao nhất (MAP là viết tắt của maximum a posterior), tức là:

$$y = c_{MAP} = \arg\max_{c_j \in C} P(c_j \mid x_1, x_2, ..., x_n)$$

Sử dụng quy tắc Bayes, biểu thức trên được viết lại như sau

$$c_{MAP} = \arg\max_{c_j \in C} \frac{P(x_1, x_2, ..., x_n \mid c_j) P(c_j)}{P(x_1, x_2, ..., x_n)}$$

Trong vế phải của biểu thức này, mẫu số không phụ thuộc vào c_j và vì vậy không ảnh hưởng tới giá trị của C_{MAP}. Do đó, ta có thể bỏ mẫu số và viết lại như sau:

$$C_{MAP} = \arg\max_{c_j \in C} P(x_1, x_2, ..., x_n \mid c_j) P(c_j)$$

Hai thành phần trong biểu thức trên được tính từ dữ liệu huấn luyện. Giá trị $P(c_j)$ được tính bằng tần suất quan sát thấy nhãn c_j trên tập huấn luyện, tức là bằng số mẫu có nhãn là c_j chia cho tổng số mẫu. Việc tính $P(x_1, x_2, ..., x_n \mid c_j)$ khó khăn hơn nhiều. Vấn đề là số tổ hợp giá trị của n thuộc tính cùng với nhãn phân loại là rất lớn khi n lớn. Để tính xác suất này được chính xác, mỗi tổ hợp giá trị thuộc tính phải xuất hiện cùng nhãn phân loại đủ nhiều, trong khi số mẫu huấn luyện thường không đủ lớn.

Để giải quyết vấn đề trên, ta giả sử các thuộc tính là độc lập về xác suất với nhau khi biết nhãn phân loại c_j. Trên thực tế, các thuộc tính thường không độc lập với nhau như vậy, chẳng hạn đối với ví dụ chơi tennis, khi trời nắng thì xác suất nhiệt độ cao cũng lớn hơn. Chính vì dựa trên giả thiết độc lập xác suất đơn giản như vậy nên phương pháp có tên gọi "Bayes đơn giản". Tuy nhiên, như ta thấy sau đây, giả thiết như vậy cho phép tính xác suất điều kiện đơn giản hơn nhiều và trên thực tế phân loại Bayes có độ chính xác tốt trong rất nhiều ứng dụng.

Với giả thiết về tính độc lập xác suất có điều kiện, có thể viết:

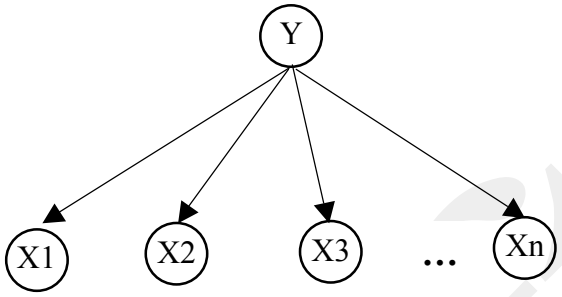
$$P(x_1, x_2, ..., x_n \mid c_j) = P(x_1 \mid c_j) P(x_2 \mid c_j) ... P(x_n \mid c_j)$$

tức là xác suất đồng thời quan sát thấy các thuộc tính bằng tích xác suất điều kiện của từng thuộc tính riêng lẻ. Thay vào biểu thức ở trên, ta được **bộ phân loại Bayes đơn giản** (có đầu ra ký hiệu là c_{NB}) như sau.

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_i P(x_i \mid c_j)$$

trong đó, $P(x_i \mid c_j)$ được tính từ dữ liệu huấn luyện bằng số lần x_i xuất hiện cùng với c_j chia cho số lần c_j xuất hiện. Việc tính xác suất này đòi hỏi ít dữ liệu hơn nhiều so với tính $P(x_1, x_2, ..., x_n \mid c_j)$.

Trên hình 5.9 là biểu diễn mô hình phân loại Bayes đơn giản dưới dạng mạng Bayes. Các thuộc tính không được nối với nhau bởi các cạnh và do vậy các thuộc tính độc lập xác suất với nhau nếu biết giá trị của nhãn phân loại.



Hình 5.9: Mô hình Bayes đơn giản: các thuộc tính Xi độc lập xác suất với nhau nếu biết giá trị nhãn phân loại Y.

Huấn luyện.

Quá trình huấn luyện hay học Bayes đơn giản là quá trình tính các xác suất $P(c_j)$ và các xác suất điều kiện $P(x_i \mid c_j)$ bằng cách đếm trên tập dữ liệu huấn luyện. Như vậy, khác với học cây quyết định, Học Bayes đơn giản không đòi hỏi tìm kiếm trong không gian các bộ phân loại. Các xác suất $P(c_j)$ và các xác suất điều kiện $P(x_i \mid c_j)$ được tính trên tập dữ liệu huấn luyện theo công thức sau:

$$P(c_j) = \frac{\text{Số mẫu có nhãn là } c_j}{\text{Tổng số mẫu trong tập huấn luyện}}$$

$$P(x_i \mid c_j) = \frac{\text{Số mẫu có giá trị thuộc tính } X_i = x_i \text{ và nhãn là } c_j}{\text{Số mẫu có nhãn là } c_j}$$

Ví dụ.

Để minh họa cho kỹ thuật học Bayes đơn giản, ta sử dụng lại bài toán phân chia ngày thành phù hợp hay không phù hợp cho việc chơi tennis theo điều kiện thời tiết đã được sử dụng trong phần học cây quyết định với dữ liệu huấn luyện cho trong bảng 4.1. Giả sử phải xác định nhãn phân loại cho ví dụ sau:

< Trời = nắng, Nhiệt độ = trung bình, Độ ẩm = cao, Gió = mạnh >

Thay số liệu của bài toán vào công thức Bayes đơn giản, ta có:

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_i P(x_i \mid c_j)$$

$$= \arg\max_{c_j \in \{co, khong\}} P(\text{Trời=nắng} \mid c_j) P(\text{Nh. độ=t. bình} \mid c_j) P(\text{Độ ẩm=cao} \mid c_j)$$

P (Gió=mạnh $|c_j)P$ (c_j)

Do c_j có thể nhận hai giá trị, ta cần tính 10 xác suất. Các xác suất P(có) và P(không) được tính bằng tất suất "có" và "không" trên dữ liệu huấn luyện.

P(có) = 9/14 = 0,64

P(không) = 5/14 = 0,36

Các xác suất điều kiện cũng được tính từ dữ liệu huấn luyện, ví dụ ta có:

P (Độ ẩm = cao | có) = 3/9 = 0,33

P (Độ ẩm = cao | không) = 4/5 = 0,8

Thay các xác suất thành phần vào công thức Bayes đơn giản, ta được:

P(có)P(nắng| có)P(trung bình| có)P(cao| có)P(mạnh| có) = 0.0053

P(không)P(nắng| không)P(trung bình| không)P(cao| không)P(mạnh| không) = 0.0206

Như vậy, theo phân loại Bayes đơn giản, ví dụ đang xét sẽ được phân loại là "không". Cần chú ý rằng, 0.0053 và 0.0206 không phải là xác suất thực của nhãn "có" và "không". Để tính xác suất thực, ta cần chuẩn hóa để tổng hai xác suất bằng 1. Việc chuẩn hoá được thực hiện bằng cách chia mỗi số cho tổng của hai số. Chẳng hạn xác suất có chơi sẽ bằng 0.0053/(0.0053+0.0206) = 0.205.

5.3.2. Vấn đề tính xác suất trên thực tế

Phân loại Bayes đơn giản đòi hỏi tính các xác suất điều kiện thành phần $P(x_i \mid c_j)$. Xác suất này được tính bằng n_c / n, trong đó n_c số lần x_i và c_j xuất hiện đồng thời trong tập huấn luyện và n là số lần c_j xuất hiện.

Trong nhiều trường hợp, giá trị n_c có thể rất nhỏ, thậm chí bằng không, và do vậy ảnh hưởng tới độ chính xác khi tính xác suất điều kiện. Nếu $n_c = 0$, xác suất điều kiện cuối cùng sẽ bằng không, bất kể các xác suất thành phần khác có giá trị thế nào.

Để khắc phục vấn đề này, một kỹ thuật được gọi là *làm trơn* thường được sử dụng. Kỹ thuật làm trơn đơn giản nhất sử dụng công thức tính $P(x_i \mid c_j)$ như sau:

$$P(x_i \mid c_j) = (n_c + 1) / (n + 1)$$

Như vậy, kể cả khi $n_c = 0$, xác suất vẫn nhận giá trị khác 0.

Trong trường hợp chung, có thể sử dụng công thức được làm trơn sau:

$$P(x_i \mid c_j) = \frac{n_c + mp}{n + m}$$

trong đó p là xác suất tiền nghiệm của x_i và m là tham số cho phép xác định ảnh hưởng của p tới công thức. Nếu không có thêm thông tin gì khác thì xác suất tiền nghiệm thường được tính $p = 1 / k$, trong đó k là số thuộc tính của thuộc tính X_i. Ví dụ, nếu không có thêm thông tin gì thêm thì xác suất quan sát thấy Gió = mạnh sẽ là 1/2 do thuộc tính Gió có hai giá trị. Nếu $m = 0$, ta được công thức không làm trơn ban đầu. Ngược lại, khi m→ ∞, xác suất hậu nghiệm sẽ bằng p, bất kể n_c thế nào. Trong những trường hợp còn lại, cả n_c / n và p cùng đóng góp vào công thức.

5.3.3. Ứng dụng trong phân loại văn bản tự động

Phân loại văn bản tự động là bài toán có nhiều ứng dụng thực tế. Trước tiên, cho một tập huấn luyện bao gồm các văn bản. Mỗi văn bản có thể thuộc vào một trong C loại khác nhau (ở đây ta không xét trường hợp mỗi văn bản có thể thuộc vào nhiều loại khác nhau). Sau khi huấn luyện xong, thuật toán phân loại nhận được văn bản mới và cần xác định phân loại cho văn bản này. Ví dụ, với các văn bản là nội dung thư điện tử, thuật toán có thể phân loại thư thành "thư rác" và "thư bình thường". Khi huấn luyện, thuật toán học được cung cấp một tập thư rác và một tập thư thường. Sau đó, dựa trên nội dung thư mới nhận, bộ phân loại sẽ tự xác định đó có phải thư rác không. Một ứng dụng khác là tự động phân chia bản tin thành các thể loại khác nhau, ví dụ "chính trị", "xã hội", "thể thao".v.v. như trên báo điện tử.

Phân loại văn bản tự động là dạng ứng dụng trong đó phân loại Bayes đơn giản và các phương pháp xác suất khác được sử dụng rất thành công. Chương trình lọc thư rác mã nguồn mở SpamAssassin (http:// spamassassin.apache.org) là một chương trình lọc thư rác được sử dụng rộng rãi với nhiều cơ chế lọc khác nhau, trong đó lọc Bayes đơn giản là cơ chế lọc chính được gán trọng số cao nhất.

Sau đây ta sẽ xem xét cách sử dụng phân loại Bayes đơn giản cho bài toán phân loại văn bản. Để đơn giản, ta sẽ xét trường hợp văn bản có thể nhận một trong hai nhãn: "rác" và "không".

Để sử dụng phân loại Bayes đơn giản, cần giải quyết hai vấn đề chủ yếu: thứ nhất, biểu diễn văn bản thế nào cho phù hợp; thứ hai: lựa chọn công thức cụ thể cho bộ phân loại Bayes.

Cách thông dụng và đơn giản nhất để biểu diễn văn bản là cách biểu diễn bằng "túi từ" (bag-of-word). Theo cách này, mỗi văn bản được biểu diễn bằng một tập hợp, trong đó mỗi phần tử của tập hợp tương ứng với một từ khác nhau của văn bản. Để đơn giản, ở đây ta coi mỗi từ là một đơn vị ngôn ngữ được ngăn với nhau bởi dấu cách. Lưu ý rằng đây là cách đơn giản nhất, ta cũng có thể thêm số lần xuất hiện thực tế của từ trong văn bản. Cách biểu diễn này không quan tâm tới vị trí xuất hiện của từ trong văn bản cũng như quan hệ với các từ xung quanh, do vậy có tên gọi là túi từ. Ví dụ, một văn bản có nội dung "Chia thư thành thư rác và thư thường" sẽ được biểu diễn bởi tập từ {"chia", "thư", "thành", "rác", "và", "thường"} với sáu phần tử.

Giả thiết các từ biểu diễn cho thư xuất hiện độc lập với nhau khi biết nhãn phân loại, công thức Bayes đơn giản cho phép ta viết:

$$c_{NB} = \arg\max_{c_j \in \{rac, khong\}} P(c_j) \prod_i P(x_i \mid c_j)$$

$$= \arg\max_{c_j \in \{rac, khong\}} P(c_j)\, P(\text{"chia"} \mid c_j)\, P(\text{"thư "} \mid c_j)\, P(\text{"thành"} \mid c_j)$$

$$P(\text{"rác"} \mid c_j)\, P(\text{"và"} \mid c_j)\, P(\text{"thường "} \mid c_j)$$

Các xác suất $P(\text{"rác"} \mid c_j)$ được tính từ tập huấn luyện như mô tả ở trên. Những từ chưa xuất hiện trong tập huấn luyện sẽ bị bỏ qua, không tham gia vào công thức.

Cần lưu ý rằng cách biểu diễn và áp dụng phân loại Bayes đơn giản cho phân loại văn bản vừa trình bày là những phương án đơn giản. Trên thực tế có rất nhiều biến thể khác nhau

cả trong việc chọn từ, biểu diễn văn bản bằng các từ, cũng như công thức tính xác suất điều kiện của văn bản.

Mặc dù đơn giản, nhiều thử nghiệm cho thấy, phân loại văn bản tự động bằng Bayes đơn giản có độ chính xác khá cao. Trên nhiều tập dữ liệu thư điện tử, tỷ lệ phân loại chính xác thư rác có thể đạt trên 98%. Kết quả này cho thấy, mặc dù giả thiết các từ độc lập với nhau là không thực tế, độ chính xác phân loại của Bayes đơn giản không bị ảnh hưởng đáng kể.

5.4. HỌC DỰA TRÊN VÍ DỤ: THUẬT TOÁN K LÁNG GIỀNG GẦN NHẤT

5.4.1. Nguyên tắc chung

Trong các phương pháp học cây quyết định và Bayes đơn giản, thuật toán học dựa trên dữ liệu huấn luyện để học ra mô hình và tham số cho bộ phân loại. Mô hình phân loại sau đó được sử dụng để dự đoán nhãn cho ví dụ mới. Quá trình học thực chất là quá trình xác định dạng và tham số của hàm đích, là hàm xấp xỉ giá trị nhãn phân loại.

Phần này sẽ trình bày kỹ thuật học máy dựa trên một nguyên tắc khác gọi là *học dựa trên ví dụ* (instance-based learning). Khác với các phương pháp học ở trên, học dựa trên ví dụ không tạo ra mô hình hay hàm đích cho dữ liệu, thay vào đó, trong quá trình học thuật toán chỉ lưu lại tất cả các mẫu huấn luyện được cung cấp. Khi cần phân loại hay ra quyết định cho ví dụ mới, thuật toán tìm những mẫu huấn luyện tương tự và xác định nhãn phân loại hay giá trị của ví dụ dựa trên những mẫu này.

Do thuật toán không làm gì trong quá trình học mà chỉ lưu lại các mẫu huấn luyện, phương pháp học dựa trên ví dụ còn được gọi là học lười (lazy learning) hay học bằng cách nhớ (memory-based learning). Học dựa trên ví dụ bao gồm một số kỹ thuật học khác nhau như thuật toán k-láng giềng gần nhất (k-nearest neighbor viết tắt là k-NN), suy diễn theo trường hợp (case-based reasoning). Điểm khác nhau cơ bản giữa nhứng kỹ thuật này là cách biểu diễn và tính độ tương tự giữa các ví dụ. Thuật toán k-hàng xóm gần nhất sử dụng cách biểu diễn ví dụ đơn giản dưới dạng vec tơ trong không gian Ơclit và sử dụng khoảng cách Ơclit để tính độ tương tự, trong khi suy diễn theo trường hợp dựa trên việc biểu diễn các mẫu (gọi là trường hợp) phức tạp hơn và dùng những kỹ thuật phức tạp được xây dựng riêng để tính độ tương tự cho các trường hợp.

Ưu điểm. So với phương pháp học dựa trên mô hình, học dựa trên ví dụ có một số ưu điểm. Thứ nhất, do không quy định trước mô hình hay dạng của hàm đích, học dựa trên ví dụ có thể xây dựng những hàm đích rất phức tạp. Thứ hai, thay vì xây dựng hàm đích chung cho toàn bộ dữ liệu, học dựa trên ví dụ xây dựng hàm đích dựa trên một số mẫu gần với ví dụ đang cần dự đoán, do vậy có thể tận dụng được đặc điểm mang tính cục bộ của dữ liệu để mô tả tốt hơn giá trị ví dụ mới.

Nhược điểm. Nhược điểm thứ nhất của học dựa trên ví dụ là tốc độ chậm trong giai đoạn phân loại. Do thuật toán phải so sánh ví dụ mới với toàn bộ tập mẫu để tìm ra những mẫu tương tự nên thời gian phân loại tỷ lệ thuận với kích thước tập mẫu. Để khắc phục vấn đề tốc độ, cách thông dụng nhất là sử dụng kỹ thuật đánh chỉ số để tìm kiếm nhanh mẫu tương tự. Nhược điểm thứ hai của học dựa trên ví dụ là việc tính độ tương tự được thực hiện với toàn bộ thuộc tính. Nếu thuộc tính không liên quan tới phân loại của ví dụ thì khi sử dụng sẽ

gây nhiễu, khiến những ví dụ cùng nhãn không tương tự với nhau. Vấn đề chọn và sử dụng những thuộc tính tốt, do vậy, có ảnh hưởng quyết định tới độ chính xác của phương pháp này.

5.4.2. Phương pháp k-láng giềng gần nhất

K-láng giềng gần nhất (k-nearest neighbors, viết tắt là k-NN) là phương pháp tiêu biểu nhất của học dựa trên ví dụ. Nguyên tắc của phương pháp này là đặc điểm của mẫu được quyết định dựa trên đặc điểm của *k* mẫu giống mẫu đang xét nhất. Ví dụ, muốn xác định nhãn phân loại, ta tìm *k* mẫu gần nhất và xem những mẫu này mang nhãn gì.

Phương pháp k-NN thường làm việc với dữ liệu trong đó các thuộc tính được cho dưới dạng vec tơ các số thực. Như vậy, mỗi mẫu tương ứng với một điểm trong không gian Ơ clit. Giả sử mẫu *x* có giá trị thuộc tính là < $a_1(x)$, $a_2(x)$,…, $a_n(x)$ >. Để xác định các mẫu giống x, cần có độ đo khoảng cách giữa các mẫu. Do mẫu tương ứng với điểm trong không gian, khoảng cách Ơ clit thường được dùng cho mục đích này. Khoảng cách Ơ clit giữa hai mẫu x_i và x_j được tính như sau:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{n} (a_l(x_i) - a_l(x_j))^2}$$

Với khoảng cách $d(x_i, x_j)$ vừa được định nghĩa, phương pháp k-NN cho hai trường hợp: phân loại và hồi quy (regression) được thực hiện như sau.

Phân loại

Mỗi mẫu *x* có thể nhãn phân loại f(x) với f(x) nhận một giá trị trong tập hữu hạn các phân loại *C*. Thuật toán k-NN cho phân loại được cho trên hình 5.10.

Giai đoạn học (huấn luyện)

Lưu các mẫu huấn luyện có dạng <*x*, *f*(*x*)> vào cơ sở dữ liệu

Giai đoạn phân loại

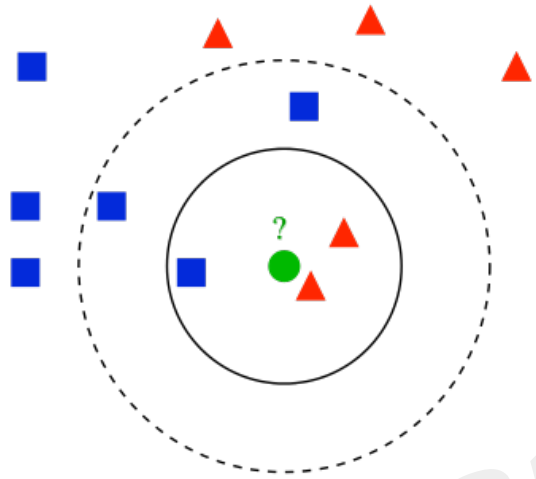
Đầu vào: tham số k

Với mẫu x cần phân loại:

1. Tính khoảng cách $d(x, x_i)$ từ *x* tới tất cả mẫu x_i trong cơ sở dữ liệu
2. Tìm *k* mẫu có $d(x, x_i)$ nhỏ nhất, giả sử *k* mẫu đó là $x_1, x_2, \ldots, x_k$.
3. Xác định nhãn phân loại *f'(x)* là nhãn chiếm đa số trong tập { $x_1, x_2, \ldots, x_k$.}

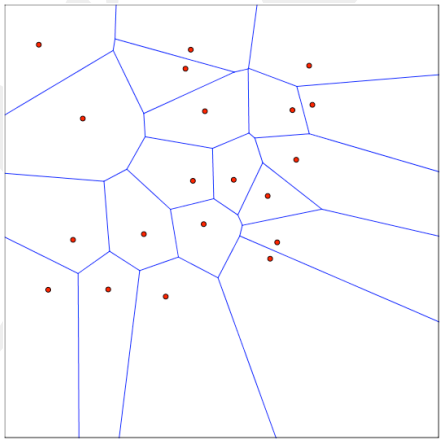
Hình 5.10. Thuật toán k-NN cho bài toán phân loại

Kết quả phân loại của thuật toán k-NN được minh họa trên hình 5.11 cho trường hợp phân loại hai lớp. Các hình vuông biểu diễn các ví dụ huấn luyện thuộc một lớp, hình tam giác biểu diễn các ví dụ huấn luyện thuộc lớp còn lại. Ví dụ cần phân loại được biểu diễn bởi hình tròn. Khoảng cách giữa các ví dụ là khoảng cách Ơ clit trên mặt phẳng. Với $k = 3$, ví dụ được phân loại thành tam giác do có 2 trong số 3 láng giềng gần nhất là tam giác, trong khi với $k = 5$, ví dụ đang xét được phân loại thành hình vuông do có 3 trong số 5 láng giềng gần nhất là hình vuông. Với $k = 1$, ví dụ sẽ được phân loại thành tam giác.



Hình 5.11. Kết quả phân loại của k-NN với $k = 3$ và $k = 5$.

Mặc dù không xây dựng hàm phân loại một cách tường minh, ta vẫn có thể xét hàm phân loại do k-NN tạo ra. Với một tập cố định các ví dụ huấn luyện và giả sử $k = 1$ (1-NN), hàm phân loại được thể hiện như trên hình 5.12, trong đó mỗi điểm biểu diễn một ví dụ huấn luyện. Với $k = 1$, mỗi khối đa diện bao quanh một ví dụ huấn luyện thể hiện vùng không gian có cùng nhãn phân loại với ví dụ đó. Tức là tất cả ví dụ nằm trong đa diện của ví dụ huấn luyện nào sẽ có nhãn phân loại giống ví dụ huấn luyện đó. Biểu diễn này được gọi là đồ thị Voronoi.



Hình 5.11. Hàm phân loại tạo ra từ thuật toán k-NN với k = 1.

Hồi quy (regression)

Thuật toán k-NN có thể dùng cho trường hợp hồi quy, trong đó mỗi mẫu x có thể nhãn phân loại $f(x)$ với $f(x)$ là một số thực. Thuật toán k-NN ở trên có thể thay đổi dễ dàng cho bài toán hồi quy này bằng cách thay bước đánh số 3 trong thuật toán hình 5.10 như sau:

$$f'(x) = \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Thuật toán k-NN có một tham số đầu vào là k: số hàng xóm được dùng để quyết định nhãn cho mẫu đang xét. Nếu $k = 1$, giá trị hàm $f'(x)$ được chọn bằng giá trị hàm f của mẫu gần nhất. Thông thường $k = 1$ không cho kết quả tốt do hàng xóm gần nhất có ảnh hưởng quyết định tới giá trị $f'(x)$. Trong trường hợp hàng xóm gần nhất là nhiễu sẽ khiến kết quả bị sai. Nhiều nghiên cứu cho thấy giá trị k trong khoảng từ 5 đến 10 là phù hợp. Để xác định giá trị cụ thể của k có thể sử dụng phương pháp kiểm tra chéo như đã trình bày ở phần tỉa cây. Giá trị k cho độ chính xác khi kiểm tra chéo tốt nhất sẽ được lựa chọn cho thuật toán.

5.4.3. Một số lưu ý với thuật toán k-NN

a) Các độ đo khoảng cách và độ tương tự

Khoảng cách Ơ clit là độ đo thông dụng để tính khoảng cách giữa các ví dụ. Bên cạnh đó có thể sử dụng những độ đo khác.

a. *Khoảng cách Mahalanobis*. Khoảng cách Mahalanobis cho phép tính độ tương quan giữa các thuộc tính và được sử dụng trong trường hợp các thuộc tính được biểu diễn theo những thang khác nhau, chẳng hạn khi hai thuộc tính là chiều cao và cân nặng. Trong trường hợp đó, khoảng cách Mahalanobis cho phép chuẩn hóa khoảng cách, cân bằng đóng góp của hai thuộc tính.

b. *Khoảng cách Hamming*. Khoảng cách Ơ clit không thể sử dụng được nếu thuộc tính nhận giá trị rời rạc. Trong trường hợp này có thế sử dụng khoảng cách Hamming, được tính bằng số thuộc tính có giá trị khác nhau.

b) Sử dụng trọng số cho các hàng xóm

Theo thuật toán k-NN trình bày ở trên cho bài toán hồi quy, mỗi hàng xóm có đóng góp như nhau vào giá trị của ví dụ mới. Tuy nhiên, sẽ hợp lý hơn nếu cho phép những hàng xóm ở gần có ảnh hưởng nhiều hơn tới giá trị dự đoán. Có thể thực hiện điều này bằng cách nhân giá trị hàng xóm với trọng số, trọng số càng lớn nếu hàng xóm ở càng gần ví dụ cần dự đoán. Ví dụ, trọng số có thể tính bằng nghịch đảo của bình phương khoảng cách tới ví dụ cần dự đoán, tức là:

$$w_i = \frac{1}{d(x, x_i)^2}$$

Thêm trọng số vào công thức hồi quy, giá trị hàm $f'(x)$ của ví dụ x được tính như sau:

$$f'(x) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

Trong trường hợp này, mẫu số là tổng trọng số và cho phép chuẩn hóa độ đóng góp của từng hàng xóm.

Với việc sử dụng trọng số, có thể không cần giới hạn số lượng hàng xóm do những ví dụ ở càng xa sẽ có ảnh hưởng càng nhỏ tới giá trị hàm đích của ví dụ mới. Tuy nhiên, việc không giới hạn số lượng hàng xóm đòi hỏi tính toán nhiều và do vậy tốc độ của thuật toán sẽ bị ảnh hưởng.

c) Ảnh hưởng của thuộc tính tới thuật toán

Để tính khoảng cách, k-NN sử dụng toàn bộ thuộc tính của ví dụ, bất kể thuộc tính có liên quan tới nhãn phân loại của ví dụ hay không. Đây là điểm khác với phương pháp học cây quyết định, trong đó chỉ những thuộc tính liên quan được chọn trên các nút, hay phương pháp Bayes đơn giản, trong đó chỉ những thuộc tính liên quan mới có xác suất điều kiện cao. Nếu dữ liệu bao gồm cả những thuộc tính không liên quan tới nhãn phân loại, những thuộc tính này sẽ ảnh hưởng tới khoảng cách. Ví dụ, giả sử dữ liệu có 100 thuộc tính, trong đó chỉ có 2 thuộc tính có ảnh hưởng tới nhãn phân loại. Khi đó những ví dụ có hai thuộc tính này giống nhau vẫn có thể nằm rất xa nhau trong không gian 100 chiều.

Ảnh hưởng của số lượng thuộc tính lớn và không liên quan làm giảm đáng kể độ chính xác của k-NN. Để giải quyết vấn đề này có thể sử dụng hai phương pháp:

c. Đánh trọng số cho thuộc tính sao cho thuộc tính ít liên quan có trọng số nhỏ và ngược lại.

d. Lựa chọn thuộc tính (hay còn gọi là trích chọn đặc trưng): chỉ những thuộc tính liên quan được giữ lại, trong khi những thuộc tính không liên quan bị bỏ đi. Đây là trường hợp riêng phương pháp đánh trọng số cho thuộc tính, trong đó những thuộc tính bị loại có trọng số bằng không.

Có rất nhiều nghiên cứu đề cập tới việc lựa chọn và đánh trọng số cho thuộc tính. Do giới hạn của môn học, các nội dung này sẽ không được đề cập tới ở đây.

5.5. HỒI QUY TUYẾN TÍNH VÀ HỒI QUY LOGISTIC

Trong phần này ta sẽ xem xét một số kỹ thuật phân loại và hồi quy sử dụng các *mô hình tuyến tính* hoặc dạng tổng quát hoá của mô hình này. Cụ thể, hàm đích cần xây dựng là tổ hợp tuyến tính các thuộc tính, tức là tổng các thuộc tính có nhân với trọng số. Mô hình dạng này có thể sử dụng trong trường hợp đặc trưng là các số thực. Trước hết, ta sẽ xem xét phương pháp hồi quy tuyến tính dùng cho bài toán hồi quy. Sau đó, ta sẽ xem xét kỹ thuật phân loại sử dụng hồi quy logistic – một dạng khái quát của mô hình tuyến tính.

5.5.1. Hồi quy tuyến tính

Trong phần này sẽ trình bầy về kỹ thuật *hồi quy tuyến tính* (linear regression). Đây là phương pháp hồi quy đơn giản nhưng có ứng dụng khá rộng. Ngoài ra, phương pháp này được chọn trình bầy do có liên quan tới một lớp các phương pháp phân loại khác, đặc biệt là phương pháp phân loại có tên *hồi quy logistic* (logistic regression). Cần lưu ý rằng, mặc dù hồi quy và phân loại là hai bài toán khác nhau, song có thể dễ dàng sử dụng kỹ thuật hồi quy cho phân loại như sẽ trình bầy trong phần tiếp sau. Chẳng hạn, có thể sử dụng một biến với giá trị số để biểu diễn nhãn phân loại, sao cho mỗi giá trị của biến này ứng với một nhãn. Sau đó có thể sử dụng thuật toán hồi quy để dự đoán giá trị cho biến này và tùy thuộc vào giá trị dự đoán được mà quyết định nhãn phân loại.

Giả sử ta có bộ dữ liệu về diện tích, số phòng tắm, và giá bán nhà như trong bảng sau.

Diện tích (m^2)	Số phòng tắm	Giá bán (triệu đồng)
100	2	2400

80	2	2000
75	1	1900
80	1	1950
110	2	2600
90	2	2300
…	…	…

Hình 5.12. Dữ liệu huấn luyện cho bài toán hồi quy

Nhắc lại, dữ liệu trên được cho dưới dạng các cặp ($\mathbf{x}_i$, y_i), trong đó i là số thứ tự của ví dụ thứ i, vec tơ đặc trưng $\mathbf{x}_i$ có hai phần tử x_{i1} và x_{i2} tương ứng với diện tích và số phòng tắm. Trong trường hợp tổng quát, dữ liệu gồm m ví dụ, mỗi ví dụ được biểu diễn bởi n đặc trưng liên tục.

Từ dữ liệu trên cần tìm mô hình cho phép xác định quan hệ của giá nhà với các đặc trưng diện tích và số phòng tắm, từ đây dự đoán giá nhà nếu biếu giá trị của hai đặc trưng này.

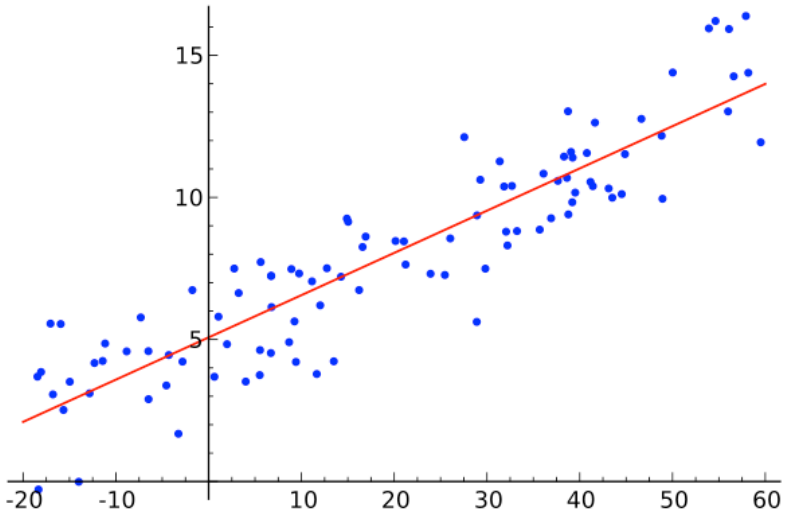
Mô hình hồi quy tuyến tính. Để xây dựng mô hình học có giám sát, trước tiên cần lựa chọn dạng mô hình, hay dạng hàm đích, chẳng hạn mô hình có dạng cây như trong phần học cây quyết định. Trong hồi quy tuyến tính, mô hình sử dụng là mô hình tuyến tính có dạng sau:

$$h(x) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

Trong công thức này, giá trị biến đầu ra được tính bằng tổng của các thuộc tính, mỗi thuộc tính nhân với một tham số β_i. Để cho đơn giản, ta có thể bổ sung tham số giả $x_0 = 1$. Khi đó, công thức trên có thể viết lại như sau:

$$h(x) = \sum_{i=0}^{n} \beta_i x_i$$

trong đó n là số đặc trưng đầu vào. Trong trường hợp $n = 1$, công thức trên là phương trình đường thẳng, trong trường hợp $n = 2$ là phương trình mặt phẳng, trong trường hợp $n > 2$, công thức trên xác định một *siêu phẳng* (hyperplane). Hình 5.13 minh họa cho đường thẳng trong trường hợp có một thuộc tính đầu vào.



Hình 5.13. Hồi quy tuyến tính với một đặc trưng đầu vào

Hàm lỗi.

Mục đích của giai đoạn huấn luyện là xác định giá trị các tham số β_i sao cho mô hình phù hợp với dữ liệu, tức là sao cho giá trị $h(x)$ mà mô hình tính được càng ít lệch với giá trị của y càng tốt trên các ví dụ huấn luyện. Độ lệch giữa $h(x)$ và y được tính bằng $(h(x)-y)^2$. Trên cơ sở đó, ta có thể xác định một hàm cho phép xác định độ phù hợp của mô hình trên tất cả các ví dụ huấn luyện bằng cách lấy tổng bình phương độ lệch giữa $h(x)$ và y trên toàn tập dữ liệu huấn luyện. Hàm này được gọi là *hàm giá* (cost function) hoặc *hàm lỗi* (error function), và được tính như sau:

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{m}(h(x_i)-y_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{m}(\sum_{j=0}^{n}\beta_j x_{ij} - y_i)^2$$

trong đó m là số ví dụ trong tập huấn luyện. *Mục tiêu của việc huấn luyện mô hình* là xác định các tham số β_i sao cho hàm lỗi J đạt giá trị nhỏ nhất. Dưới đây ta sẽ xem xét hai cách để xác định các tham số này.

Xác định tham số bằng cách giải phương trình tuyến tính

Cách thứ nhất để xác định giá trị tham số β_i làm cho hàm lỗi J nhỏ nhất là lấy đạo hàm J theo β_i, sau đó đặt đạo hàm bằng không và giải phương trình nhận được. Để thuận tiện cho việc trình bầy, ta sẽ sử dụng cách biểu diễn sử dụng vec tơ và ma trận. Các giá trị đích y_i được biểu diễn bằng vec tơ cột **y** với m phần tử. Đặc trưng đầu vào x_{ij} được biểu diễn bằng ma trận $\boldsymbol{X}$ kích thước m x n, trong đó mỗi dòng ứng với một ví dụ huấn luyện, mỗi cột tương ứng với một đặc trưng. Tiếp theo, tham số β_i được biểu diễn dưới dạng vec tơ cột $\boldsymbol{\beta}$ với n phần tử.

Sử dụng biểu diễn ma trận, hàm lỗi J được viết lại như sau

$$J(\beta) = \frac{1}{2}(\mathbf{X\beta} - \mathbf{y})^T(\mathbf{X\beta} - \mathbf{y})$$

Tính đạo hàm của J đối với $\boldsymbol{\beta}$ và cho đạo hàm bằng không, ta được hệ phương trình tuyến tính:

$$\mathbf{X}^T(\mathbf{X\beta} - \mathbf{y}) = 0$$

Giải phương trình tuyến tính, đa được lời giải như sau:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Lưu ý, chỉ có thể tìm được lời giải trên nếu có thể tìm được ma trận nghịch đảo $(\mathbf{X}^T\mathbf{X})^{-1}$ của ma trận $(\mathbf{X}^T\mathbf{X})$, hay ma trận $(\mathbf{X}^T\mathbf{X})$ là ma trận khả nghịch (có định thức khác 0).

Sau khi tìm được $\boldsymbol{\beta}$, với một ví dụ mới có vec tơ đặc trưng là $\mathbf{x}$, giá trị đích (dự đoán) được tính bằng $\mathbf{x}^T\boldsymbol{\beta}$.

Xác định tham số bằng phương pháp giảm gradient

Thay vì giải hệ phương trình tuyến tính để xác định tham số $\boldsymbol{\beta}$ như trình bầy ở trên, có thể xác định tham số bằng cách sử dụng phương pháp giảm gradient (gradient descent). Gradient của một hàm khả vi là tập hợp các đạo hàm riêng của hàm đó. Có thể coi gradient như một vec tơ hướng về phía giá trị thay đổi lớn nhất của hàm tương ứng. Như vậy di chuyển theo hướng gradient cho phép tiến nhanh nhất tới cực trị của hàm. Cực trị này có thể là cực trị địa phương hay cực trị toàn cục dựa trên hàm cụ thể.

Dựa trên tính chất này, thuật toán giảm gradient thực hiện như sau. Bắt đầu từ một giá trị ban đầu của tham số ($\boldsymbol{\beta}$ trong trường hợp này), thường là giá trị dương nhỏ được chọn ngẫu nhiên. Thuật toán cập nhật dần tham số bằng cách trừ đi một đại lượng tỷ lệ với giá trị của gradient, sau đó tính lại giá trị hàm mục tiêu. Quá trình này được lặp lại nhiều lần cho đến khi thuật toán hội tụ, tức là đạt tới cực trị của hàm mục tiêu và do vậy gradient bằng 0.

Trong trường hợp hồi quy tuyến tính đang xét, tại mỗi bước lặp, thuật toán thực hiện cập nhật sau với mỗi tham số β_j:

$$\beta_j \leftarrow \beta_j - \alpha\frac{\partial}{\partial\beta_j}J(\beta)$$

trong đó α là tham số có tên là *tốc độ học*. Giá trị α cho phép thay đổi mức độ cập nhật tham số sau mỗi bước học. Giá trị này càng lớn thì tham số thay đổi càng nhanh, tức là càng mau tiến tới cực trị. Tuy nhiên, đặt tốc độ học lớn có thể dẫn tới hiện tượng tham số nhảy qua giá trị tối ưu khiến cho thuật toán không hội tụ và không có lời giải tốt nhất. Ngược lại, đặt tốc độ học quá nhỏ sẽ làm quá trình huấn luyện chậm lại. Trong trường hợp $\alpha = 0$, tham số sẽ không thay đổi.

Để thực hiện thuật toán, ta cần xác định đạo hàm riêng của J theo β_i. Để cho đơn giản, tạm giả thiết dữ liệu huấn luyện chỉ gồm một ví dụ duy nhất ($\mathbf{x}$, y). Đạo hàm riêng theo β_j khi đó được tính như sau:

$$\frac{\partial}{\partial \beta_j} J(\beta) = \frac{\partial}{\partial \beta_j} \frac{1}{2} (h(\mathbf{x}) - y)^2$$

$$= 2.\frac{1}{2} (h(\mathbf{x}) - y) \frac{\partial}{\partial \beta_j} (\sum_{i=0}^{n} \beta_i x_i - y)$$

$$= (h(\mathbf{x}) - y) x_j$$

Như vậy, với mỗi ví dụ $(\mathbf{x}_i, y_i)$, giá trị tham số β_i được cập nhật theo quy tắc sau:

$$\beta_j \leftarrow \beta_j + \alpha(y_i - h(\mathbf{x}_i)) x_{ij}$$

Quy tắc này được gọi là quy tắc *bình phương trung bình nhỏ nhất* (Least Mean Squares), viết tắt là LMS. Quy tắc này còn có tên gọi khác là quy tắc học Widrow-Hoff. Có thể thấy, giá trị của mỗi lần cập nhật tỷ lệ thuận với giá trị sai số $(y_i - h(\mathbf{x}_i))$. Như vậy, với những ví dụ huấn luyện mà giá trị dự đoán trùng với giá trị đích thì không xảy ra cập nhật. Ngược lại, nếu sai số $(y_i - h(\mathbf{x}_i))$ lớn, tức là giá trị dự đoán lệch nhiều so với giá trị thực thì mức độ cập nhật tham số sẽ lớn hơn.

Tiếp theo, cần mở rộng quy tắc học trên cho nhiều ví dụ huấn luyện. Có hai quy tắc học cho trường hợp dữ liệu gồm nhiều ví dụ. Quy tắc thứ nhất được gọi là quy tắc *giảm gradient theo mẻ* (batch gradient descent) và có dạng như sau:

For (j = 0 to n) {

$$\beta_j \leftarrow \beta_j + \alpha \sum_{i=1}^{m} (y_i - h(\mathbf{x}_i)) x_{ij}$$

}

Quá trình cập nhật này được thực hiện cho tới khi hội tụ, tức là khi β_j không còn thay đổi hoặc thay đổi không đáng kể. Dễ dàng nhận thấy, đại lượng $\sum_{i=1}^{m} (y_i - h(\mathbf{x}_i)) x_{ij}$ chính là giá trị đạo hàm riêng $\frac{\partial}{\partial \beta_j} J(\beta)$. Theo quy tắc này, tại mỗi bước lặp, thuật toán sử dụng toàn bộ ví dụ huấn luyện để tính toán giá trị cập nhật. Cách này thường có thời gian học tương đối dài.

Trong trường hợp nói chung, với một hàm bất kỳ, thuật toán giảm gradient sẽ tìm được cực tiểu địa phương. Tuy nhiên, do hàm mục tiêu trong hồi quy tuyến tính là hàm bậc hai với một điểm cực tiểu duy nhất, thuật toán giảm gradient theo mẻ sẽ cho phép hội tụ về giá trị cực tiểu toàn thể này.

Quy tắc cập nhật thứ hai gọi là quy tắc giảm gradient ngẫu nhiên (stochastic gradient descent) và được thực hiện như sau:

For (i = 1 to m) {

    For (j = 0 to n) {

$$\beta_j \leftarrow \beta_j + \alpha(y_i - h(\mathbf{x}_i)) x_{ij}$$

    }

}

Khác với học theo mẻ, thuật toán giảm gradient ngẫu nhiên lần lượt xét từng ví dụ huấn luyện. Với mỗi ví dụ, thuật toán tiến hành cập nhật ngay bộ tham số theo gradient tính được theo giá trị lỗi cho mình ví dụ đó. Như vậy, thay vì xét toàn bộ ví dụ huấn luyện trước khi tiến hành cập nhật tham số như trong phương pháp theo mẻ, thuật toán ngẫu nhiên bắt đầu cập nhật tham số ngay khi gặp mỗi ví dụ huấn luyện. Trên thực tế, phương pháp này cho phép đạt tới giá trị tham số gần tối ưu nhanh hơn nhiều so với giảm gradient theo mẻ. Trong nhiều trường hợp, thuật toán ngẫu nhiên không cho phép đạt được giá trị tham số tối ưu do giá trị này chỉ dao động ở gần mức tối ưu, đặc biệt nếu tham số tốc độ học lớn. Tuy nhiên, trên thực tế, giá trị xấp xỉ tối ưu là có thể chấp nhận được và do vậy phương pháp ngẫu nhiên thường được ưu tiên lựa chọn nếu bộ dữ liệu huấn luyện lớn. Một kỹ thuật khác cũng có thể sử dụng để đạt được tham số tối ưu là giảm dần tốc độ học α theo thời gian. Khi đó, tham số sẽ đạt được cực tiểu thay vì dao động quanh đó.

5.5.2. Hồi quy logistic

Mặc dù có tên gọi là hồi quy, thuật toán *hồi quy logistic* là thuật toán phân loại và được dùng cho bài toán phân loại hai lớp. Thuật ngữ logistic xuất phát từ hàm logit được dùng để biểu diễn mô hình phân loại và không liên quan tới từ logistic có nghĩa là hậu cần.

Trong bài toán phân loại hai lớp, đầu ra hay hàm đích có thể nhận một trong hai nhãn phân loại. Để cho tiện ta sẽ ký hiệu hai nhãn này là 0 và 1, tương ứng với nhãn âm và nhãn dương (trong một số trường hợp có thể sử dụng giá trị –1 cho nhãn âm và +1 cho nhãn dương, tuy nhiên trong phần này sẽ dùng 0 và 1).

Mô hình

Để phân loại hai lớp, ta có thể sử dụng thuật toán hồi quy tuyến tính trình bầy ở trên nếu coi nhãn 0 và 1 là các số. Tuy nhiên, cách sử dụng hồi quy tuyến tính cho phân loại cho kết quả không tốt trong nhiều trường hợp. Ngoài ra, hồi quy tuyến tính có thể cho kết quả không mong muốn là kết quả đầu ra lớn hơn 1 hoặc nhỏ hơn 0.

Để giới hạn giá trị đầu ra luôn nằm trong khoảng [0, 1], thay vì sử dụng mô hình tuyến tính, hồi quy logstic sử dụng mô hình sau:

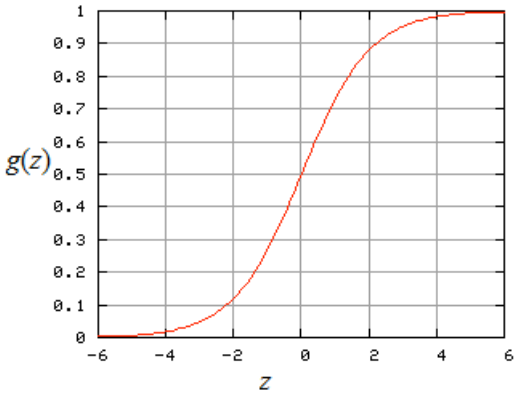
$$h(x) = g(\beta^T x) = \frac{1}{1 + e^{-\beta^T x}}$$

Trong đó

$\beta^T x = \sum_{j=0}^{n} \beta_j x_j$ với $x_0 = 1$ tương tự như với hồi quy tuyến tính,

và $g(z) = \frac{1}{1 + e^{-z}}$

là hàm *logit* hay hàm *sigmoid*. Đồ thị của hàm logit $g(z)$ có dạng như sau:



Hình 5.14. Đồ thị hàm logit

Hàm logit có hai tính chất quan trọng, có ảnh hưởng tới việc chọn hàm này trong hồi quy logistic. Thứ nhất, như đồ thị trong hình trên cho thấy, giá trị hàm logit $g(z)$ tiến tới 1 khi z tiến tới $+\infty$ và $g(z)$ tiến tới 0 khi z tiến tới $-\infty$. Như vậy, bằng cách sử dụng hàm logit, giá trị hàm đích $h(x)$ luôn nằm trong khoảng [0, 1].

Thứ hai, ngoài việc hàm logit luôn có giá trị nằm trong khoảng [0, 1], hàm này là hàm khả vi với công thức tính đạo hàm g' tương đối đơn giản:

$$g'(z) = g(z)(1 - g(z))$$

Như ta sẽ thấy, đạo hàm sẽ được sử dụng để tính gradient trong quá trình huấn luyện và ước lượng tham số của mô hình.

Do giá trị $h(x)$ luôn nằm trong khoảng [0, 1], phương pháp hồi quy logistic sử dụng giá trị này như xác suất đầu ra $y = 1$ khi biết giá trị đầu vào x và tham số β, tức là:

$$P(y = 1 \mid x, \beta) = h(x) = g(\beta^T x)$$

Do đầu ra chỉ có thể nhận hai giá trị 0 hoặc 1 và tổng xác suất bằng 1 nên ta có xác suất đầu ra $y = 0$ là:

$$P(y = 0 \mid x, \beta) = 1 - h(x) = 1 - g(\beta^T x)$$

Như vậy, giá trị đầu ra dự đoán được xác định bằng cách so sánh $h(x)$ với một ngưỡng, chẳng hạn 0.5. Nếu $h(x) > 0.5$, tương đương với xác suất $y = 1$ lớn hơn 50% thì nhãn đầu ra nhận giá trị bằng 1. Nếu ngược lại, nhãn phân loại đầu ra sẽ nhận giá trị 0. Trong một số trường hợp ta có thể tăng hoặc giảm ngưỡng này. Ví dụ, trong bài toán phát hiện thư rác, nếu coi thư rác có nhãn là 1 và thư thường có nhãn là 0 thì yêu cầu quan trọng là không phân loại nhầm thư thường thành thư rác. Khi đó ta có thể đặt ngưỡng cao hơn, chẳng hạn bằng 0.8, tức là nếu mô hình xác định xác suất thư là thư rác lớn hơn 80% thì thư mới được xếp vào loại thư rác.

Huấn luyện mô hình

Trong quá trình huấn luyện cần xác định các tham số β cho phép tối ưu một hàm mục tiêu nào đó. Trong phương pháp hồi quy tuyến tính trình bầy ở phần trên ta có thể chọn làm mục tiêu là hàm tổng bình phương lỗi, sau đó tìm được tham số cho phép cực tiểu hóa hàm này bằng phương pháp phân tích hoặc phương pháp gradient giảm dần. Tuy nhiên, hàm logistic phức tạp hơn và không thể tìm được lời giải bằng phương pháp tương tự như trong

hồi quy tuyến tính. Với hồi quy logistic, việc ước lượng tham số khi huấn luyện mô hình được thực hiện bằng thủ tục *độ phù hợp cực đại* (maximum likelihood).

Độ phù hợp (likelihood) $L(\beta)$ của tham số β được định nghĩa là xác suất điều kiện của các giá trị đầu ra **y** khi biết giá trị đầu vào **X** và tham số β.

$$L(\beta) = P(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta})$$

Khi xây dựng mô hình, ta cần xác định tham số β sao cho giá trị đầu ra dự đoán gần với giá trị đầu ra thực trên dữ liệu huấn luyện, hay xác suất giá trị nhãn đầu ra khi biết đầu vào X và tham số β càng lớn càng tốt, tức là độ phù hợp xác định theo công thức trên càng lớn càng tốt. Nguyên tắc xác định tham số cho độ phù hợp lớn nhất được gọi là nguyên lý độ phù hợp cực đại.

Để xác định tham số β, ta sẽ triển khai công thức tính $L(\beta)$. Trước hết, hai công thức

$$P(y = 1 \mid x, \beta) = h(x)$$

$$P(y = 0 \mid x, \beta) = 1 - h(x)$$

có thể viết gọn thành

$$P(y \mid x, \beta) = (h(x))^{y} (1 - h(x))^{1-y}$$

Tiếp theo, do có thể coi các ví dụ huấn luyện là độc lập xác suất với nhau, ta có:

$$L(\beta) = P(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta})$$

$$= \prod_{i=1}^{m} P(y_i \mid x_i, \beta) \quad \text{//bằng tích xác suất cho từng ví dụ huấn luyện}$$

$$= \prod_{i=1}^{m} (h(x_i))^{y_i} (1 - h(x_i))^{1-y_i}$$

Để tiện cho việc tính toán, thay vì cực đại hóa $L(\beta)$ ta có thể cực đại hóa log $L(\beta)$ do làm logarit là hàm đơn điệu tiến:

$$l(\beta) = L(\beta)$$

$$= \sum_{i=1}^{m} \left[y_i \log h(x_i) + (1 - y_i) \log(1 - h(x_i)) \right]$$

Để cực đại hóa làm $l(\beta)$, ta có thể dùng phương pháp *gradient tăng dần* (gradient ascent) với công thức cập nhật tham số như sau:

$$\beta_j \leftarrow \beta_j + \alpha \frac{\partial}{\partial \beta_j} l(\beta)$$

trong đó α là tốc độ học. Công thức này tương tự như trong trường hợp hồi quy tuyến tính. Tuy nhiên, thay vì trừ đi giá trị gradient nhân với tốc độ học, trong trường hợp này đại lượng đó được cộng vào giá trị tham số trước đó do ta cần cực đại hóa hàm mục tiêu chứ không phải cực tiểu hóa như trong hồi quy tuyến tính. Do vậy, phương pháp này gọi là gradient tăng dần.

Tương tự kỹ thuật sử dụng trong hồi quy tuyến tính, trước hết, ta tính đạo hàm riêng cho trường hợp chỉ có một ví dụ huấn luyện (x, y):

$\frac{\partial}{\partial \beta_i} l(\beta) = (y - h(\mathbf{x}))x_j$ //ở đây bỏ qua các bước biến đổi, người đọc có thể tự thực hiện

Trong trường hợp dữ liệu huấn luyện gồm nhiều ví dụ, công thức trên cho phép xây dựng quy tắc tăng gradient ngẫu nhiên (stochastic gradient ascent) như sau:

$$\beta_j \leftarrow \beta_j + \alpha(y_i - h(\mathbf{x}_i))x_{ij}$$

Cần chú ý rằng, công thức này rất giống với công thức giảm gradient ngẫu nhiên sử dụng trong hồi quy tuyến tính. Điểm khác nhau duy nhất là hàm $h(x)$ không còn là hàm tuyến tính của x như trường hợp trên.

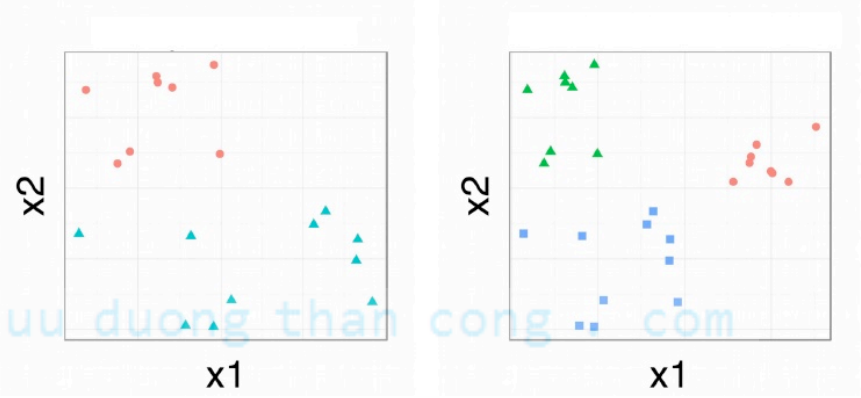
Phương pháp phân loại sử dụng hồi quy logistic là phương pháp phân loại theo nguyên tắc xác suất có độ chính xác tương đối cao. Nhược điểm của phương pháp này là thời gian huấn luyện lâu hơn so với một số phương pháp phân loại khác. Ngoài ra, phương pháp này cũng đòi hỏi lựa chọn thuộc tính đầu vào phù hợp.

5.5.3. Hồi quy logistic cho phân loại đa lớp

Phương pháp hồi quy logistic trình bầy trong phần trước chỉ dùng được trong phân loại hai lớp, với nhãn đầu ra nhận một trong hai giá trị. Trong phần này, ta sẽ xem xét cách mở rộng hồi quy logistic cho phân loại đa lớp, tức là khi nhãn đầu ra có thể nhận một trong nhiều (hơn hai) giá trị. Các trường hợp phân loại đa lớp rất thường gặp trên thực tế, ví dụ một bài báo có thể phân vào một trong các loại: chính trị, xã hội, thể thao, văn hóa, giáo dục, giải trí.

Trên hình 5.15 là ví dụ minh họa cho phân loại hai lớp và ba lớp. Để đơn giản cho việc minh họa bằng, các ví dụ trong hình này được biểu diễn bằng hai đặc trưng.

Sau đây ta sẽ xem xét một phương pháp mở rộng mô hình hồi quy tuyến tính cho phân loại đa lớp.



Hình 5.15. Phân loại hai lớp (trái) và đa lớp (phải)

Sử dụng nhiều mô hình hồi quy logistic

Cách đơn giản nhất để thực hiện phân loại đa lớp là kết hợp nhiều mô hình hồi quy logistic thông thường, tức là mô hình phân loại hai lớp, để tạo ra mô hình đa lớp theo sơ đồ *một chọi tất cả* (one vs. all).

Giả sử cho bài toán phân loại với k lớp, mỗi lớp có nhãn phân loại là 1, 2, …k. Ta sẽ xây dựng k mô hình hồi quy logistic như sau. Từ bộ dữ liệu huấn luyện, ta tạo ra k bộ dữ liệu cho k bài toán phân loại hai lớp như minh họa trên hình 5.16:

Bộ dữ liệu thứ 1: các ví dụ với nhãn 1 (hình tam giác trong hình minh họa) được chọn làm nhãn dương (1), tất cả ví dụ còn lại làm nhãn âm (0). Dùng bộ dữ liệu này huấn luyện mô hình $h^{(1)}(x)$ để dự đoán lớp 1.

Bộ dữ liệu thứ 2: các ví dụ với nhãn 2 (hình tròn trong hình minh họa) được chọn làm nhãn dương (1) , tất cả ví dụ còn lại làm nhãn âm (0). Dùng bộ dữ liệu này huấn luyện mô hình $h^{(2)}(x)$ để dự đoán lớp 2.

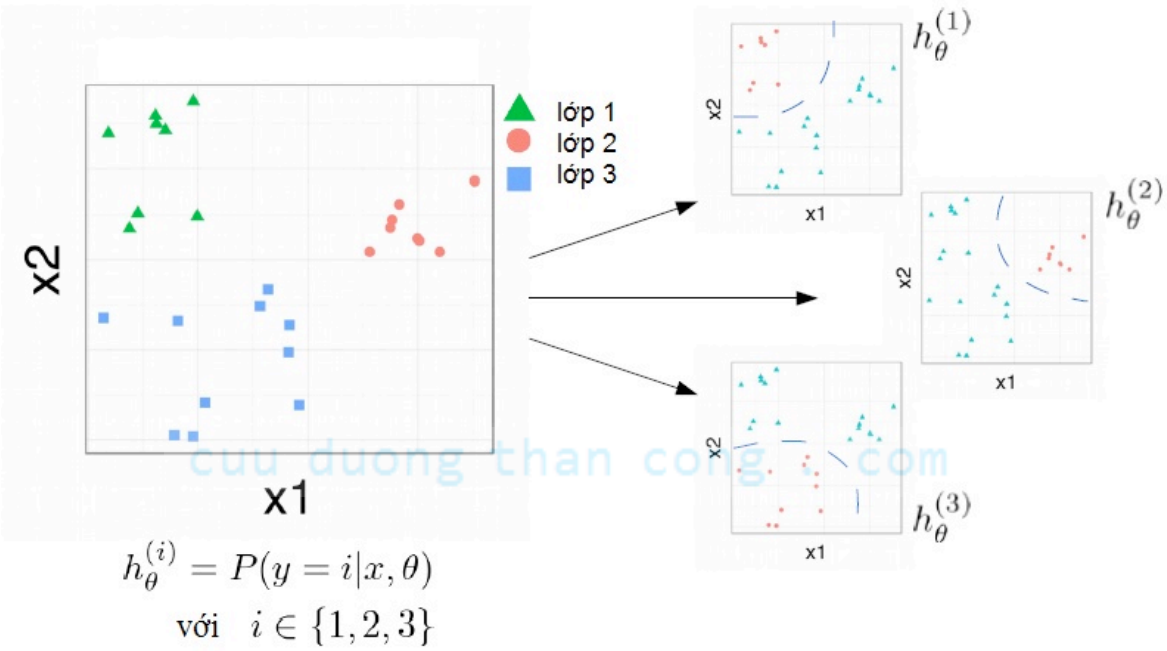
….

Bộ dữ liệu thứ k: các ví dụ với nhãn k được chọn làm nhãn dương (1) , tất cả ví dụ còn lại làm nhãn âm (0). Dùng bộ dữ liệu này huấn luyện mô hình $h^{(k)}(x)$ để dự đoán lớp k.

Như vậy, ta có xác suất một ví dụ nhận nhãn bằng i, $i = 1…k$ là: $P(y = i \mid x) = h^{(i)}(x)$.

Với mỗi ví dụ mới x, nhãn phân loại là nhãn có giá trị xác suất lớn nhất:

$$y = \arg\max_i P(y = i \mid x)$$
$$= \arg\max_i h^{(i)}(x)$$



Hình 5.16. Kết hợp nhiều mô hình hai lớp theo sơ đồ một chọi tất cả (one vs all)

Phân loại đa lớp sử dụng hồi quy tuyến tính đòi hỏi thời gian huấn luyện lâu hơn so với Bayes đơn giản và cũng cần nhiều dữ liệu huấn luyện hơn. Phương pháp phân loại này cho độ

chính xác tốt hơn so với Bayes đơn giản tuy nhiên đòi hỏi lựa chọn đặc trưng cẩn thận. Trong trường hợp các đặc trưng có quan hệ tỷ lệ tuyến tính với nhau, độ chính xác khi phân loại sẽ bị ảnh hưởng.

5.6. SUPPORT VECTOR MACHINES

Support vector machines (SVM) (một số tài liệu dịch là *máy véc tơ tựa*) là kỹ thuật học có giám sát tương đối mới, được đề xuất lần đầu vào năm 1992 và được áp dụng nhiều từ khoảng cuối những năm chín mươi thế kỷ trước. SVM được đề xuất ban đầu cho bài toán phân loại nhị phân, và có thể mở rộng cho phân loại đa lớp tương tự như hồi quy logistic.

Hiện nay, theo nhiều tài liệu, SVM là phương pháp học có giám sát *phổ biến nhất*. Nhiều thư viện hoặc gói phần mềm cho phép sử dụng ngay SVM cho bài toán học có giám sát (phân loại hoặc hồi quy), người sử dụng chỉ cần chọn rất ít tham số để có thể sử dụng các thư viện này. SVM cũng cho độ chính xác cao, thường ở trong nhóm đứng đầu trong số thuật toán phân loại, trong nhiều ứng dụng. Nhìn chung, khi có bài toán học có giám sát, nếu không có thêm thông tin gì đặc biệt thì thuật toán đầu tiên nên thử là SVM.

Nguyên tắc chung

SVM dựa trên hai nguyên tắc chính:

- Thứ nhất, SVM tìm cách phân chia ví dụ có nhãn khác nhau bằng một siêu phẳng sao cho khoảng cách từ siêu phẳng tới những ví dụ có nhãn khác nhau là lớn nhất. Nguyên tắc này được gọi là nguyên tắc *lề cực đại* (max margin). Trong quá trình huấn luyện, thuật toán SVM xác định siêu phẳng có lề cực đại bằng cách giải bài toán tối ưu cho một hàm mục tiêu bậc 2.
- Thứ hai, để giải quyết trường hợp các ví dụ không thể phân chia bằng một siêu phẳng, phương pháp SVM sẽ *ánh xạ không gian ban đầu của các ví dụ sang một không gian khác thường là có số chiều cao hơn*, sau đó tìm siêu phẳng với lề cực đại trong không gian này. Để tăng tính hiệu quả khi ánh xạ, một kỹ thuật được sử dụng là *kỹ thuật dùng hàm nhân* (kernel function) thay cho tích có hướng của các vec tơ.

Trong các phần tiếp theo, ta sẽ xem xét cụ thể cách triển khai hai nguyên tắc này.

5.6.1. Phân loại tuyến tính với lề cực đại

Bộ phân loại tuyến tính

SVM được phát triển từ mô hình phân loại tuyến tính. Do vậy, trước hết ta sẽ xem xét mô hình phân loại dạng này. Xét bài toán phân loại hai lớp, trong đó mỗi ví dụ được biểu diễn bằng *d đặc trưng liên tục* (số thực), và có thể nhận nhãn +1 (ví dụ dương) hoặc −1 (ví dụ âm). Tương tự các phần trước, giả sử tập dữ liệu huấn luyện gồm *n* ví dụ, trong đó ví dụ thứ *i* được xác định bởi $(\mathbf{x}_i, y_i)$, với $\mathbf{x}_i$ là vec tơ các đặc trưng và y_i là nhãn phân loại. Trong ví dụ trên hình 5.17, vec tơ $\mathbf{x}$ có hai chiều, do vậy mỗi mỗi ví dụ được biểu diễn như một điểm trong không gian hai chiều, các điểm đen tương ứng với ví dụ +1 và điểm trắng biểu diễn ví dụ −1. Trong phần tiếp theo, không gian hai chiều sẽ được sử dụng để tiện cho việc minh hoạ.

Tương tự trường hợp hồi quy tuyến tính, ta sẽ sử dụng hàm phân loại có dạng:

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$$

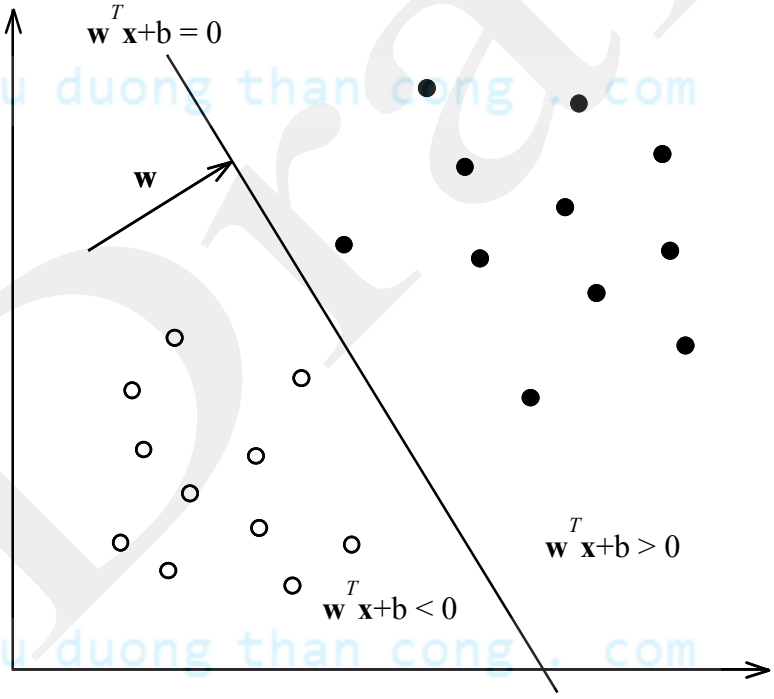
trong đó **w** là vec tơ trọng số và b là độ lệch. Để hiểu ý nghĩa các đại lượng này, ta xét trường hợp $b = 0$ trước. Khi đó, tập các điểm (hay vec tơ) **x** sao cho $\mathbf{w}^T\mathbf{t} = 0$ là một điểm trong không gian một chiều ($d = 1$), đường thẳng trong không gian hai chiều ($d = 2$), mặt phẳng trong không gian ba chiều, và *siêu phẳng* (hyperplane) trong trường hợp tổng quát. Siêu phẳng này đi qua gốc toạ độ và vuông góc với vec tơ **w**. Với $b \neq 0$, siêu phẳng này sẽ được đẩy khỏi gốc toạ độ một khoảng bằng $|b|/\|\mathbf{w}\|$, trong đó $\|\mathbf{w}\|$ là độ dài (chuẩn) của vec tơ **w**. Siêu phẳng $f(\mathbf{x})$ phân chia không gian thành hai phần, phần bên phải gồm những điểm **x** có $f(\mathbf{x}) > 0$ và phần bên trái gồm những điểm có $f(\mathbf{x}) < 0$ (xem hình 5.17).

Như vậy, một ví dụ **x** sẽ có nhãn phân loại y được xác định như sau:

$$y = sgn(f(\mathbf{x}))$$

trong đó sgn() là hàm dấu, nhận giá trị +1 nếu $f(\mathbf{x}) > 0$ và -1 nếu $f(\mathbf{x}) < 0$.

Cần lưu ý rằng, trong ví dụ trên hình 5.17, ta có thể tìm được siêu phẳng $f(\mathbf{x})$ sao cho các ví dụ dương nằm về một phía và ví dụ âm nằm về phía còn lại. Trong trường hợp như vậy, ta nói rằng dữ liệu *có thể phân chia tuyến tính*. Trường hợp dữ liệu không phân chia tuyến tính, tức là không tồn tại siêu phẳng như vậy, sẽ được đề cập sau.



Hình 5.17. Bộ phân loại tuyến tính

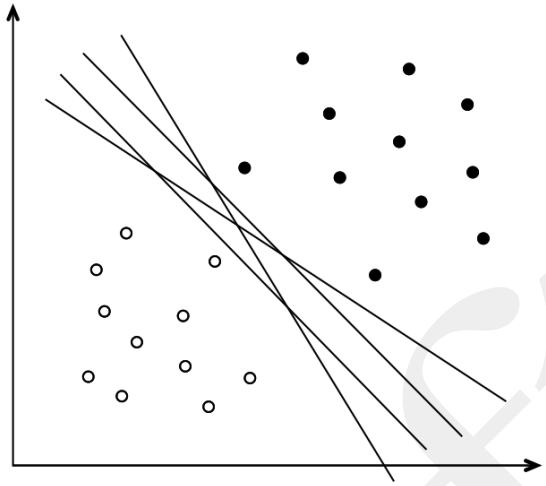
Phân loại tuyến tính với lề cực đại

Có thể nhận thấy, trong trường hợp dữ liệu phân chia tuyến tính, có thể tìm được nhiều siêu phẳng cho phép phân cách đúng ví dụ dương khỏi ví dụ âm như thể hiện trên hình 5.18. Vậy nên chọn siêu phẳng nào để làm hàm phân loại?

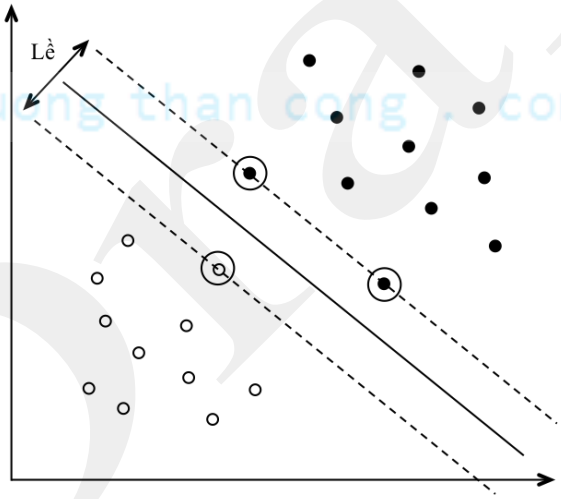
Gọi m_+ và m_- là khoảng cách từ siêu phẳng $f(\mathbf{x})$ tới ví dụ dương và ví dụ âm gần nhất. Các điểm nằm cách đều siêu phẳng các khoảng m_+ và m_- được thể hiện bởi hai đường đứt nét trên hình 5.19 và tạo ra một vùng ngăn cách các ví dụ khác dấu. Siêu phẳng được chọn nằm

giữa vùng phân cách, tức là $m_+ = m_-$. Độ rộng của vùng phân cách cách này, tức là $(m_+ + m_-)$ được gọi là "**lề**" ("margin") của siêu phẳng phân cách.

Nguyên tắc lề cực đại: *Trong số các siêu phẳng cho phép phân chia ví dụ khác dấu, SVM sử dụng siêu phẳng có lề lớn nhất làm bộ phân loại*



Hình 5.18. Tồn tại nhiều siêu phẳng phân cách ví dụ dương và ví dụ âm



Hình 5.19. Siêu phẳng phân cách. Các vec tơ tựa được khoanh tròn bên ngoài

Các ví dụ cho phép xác định ranh giới của lề (các điểm được khoanh tròn trên hình vẽ) được gọi là các *vec tơ tựa* (support vector).

Tại sao sử dụng siêu phẳng với lề cực đại? Khi xây dựng bộ phân loại, mục tiêu là giảm tối đa số ví dụ bị phân loại sai, gọi là *tổn thất* (lost). Trong giai đoạn huấn luyện, ta chỉ có thể tính tổn thất trên dữ liệu huấn luyện, được gọi là *tổn thất thực nghiệm* (empirical lost). Trong khi đó, mục tiêu là tính tổn thất trên dữ liệu mới, tức là *tổn thất thực*. Dựa trên một lý thuyết về học máy, siêu phẳng có lề cực đại là siêu phẳng cho *tổn thất thực* nhỏ nhất trên dữ liệu mới. Về mặt trực giác, siêu phẳng như vậy cũng là siêu phẳng "an toàn" nhất do nó nằm xa vùng các ví dụ dương và âm nên khả năng ví dụ mới rơi vào vùng ngược dấu cũng nhỏ hơn.

Xác định siêu phẳng với lề cực đại

Để xác định siêu phẳng với lề cực đại, ta cần tìm **w** và *b* tương ứng. Quá trình đó được thực hiện như sau. Trước hết, có thể yêu cầu hàm phân loại cho phép phân chia toàn bộ ví dụ huấn luyện với một khoảng cách an toàn, chẳng hạn phải thoả mãn:

$$\mathbf{w}^T\mathbf{x}_i + b \geq +1 \text{ nếu } y_i = +1$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 \text{ nếu } y_i = -1$$

hay viết gọn lại thành $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0$ với mọi *i*. Điều này có thể thực hiện được, chẳng hạn bằng cách nhân **w** và *b* với hệ số phù hợp.

Các điểm $\mathbf{x}_i$ sao cho $\mathbf{w}^T\mathbf{x}_i + b = +1$ nằm trên siêu phẳng song song với siêu phẳng phân cách và cách gốc toạ độ một khoảng |1-b|/||**w**||. Tương tự, các điểm $\mathbf{x}_i$ sao cho $\mathbf{w}^T\mathbf{x}_i + b = -1$ nằm trên siêu phẳng song song với siêu phẳng phân cách và cách gốc toạ độ một khoảng |-1-b|/||**w**||. Khi đó, $m_+ = m_- = 1/||\mathbf{w}||$, và giá trị của lề sẽ bằng $2/||\mathbf{w}||$. Để giá trị lề là cực đại, ta cần xác định **w** sao cho $2/||\mathbf{w}||$ là lớn nhất, hay $||\mathbf{w}||^2/2$ là nhỏ nhất (cực tiểu hoá $||\mathbf{w}||^2$ thuận lợi hơn do dễ tính đạo hàm hơn). Như vậy, siêu phẳng có lề cực đại được xác định bằng cách tìm **w** và *b* sao cho:

$$\frac{1}{2}\|\mathbf{w}\|^2 \text{ là nhỏ nhất,}$$

đồng thời thoả mãn ràng buộc: $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0$ với mọi *i*.

Phân loại với lề mềm cho dữ liệu không phân chia tuyến tính

Trong trường hợp dữ liệu không thể phân chia tuyến tính, chẳng hạn do một số ví dụ nhiễu, hay thậm chí trong trường hợp phân chia tuyến tính, vẫn có thể tìm được siêu phẳng với lề rộng nếu chấp nhận phân loại sai một số ví dụ. Việc chấp nhận một số trường hợp sai được thể hiện bằng cách thay các ràng buộc thành:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

trong đó $\xi_i \geq 0$ là các biến phụ, được thêm vào để cho phép các ví dụ nằm trong vùng của lề ($1 \geq \xi_i > 0$) hoặc thậm chí bị phân loại sai ($\xi_i \geq 1$). Tổng của các biến phụ này sẽ giới hạn mức độ sai khi phân loại, tổng càng lớn chứng tỏ càng sai nhiều, nếu tất cả biến phụ bằng 0 thì toàn bộ dữ liệu không lấn vào vùng lề và được phân loại đúng. Do vậy, ta có thể thêm tổng các biến phụ vào hàm mục tiêu như một thành phần phạt cho các trường hợp bị sai. Bài toán tìm cực trị ở trên sẽ trở thành:

Tìm **w** và *b* sao cho:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \text{ là nhỏ nhất,}$$

đồng thời thoả mãn ràng buộc:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, \quad \text{với mọi } i$$

Tham số $C > 0$ thể hiện sự ưu tiên giữa hai mục tiêu: 1) tìm được lề lớn; và 2) giảm số lỗi. *C* càng lớn thì mục tiêu thứ hai càng được ưu tiên và ngược lại. Mô hình phân loại như vậy gọi là phân loại với *lề mềm* (soft margin).

Bài toán tối ưu trên gồm yêu cầu cực tiểu hoá mục tiêu và các ràng buộc. Bài toán này có thể giải bằng phương pháp nhân tử Lagrange (chi tiết về việc áp dụng phương pháp này

không được trình bầy ở đây). Sử dụng phương pháp nhân tử Lagrange, ta đưa bài toán về dạng đối ngẫu như sau:

Tìm các tham số α_i sao cho hàm mục tiêu:

$$\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

đạt giá trị lớn nhất (cực đại hoá), đồng thời thoả mãn các ràng buộc:

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \qquad 0 \le \alpha_i \le C$$

Sau khi xác định được α_i , trọng số **w** được tính như sau:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i$$

để tính b, ta có thể chọn một giá trị i sao cho $\alpha_i > 0$ và tính b theo công thức sau:

$$b = y_i - \mathbf{w}^T \mathbf{x}_i = y_i - \sum_{j=1}^{n} y_j \alpha_j \mathbf{x}_j^T \mathbf{x}_i$$

SVM với vec tơ trọng số **w** và tham số b xác định như trên được gọi là *SVM tuyến tính*.

Cần lưu ý rằng mỗi ví dụ đều có giá trị tham số α_i tương ứng. Các ví dụ có giá trị $\alpha_i > 0$ nằm trên ranh giới của lề (hoặc bên trong lề nếu sử dụng lề mềm). Các ví dụ này tạo thành các **vec tơ tựa** (support vector) xác định giới hạn lề. Đây là các ví dụ quan trọng trong dữ liệu huấn luyện, cho phép xác định siêu phẳng phân cách của SVM và cũng nằm gần siêu phẳng này nhất. Các ví dụ còn lại nằm bên ngoài vùng lề và có thể bỏ đi mà không ảnh hưởng tới vị trí của lề và hàm phân loại. Nói cách khác, các vec tơ tựa chứa toàn bộ thông tin cần thiết từ dữ liệu huấn luyện để xây dựng hàm phân loại cho SVM.

Như vậy, mặc dù hàm phân loại được tạo thành từ các ví dụ huấn luyện, nhưng chỉ những ví dụ tương ứng với $\alpha_i > 0$ mới đóng góp vào hàm phân loại (trọng số **w** là tổ hợp của các ví dụ này), và số lượng ví dụ như vậy thường không lớn. Đây là một trong những đặc điểm giúp SVM có tính khái quát hoá và khả năng dự toán tốt cho dữ liệu mới.

5.6.2. Kỹ thuật hàm nhân và SVM tổng quát

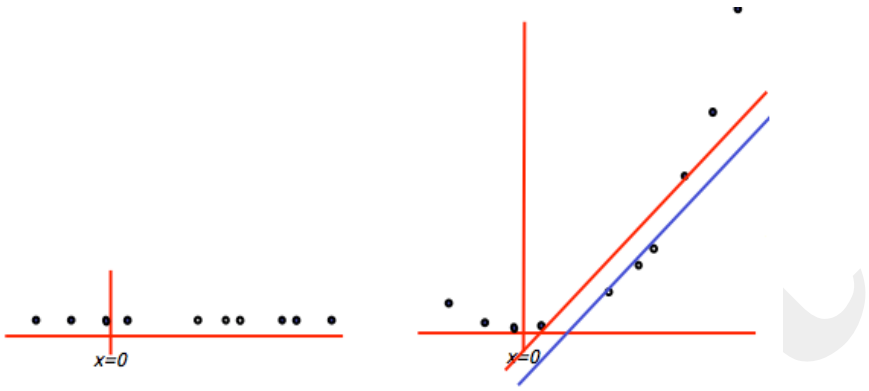
Bộ phân loại tuyến tính trình bầy ở trên cho kết quả tốt trong trường hợp dữ liệu có thể phân chi tuyến tính. Với dữ liệu không phân chia tuyến tính, việc sử dụng hàm phân loại phi tuyến sẽ phù hợp và cho kết quả tốt hơn. Trong phần này, ta sẽ xem xét cách mở rộng SVM tuyến tính cho trường hợp phi tuyến.

Việc sử dụng hàm phân loại tuyến tính cho trường hợp không tuyến tính có thể thực hiện bằng cách ánh xạ không gian dữ liệu ban đầu X sang không gian mới F bằng một hàm ánh xạ phi tuyến $\phi(\mathbf{x})$. Thông thường không gian F có số chiều lớn hơn số chiều trong không gian dữ liệu ban đầu. Trong không gian mới, hàm phân loại trở thành:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

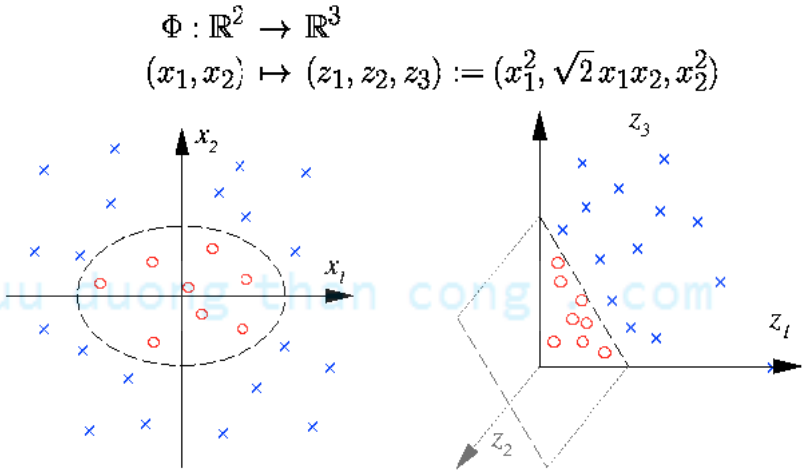
Ảnh hưởng của việc ánh xạ sang không gian mới được minh hoạ qua một số ví dụ sau.

Ví dụ 1. Để minh hoạ cho việc ánh xạ sang không gian mới, trước tiên ta xét một ví dụ đơn giản như trên hình 5.20. Dữ liệu ban đầu (bên trái) thuộc không gian 1 chiều, tức là mỗi ví dụ được biểu diễn bởi 1 thuộc tính (x). Các chấm đen tương ứng ví dụ dương, chấm trắng là ví dụ âm. Rõ ràng không thể tìm được một điểm cho phép phân chia các ví dụ khác dấu. Hình bên phải thể hiện việc ánh xạ sang không gian mới với 2 chiều (z_1, z_2) với ($z_1 = x$, $z_2 = x^2$). Có thể thấy trong không gian mới có thể tìm được đường thẳng cho phép phân cách ví dụ dương khỏi ví dụ âm.



Hình 5.20. Ví dụ ánh xạ từ không gian một chiều sang không gian hai chiều sử dụng ánh xạ $(x) \mapsto (z_1, z_2) = (x, x^2)$. Trong không gian hai chiều, dữ liệu có thể phân chia tuyến tính.

Ví dụ 2. Trên hình 5.21 là ví dụ khác. Từ không gian 2 chiều ban đầu (x_1, x_2), bằng ánh xạ $(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ ta được không gian mới ba chiều, trong đó các ví dụ dương (dấu nhân) được phân chia tuyến tính khỏi ví dụ âm (chấm trắng) bằng một mặt phẳng. Vec tơ đặc trưng trong không gian mới bao gồm tất cả các tích bậc hai của các đặc trưng trong không gian gốc.



Hình 5.21. Ví dụ một ánh xạ từ không gian hai chiều sang không gian ba chiều. Trong không gian mới, các ví dụ có thể phân chia tuyến tính bởi một mặt phẳng.

Ánh xạ bằng cách dùng hàm nhân (kernel)

Các ví dụ trên cho thấy, việc ánh xạ sang không gian mới (nhiều chiều hơn) làm cho dữ

liệu trở thành phân chia tuyến tính. Khó khăn ở chỗ, việc thực hiện các ánh xạ như vậy đòi hỏi tính toán các đặc trưng mới. Số lượng đặc trưng như vậy có thể rất lớn. Chẳng hạn, với phép ánh xạ trong ví dụ 2 ở trên, số đặc trưng mới xấp xỉ bình phương số đặc trưng cũ. Một số phép ánh xạ khác dẫn tới số đặc trưng lớn hơn nữa, thậm chí là vô cùng. Việc tính số lượng đặc trưng mới nhiều như vậy là không thực tế.

Để tránh việc tính toán các đặc trưng trong không gian mới, SVM sử dụng các **hàm nhân** (kernel function). Cụ thể, do trong không gian mới, vec tơ trọng số của SVM được tính bởi

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \phi(\mathbf{x}_i)$$

nên hàm phân loại sẽ trở thành

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i=1}^{n} y_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

Hàm mục tiêu cần cực đại hoá cũng trở thành

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i)$$

Do các biểu thức trên chứa $\phi(\mathbf{x})^T \phi(\mathbf{x}')$ nên nếu ta có thể tìm được hàm $K(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$, ta có thể sử dụng $K(\mathbf{x},\mathbf{x}')$ trong các biểu thức trên và không cần tính cụ thể các ánh xạ $\phi(\mathbf{x})$, $\phi(\mathbf{x}')$. Hàm $K(\mathbf{x},\mathbf{x}')$ như vậy được gọi là *hàm nhân* (kernel function) và cách sử dụng hàm nhân như vậy được gọi là kỹ thuật nhân (kernel trick).

Ví dụ. Quay lại ví dụ 2 ở trên với ánh xạ $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. Dễ dàng chứng minh

$$K(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2$$

như vậy, thay vì tính $\phi(\mathbf{x})$, ta sẽ dùng $K(\mathbf{x},\mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2$.

Với kỹ thuật sử dụng hàm nhân, ta được SVM tổng quát có dạng:

Hàm phân loại:

$$f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

Trong đó các tham số được xác định bằng cách giải bài toán tối ưu sau:

Cực đại hoá hàm mục tiêu:

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

đồng thời thoả mãn các ràng buộc:

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \qquad 0 \le \alpha_i \le C$$

sau đó tính b theo công thức sau:

$$b = y_i - \mathbf{w}^T\mathbf{x}_i = y_i - \sum_{j=1}^{n} y_j \alpha_j \mathbf{x}_j^T \mathbf{x}_i$$

Một số dạng hàm nhân thông dụng

Với mỗi bài toán phù hợp với một dạng hàm nhân cụ thể. Sau đây là một số dạng hàm nhân thường được sử dụng với SVM:

Hàm nhân đa thức: $K(\mathbf{x},\mathbf{x}') = (\mathbf{x}^T\mathbf{x}'+1)^d$

Trong trường hợp $d = 1$, hàm nhân gọi là hàm tuyến tính (không ánh xạ gì cả) và SVM trở thành SVM tuyến tính. Trong trường hợp nói chung, d là bậc của hàm nhân đa thức.

Hàm nhân Gauss: $K(\mathbf{x},\mathbf{x}') = \exp(-\gamma \| \mathbf{x} - \mathbf{x}' \|^2)$

trong đó ||**x**|| là độ dài của vec tơ **x** : $\| \mathbf{x} \| = \sqrt{\mathbf{x}^T\mathbf{x}}$; γ là tham số thể hiện độ rộng của nhân.

Hàm nhân sigmoid: $K(\mathbf{x},\mathbf{x}') = \tanh(\kappa \mathbf{x}^T\mathbf{x} + \theta)$

Trong trường hợp nói chung, nếu không có thông tin gì về bài toán thì nên thử hàm nhân Gauss. Đây là dạng hàm nhân thông dụng nhất.

5.6.3. Sử dụng trên thực tế

Huấn luyện SVM

Huấn luyện SVM là quá trình xác định các tham số α bằng cách giải bài toán tối ưu đã trình bầy ở trên. Đây là bài toán tối ưu với hàm mục tiêu bậc hai và có thể giải bằng cách phương pháp số. Trên thực tế, một số phương pháp đã được thiết kế để giải bài toán tối ưu này, tất cả đều dựa trên việc chia nhỏ bài toán tối ưu thành các bài toán con. Phương pháp thông dụng nhất là Sequential Minimal Optimization (SMO). Phương pháp này lần lượt cố định các giá trị α và chỉ cho phép hai giá trị thay đổi sau đó tìm giá trị tối ưu của hai tham số. Quá trình này được thực hiện với từng đôi α cho đến khi đạt tới giá trị tối ưu. Chi tiết về thuật toán huấn luyện SMO không được trình bầy ở đây.

Phân loại đa lớp

Trong trường hợp phân loại với nhiều nhãn, cách thường dùng là kết hợp nhiều SVM nhị phân với nhau. Có hai cách kết hợp các bộ phân loại nhị phân như vậy. Cách thứ nhất được gọi là kết hợp từng đôi (Pairwise). Cách thứ hai được gọi là một chọi tất cả (one-vs-all). Chi tiết về hai cách này hoàn toàn giống như trong phần phân loại đa lớp với hồi quy logistic đã trình bầy ở trên.

Lựa chọn tham số

Tham số đầu tiên cần quan tâm khi sử dụng SVM là dạng hàm nhân và tham số *C*. Với mỗi dạng hàm nhân lại có các tham số riêng của mình như bậc d của hàm nhân đa thức, độ rộng của hàm nhân Gaus hay tham số ngưỡng của nhân sigmoid.

Việc lựa chọn hàm nhân có thể thực hiện nếu biết các đặc điểm của bài toán. Trong trường hợp không có thông tin như vậy, cần thử nghiệm các dạng nhân và tham số khác nhau bằng cách sử dụng thủ tục kiểm tra chéo (cross-validation) sẽ được trình bầy trong phần 5.7.2.

Các ưu điểm của SVM

SVM có nhiều ưu điểm:

a. Đây là phương pháp có cơ sở lý thuyết tốt dựa trên lý thuyết về khả năng khái quát hóa của bộ phân loại.

b. Có thể làm việc tốt với dữ liệu nhiều chiều (nhiều thuộc tính).
c. Cho kết quả rất chính xác so với các phương pháp khác trong hầu hết ứng dụng.
d. Hiện nay có nhiều thư viện và phần mềm cho phép sử dụng SVM rất thuận tiện và kết hợp vào các chương trình viết trên hầu hết các ngôn ngữ lập trình thông dụng.

SVM là một trong những phương pháp phân loại được dùng phổ biến nhất hiện nay do có độ chính xác cao, nhiều phần mềm và thư viện có hỗ trợ.

5.7. ĐÁNH GIÁ VÀ LỰA CHỌN MÔ HÌNH

5.7.1.Các độ đo sử dụng trong đánh giá

Để đánh giá hiệu quả của mô hình, ta cần có các tiêu chí hay các độ đo. Có nhiều độ đo khác nhau có thể sử dụng, tùy vào ứng dụng cụ thể của thuật toán phân loại hoặc hồi quy trong từng trường hợp. Phần này sẽ giới thiệu một số độ đo thông dụng nhất.

Độ đo dùng trong phân loại.

Trước tiên, xét trường hợp phân loại hai lớp, trong đó mỗi ví dụ có thể nhận nhãn *dương* hoặc *âm*. Với mỗi trường hợp ví dụ mà mô hình dự đoán nhãn, có bốn khả năng xảy ra như liệt kê trên bảng sau, trong đó nhãn thật là nhãn của ví dụ và nhãn dự đoán là do mô hình tính toán ra:

	Nhãn dự đoán		
Nhãn thật	Dương	Âm	Tổng số
Dương	tp: dương đúng	fn: âm sai	p
Âm	fp: dương sai	tn: âm đúng	n
Tổng số	p'	n'	N

Theo bảng trên, nếu một ví dụ loại dương được mô hình dự đoán là dương thì được gọi là *dương đúng* (true positive: tp), nếu được dự đoán là âm thì gọi là *âm sai* (false negative: fn). Một ví dụ loại âm nếu được mô hình dự đoán là dương thì gọi là *dương sai* (false positive: fp), nếu được dự đoán là âm thì gọi là *âm đúng* (true negative: tn).

Sử dụng các khái niệm tp, tn, fp, fn như trên, có thể định nghĩa một số độ đo hiệu quả phân loại như sau (lưu ý: ta sẽ sử dụng tp, tn, fp, fn để ký hiệu số ví dụng dương đúng, âm đúng, dương sai, âm sai, N là tổng số ví dụ):

tỷ lệ lỗi: error = (fp + fn)/N

độ chính xác accuracy: accuracy = (fp + fn)/N = 1 - error

tỷ lệ dương đúng: tp-rate = tp/p

tỷ lệ dương sai: fp-rate = fp/p

độ chính xác precision: precision = tp/p’

độ thu hồi:	recall = tp/p = tp-rate
độ đo F	F-measure = (precision + recall)/2
độ nhậy:	sensitivity = tp/p = tp-rate
độ cụ thể:	specificity = tn/n = 1-fp-rate

Các độ đo nói trên đều có giá trị nằm trong khoảng [0, 1].

Trong các độ đo nói trên, các độ đo accuracy, recall và precision thường được sử dụng nhất. Độ đo accuracy được dùng khi ta chỉ quan tâm tới độ chính xác nói chung. Độ đo precision và recall được dùng khi ta quan tâm tới hiệu suất phân loại cho một lớp cụ thể. Ví dụ, khi phân loại email thành "thư rác" và "thư bình thường", ta cần quan tâm tới tỷ lệ thư rác phát hiện được và tức là độ đo recall, và tỷ lệ thư rác phát hiện đúng trong số thư rác được dự đoán, tức là độ đo precision. Cần chú ý rằng, recall tăng thì precision thường giảm và ngược lại. Ví dụ, trong trường hợp lọc thư rác, ta có thể dự đoán tất cả thư là thư rác, khi đó recall đạt giá trị cực đại bằng 1. Tuy nhiên, khi đó, giá trị p' cũng tăng lên và do vậy precision sẽ giảm đi.

Trong trường hợp phân loại nhiều lớp, các độ đo recall, precision, sensitivity, và specificity cho mỗi lớp được tính bằng cách coi đó là lớp dương và tất cả các lớp còn lại được gộp chung thành lớp âm.

Độ do dùng trong hồi quy

Trong trường hợp hồi quy, hai độ đo thường dùng là *lỗi trung bình tuyệt đối MAE* (Mean absolute error), và *lỗi trung bình bình phương* MSE (Mean squared error).

Giả sử ta có N ví dụ huấn luyện với giá trị đầu ra là $y_1, y_2, \ldots, y_N$. Tiếp theo, giả sử với các ví dụ này, mô hình hồi quy đưa ra dự đoán giá trị đầu ra lần lượt là $f_1, f_2, \ldots, f_N$. Hai độ đo trên khi đó được định nghĩa như sau:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

Độ đo MAE tính giá trị lỗi trong mọi trường hợp như nhau, trong khi độ đo MSE nhấn mạnh vào những trường hợp có lỗi lớn. Cụ thể, các chênh lệch có trị tuyệt đối lớn hơn 1 sẽ càng lớn do được bình phương. Trong khi đó, các chênh lệch với trị tuyệt đối nhỏ hơn 1 khi được bình phương sẽ giảm đi so với giá trị ban đầu.

5.7.2. Đánh giá mô hình bằng kiểm tra chéo

Khi sử dụng các thuật toán phân loại và hồi quy, một công đoạn quan trọng là đánh giá độ chính xác của mô hình. Việc đánh giá mô hình là cần thiết do hai lý do. Thứ nhất, cần biết mô hình được xây dựng và huấn luyện có độ chính xác ra sao, có đáp ứng yêu cầu của bài toán đặt ra không, trước khi sử dụng mô hình để giải quyết bài toán. Thứ hai, thông thường ta có thể có nhiều mô hình phân loại hoặc hồi quy và cần lựa chọn mô hình tốt nhất trong số đó cho bài toán cần giải quyết. Chẳng hạn, cần lựa chọn giữa cây quyết định, k-NN, hay hồi quy logistic. Hoặc với một loại mô hình, cần lựa chọn tham số cho độ chính xác cao nhất. Ví dụ,

với k-NN, cần chọn giá trị k (số láng giềng) đem lại độ chính xác cao nhất cho mỗi bài toán và bộ dữ liệu.

Một trong những cách đánh giá mô hình là huấn luyện mô hình trên toàn bộ tập dữ liệu huấn luyện, sau đó thử nghiệm độ chính xác trên cùng tập dữ liệu đó bằng cách dùng mô hình đã huấn luyện để dự đoán giá trị đầu ra cho các ví dụ huấn luyện và so sánh giá trị dự đoán với giá trị thực của đầu ra cho các ví dụ. Tuy nhiên, cách đánh giá này là không hợp lý và không nên sử dụng. Lý do là cách đánh giá này không cho kết quả khách quan nếu mô hình bị quá vừa dữ liệu, tức là cho độ chính xác cao trên dữ liệu huấn luyện nhưng lại cho kết quả kém chính xác trên dữ liệu mới.

Thay vì đánh giá mô hình trên cùng bộ dữ liệu đã dùng huấn luyện mô hình, cách đánh giá khách quan hơn là *kiểm tra chéo* (cross-validation). Có một số biến thể khác nhau của thủ tục kiểm tra chéo, sẽ được trình bầy dưới đây.

Kiểm tra chéo với tập kiểm tra tách riêng (hold-out cross validation), hay có thể gọi là kiểm tra chéo đơn giản, là phương pháp kiểm tra chéo đơn giản nhất và được thực hiện như sau:

e. Chia tập dữ liệu huấn luyện S ban đầu một cách ngẫu nhiên thành hai tập con: tập thứ nhất S_{hl} được gọi là tập huấn luyện, và tập thứ hai (phần còn lại) S_{kt} gọi là tập kiểm tra. Thông thường, S_{hl} gồm 70% tập dữ liệu ban đầu và S_{kt} gồm 30% còn lại.

f. Huấn luyện mô hình cần đánh giá h_i trên tập S_{hl}.

g. Đánh giá độ chính xác của mô hình h_i trên tập kiểm tra S_{kt}.

h. Chọn mô hình có độ chính xác cao nhất trên tập kiểm tra để sử dụng (nếu mục đích là lựa chọn mô hình).

Do mô hình được đánh giá trên tập dữ liệu kiểm tra S_{kt} , là dữ liệu chưa được dùng khi huấn luyện nên độ chính xác của mô hình trong trường hợp dữ liệu mới nói chung được ước lượng chính xác hơn. Thông thường, tập dữ liệu huấn luyện có kích thước bằng 1/4 đến 1/3 tập dữ liệu ban đầu, hay khoảng 30% tập dữ liệu ban đầu.

Trong trường hợp lựa chọn mô hình, mô hình tốt nhất được chọn sau đó được huấn luyện lại trên toàn bộ tập dữ liệu ban đầu. Nói chung, việc huấn luyện lại mô hình trên tập dữ liệu lớn hơn như vậy cho phép tăng độ chính xác.

Một nhược điểm của phương pháp sử dụng bộ dữ liệu kiểm tra riêng là phần dùng để huấn luyện (tập S_{hl}) chỉ còn khoảng 70% tập ban đầu và do vậy bỏ phí quá nhiều dữ liệu dùng để kiểm tra. Dưới đây là phương pháp kiểm tra chéo khác cho phép sử dụng ít dữ liệu kiểm tra hơn.

Kiểm tra chéo k-fold (k-fold cross validation). Đây là một biến thể tốt hơn so với kiểm tra chéo đơn giản, và được thực hiện như sau:

1. Chia ngẫu nhiên tập dữ liệu ban đầu S thành k tập dữ liệu có kích thước (gần) bằng nhau S_1, S_2,…, S_k.
2. Lặp lại thủ tục sau k lần với $i = 1,..,k$:
   i. Dùng tập Si làm tập kiểm tra. Gộp k-1 tập còn lại thành tập huấn luyện.
   j. Huấn luyện mô hình cần đánh giá trên tập huấn luyện.

k. Đánh giá độ chính xác của mô hình trên tập kiểm tra.

3. Độ chính xác của mô hình được tính bằng trung bình cộng độ chính xác trên k lần kiểm tra ở bước trên.
4. Chọn mô hình có độ chính xác trung bình lớn nhất.

Ưu điểm chính của kiểm tra chéo k-fold là nhiều dữ liệu hơn được sử dụng cho huấn luyện. Mỗi ví dụ được sử dụng để kiểm tra đúng 1 lần, trong khi được sử dụng trong tập huấn luyện *k* - 1 lần. Nhược điểm của phương pháp này là cần huấn luyện và đánh giá mô hình *k* lần, do vậy đòi hỏi nhiều thời gian.

Thông thường, phương pháp này được sử dụng với *k* = 10. Giá trị này vừa cho kết quả đánh giá khách quan vừa không đòi hỏi huấn luyện mô hình quá nhiều.

Kiểm tra chéo với một ví dụ kiểm tra (Leave one out cross validation, hay LOOCV). Đây là trường hợp riêng của kiểm tra chéo k-fold, trong đó giá trị của *k* bằng số ví dụ trong tập dữ liệu ban đầu. Như vậy, trong mỗi lần kiểm tra, chỉ một ví dụ được dành ra làm ví dụ kiểm tra, trong khi tất cả các ví dụ còn lại được dùng để huấn luyện. Độ chính xác sau đó cũng được tính trung bình trên toàn bộ các ví dụ. Phương pháp này được dùng trong trường hợp bộ dữ liệu ban đầu có kích thước nhỏ và do vậy cần dành nhiều ví dụ cho tập huấn luyện.

5.7.3. Lựa chọn đặc trưng

Lựa chọn đặc trưng (feature selection) là một trong những kỹ thuật quan trọng khi xây dựng các hệ thống học máy. Việc lựa chọn được các đặc trưng hợp lý để biểu diễn các ví dụ cho phép tạo ra các mô hình học máy có độ chính xác cao. Như đã trình bầy ở đầu chương, để phân loại hoặc hồi quy, mỗi ví dụ hay đối tượng cần biểu diễn dưới dạng vec tơ các đặc trưng. Với mỗi bài toán cụ thể, ta có thể chọn các đặc trưng khác nhau. Ví dụ, trong bài toán phân loại email thành thư rác hoặc thư thường, có thể sử dụng đặc trưng là các từ chứa trong thư, địa chỉ nơi gửi, dòng tiêu đề, độ dài thư, địa chỉ người nhận. Trong đó, địa chỉ nơi gửi là thông tin quan trọng về việc đó là thư rác hay không, trong khi địa chỉ người nhận không liên quan tới phân loại này, và do vậy không nên sử dụng đặc trưng địa chỉ người nhận khi phân loại thư.

Tập đặc trưng phù hợp với bài toán phân loại phải thỏa mãn hai điều kiện sau:

l. Giá trị các đặc trưng có quan hệ với nhãn phân loại và do vậy việc sử dụng đặc trưng cho thêm thông tin về phân loại của các ví dụ.
m. Tập đặc trưng không dư thừa. Nếu giá trị một đặc trưng có thể suy ra từ giá trị một số đặc trưng khác thì đặc trưng đang xét bị coi là dư thừa do không cung cấp thêm thông tin cho mô hình phân loại. Việc bớt các thuộc tính dư thừa một mặt cho phép giảm thời gian tính toán, mặt khác cho phép tăng độ chính xác của mô hình trong một số trường hợp.

Khi lựa chọn đặc trưng, từ tập đặc trưng ban đầu bao gồm *n* đặc trưng ta cần lựa chọn *m* đặc trưng (m << n) thỏa mãn hai điều kiện trên.

Có hai cách tiếp cận chính cho lựa chọn đặc trưng: 1) lựa chọn bằng cách lọc (filtering), và 2) lựa chọn bằng phương pháp bao (wrapper).

Lọc đặc trưng (feature filtering).

Đây là dạng phương pháp heuristic, theo đó ta dùng một độ đo $S(x_i)$ thể hiện sự liên quan giữa từng đặc trưng x_i với nhãn phân loại, hay nói cách khác, đó là độ đo thông tin mà đặc trưng x_i cung cấp về nhãn phân loại của các ví dụ. Các đặc trưng sau đó được sắp xếp theo thứ tự giảm dần của độ đo này và m đặc trưng có độ đo liên quan lớn nhất sẽ được chọn.

Đối với các đặc trưng rời rạc, độ đo thường dùng nhất là độ đo *thông tin tương hỗ* MI (mutual information). Độ đo thông tin tương hỗ MI giữa đặc trưng x_i và nhãn phân loại y của đầu ra được định nghĩa như sau:

$$MI(x_i, y) = \sum_{x_i \in values(x_i)} \sum_{y \in values(y)} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

trong đó $values(x_i)$ là tập các giá trị mà x_i có thể nhận, và $values(y)$ là tập các giá trị nhãn phân loại mà y có thể nhận. Các giá trị xác suất $p(x_i, y)$, $p(x_i)$, và $p(y)$ được tính từ tập dữ liệu huấn luyện. Độ đo MI cho biết lượng thông tin mà ta có được về y khi biết giá trị của x_i.

Giả sử ta đã tính được độ đo MI và sắp xếp được các đặc trưng theo thứ tự giảm dần của MI. Vậy cần chọn bao nhiêu đặc trưng trong số đó, tính từ trên xuống dưới? Để xác định số đặc trưng tốt nhất, ta có thể sử dụng phương pháp kiểm tra chéo đã trình bầy ở trên. Cụ thể, lần lượt chọn $m = 1, 2, \ldots$ đặc trưng có MI cao nhất, với mỗi giá trị của m dùng kiểm tra chéo để tính độ chính xác. Lặp lại quá trình này cho tới khi độ chính xác kiểm tra chéo không tăng mà bắt đầu giảm.

Chọn đặc trưng bằng mô hình bọc (wrapper feature selection).

Phương pháp này cho phép chọn tập đặc trưng tốt cho từng mô hình phân loại cụ thể. Giả sử ban đầu ta có n đặc trưng. Từ số đặc trưng này có thể sinh ra tất cả 2^n tập đặc trưng con khác nhau. Ta có thể lần lượt thử độ chính xác của các tập đặc trưng con này (bằng kiểm tra chéo) với một thuật toán phân loại được chọn, sau đó giữ lại tập đặc trưng con cho độ chính xác cao nhất và dùng tập đặc trưng đó để xây dựng mô hình. Như vậy, theo phương pháp này, độ tốt của từng đặc trưng được tính xung quanh thuật toán phân loại cụ thể, do vậy phương pháp có tên là bọc (wrapper), theo nghĩa bọc quanh thuật toán phân loại.

Nếu n lớn, việc liệt kê và so sánh toàn bộ 2^n tập đặc trưng con đòi hỏi tính toán rất nhiều, và do vậy phương pháp vét cạn này không thể sử dụng được. Thay vào đó, có thể sử dụng tìm kiếm tham lam để tìm ra tập đặc trưng con tương đối tốt.

Có hai cách tìm kiếm tham lam được dùng để tìm tập đặc trưng con tốt: *tìm kiếm tiến* (forward search) và *tìm kiếm lùi* (backward search).

Thủ tục tìm kiếm tiến được thực hiện như sau:

Đầu vào: tập đặc trưng ban đầu gồm n đặc trưng $x_1, x_2, \ldots, x_n$
thuật toán phân loại A

Đầu ra: tập đặc trưng tốt F*

Thực hiện:

1. Khởi tạo tập đặc trưng F ← ∅
2. Lặp với (i = 1 to n -1)

i. Thêm một đặc trưng chưa thuộc F vào F để tạo thành tập đặc trưng F_i và đánh giá độ chính xác của A với tập đặc trưng F_i bằng kiểm tra chéo.

ii. Chọn tập đặc trưng F_i cho kết quả tốt nhất ở bước i.

iii. Gán $F \leftarrow F_i$

3. Trả về tập đặc trưng F cho độ chính xác cao nhất trong n bước lặp

Khởi đầu từ tập đặc trưng rỗng, thuật toán trên gồm n - 1 bước lặp chính, mỗi bước lặp bổ sung thêm một đặc trưng vào tập đặc trưng hiện thời. Cụ thể, tại mỗi bước lặp, thuật toán tìm một đặc trưng sao cho khi thêm một mình đặc trưng đó vào tập đặc trưng hiện thời, độ chính xác (kiểm tra chéo) đạt được là tốt nhất. Cuối cùng, thuật toán trả về tập đặc trưng cho độ chính xác cao nhất trong số n tập đặc trưng được tạo ra.

Ngược lại với tìm kiếm tiến, tìm kiếm lùi (backward search) khởi đầu với tập đặc trưng gồm toàn bộ n đặc trưng ban đầu, sau đó cũng lặp với n-1 bước. Tại mỗi bước, thuật toán tìm cách loại khỏi tập đặc trưng hiện thời một đặc trưng, sao cho sau khi loại một mình đặc trưng đó, độ chính xác của mô hình là lớn nhất. Cuối cùng, thuật toán trả về tập đặc trưng cho độ chính xác cao nhất trong số n tập đặc trưng được tạo ra.

So sánh phương pháp chọn đặc trưng lọc và bọc.

Phương pháp bọc cho kết quả chọn đặc trưng chính xác hơn và kết quả này là đặc thù cho thuật toán phân loại cụ thể. Tuy nhiên, phương pháp này tốn nhiều thời gian do cần xây dựng và thử nghiệm nhiều mô hình phân loại với $(O(n^2))$ tập đặc trưng con.

Trong khi đó, phương pháp lọc là một phương pháp dạng heuristic và tập đặc trưng được chọn không phụ thuộc và thuật toán phân loại cụ thể. Tuy nhiên, phương pháp này đòi hỏi tính toán ít hơn và do vậy có thể sử dụng trong trường hợp tập đặc trưng ban đầu lớn, ví dụ khi phân loại văn bản.

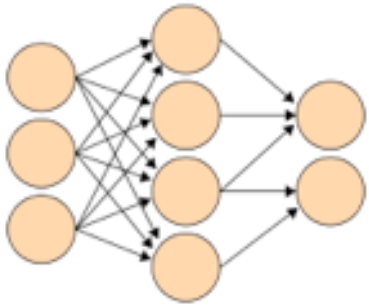
5.8. SƠ LƯỢC VỀ MỘT SỐ PHƯƠNG PHÁP HỌC MÁY KHÁC

Ngoài ba phương pháp học máy dùng cho phân loại được trình bày ở trên, rất nhiều phương pháp học máy khác đã được phát triển và sử dụng trong nhiều ứng dụng khác nhau. Sau đây là giới thiệu sơ lược về một số phương pháp được ứng dụng nhiều hoặc và độ chính xác tốt.

Mạng nơ ron nhân tạo

Mạng nơ ron nhân tạo dựa trên nguyên lý hệ thống thần kinh (nơ ron) của người và động vật bậc cao, trong đó sử dụng nhiều nút được nối với nhau thành một mạng lưới. Tín hiệu được truyền từ nút nọ sang nút kia tùy thuộc mức tín hiệu và cơ chế xử lý tại mỗi nút.

Trên hình sau là một ví dụ tổ chức mạng nơ ron trong đó một số nút nhận tín hiệu từ ngoài vào, một số nút trả tín hiệu ra ngoài, các nút có thể truyền tín hiệu tới những nút khác.



Hình 5.22. Ví dụ mạng nơ ron nhân tạo

Tại mỗi nút có một hàm cho phép trả ra kết quả theo tín hiệu đầu vào. Nhờ việc sử dụng nhiều nút với những hàm xử lý riêng tại từng nút, mạng nơ ron cho phép xây dựng những hàm đích rất phức tạp để ánh xạ từ tín hiệu đầu vào thành kết quả đầu ra.

Quá trình học mạng nơ ron là quá trình hiệu chỉnh tham số của hàm xử lý tại các nút sao cho ánh xạ đầu vào sang đầu ra phù hợp nhất với dữ liệu huấn luyện.

Mạng nơ ron nhân tạo là phương pháp học máy được phát triển từ sớm, có thể dùng cho phân loại hoặc hồi quy với độ chính xác cao, đã và đang được sử dụng trong rất nhiều ứng dụng khác nhau.

Boosting

Khác với những phương pháp trình bày ở trên, boosting là một dạng meta learning, tức là làm việc dựa trên những phương pháp học khác. Để giải quyết bài toán phân loại, boosting kết hợp nhiều bộ phân loại đơn giản với nhau để tạo ra một bộ phân loại có độ chính xác cao hơn.

Ví dụ, ta có thể xây dựng bộ phân loại đơn giản bằng cách sử dụng cây quyết định chỉ có một nút – nút gốc, còn được gọi là gốc cây quyết định. Cây quyết định như vậy sẽ có độ chính xác không cao. Thuật toán boosting khi đó kết hợp các gốc cây như sau:

1. Mỗi ví dụ huấn luyện được gán một trọng số, đầu tiên trọng số bằng nhau.
2. Thuật toán lặp lại nhiều vòng
   I. Tại mỗi vòng, lựa chọn một gốc cây có độ chính xác tốt nhất. Gốc cây chính xác nhất là gốc cây có lỗi nhỏ nhất với lỗi tính bằng tổng trọng số những ví dụ bị phân loại sai.
   II. Những ví dụ bị phân loại sai được tăng trọng số trong khi những ví dụ đúng bị giảm trọng số. Nhờ việc thay đổi trọng số như vậy, thuật toán sẽ chú ý nhiều hơn tới ví dụ bị phân loại sai trong những vòng sau.
3. Bộ phân loại cuối được tạo ra bằng tổng các cây quyết định xây dựng tại mỗi vòng lặp.

Thuật toán boosting có một số ưu điểm như:

- Độ chính xác cao.
- Ít bị ảnh hưởng bởi hiện tượng quá vừa dữ liệu.
- Có thể sử dụng với nhiều phương pháp phân loại đơn giản khác nhau.

5.9. CÂU HỎI VÀ BÀI TẬP CHƯƠNG

1. Cho dữ liệu huấn luyện như trong bảng (f là nhãn phân loại).

X	Y	Z	f
1	0	1	1
1	1	0	0
0	0	0	0
0	1	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	0

   a) Hãy xây dựng cây quyết định sử dụng thuật toán ID3. Trong trường hợp có hai thuộc tính tốt tương đương thì chọn theo thứ tự bảng chữ cái.

   b) Giả sử không biết nhãn phân loại của ví dụ cuối cùng, hãy xác định nhãn cho ví dụ đó bằng phương pháp Bayes đơn giản (chỉ rõ các xác suất điều kiện thành phần) và k láng giềng gần nhất với k = 5.

2. Cho dữ liệu huấn luyện dưới đây với 16 ví dụ.

A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1
C	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	1

Sử dụng phân loại Bayes để tính giá trị của C nếu biết A = 0, B = 1. Yêu cầu viết chi tiết các bước.

3. Cho dữ liệu huấn luyện như trong bảng (f là nhãn phân loại).

   a) Hãy xác định nhãn cho ví dụ (Màu: Trắng, Hình dạng: Tròn, KL: Nặng) bằng phương pháp Bayes đơn giản (chỉ rõ các xác suất điều kiện thành phần)

   b) Hãy xác định nút gốc cho cây quyết định sử dụng thuật toán ID3

Màu	**Hình dạng**	**KL**	**f**
Xanh	Tròn	Nặng	+
Đỏ	Tròn	Nhẹ	-
Xanh	Méo	Nhẹ	+
Trắng	Méo	Nặng	+
Đỏ	Méo	Nặng	-
Trắng	Tròn	Nhẹ	-

Trắng	Méo	Nhẹ	+

4. Cho dữ liệu huấn luyện như trong bảng sau, trong đó mỗi cột (trừ cột ngoài cùng bên trái) ứng với một mẫu, dòng dưới cùng (T) chứa giá trị đích:

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	1	1	0	1	0	1	1
3	1	1	1	0	1	0	0	1	1	0	0	0	1	1
4	0	1	0	0	1	0	0	1	0	1	1	1	0	1
5	0	0	1	1	0	1	1	0	1	1	0	0	1	0
6	0	0	0	1	0	1	0	1	1	0	1	1	1	0
T	1	1	1	1	1	1	0	1	0	0	0	0	0	0

a) Hãy thực hiện thuật toán giảm gradient cho hồi quy tuyến tính với dữ liệu trên, ghi lại giá trị trọng số sau mỗi bước.

b) Xây dựng cây quyết định cho dữ liệu trong bảng.

5. Hãy vẽ cây quyết định để biểu diễn các biểu thức logic sau:

a) $A \wedge \neg B$

b) $A \vee (B \wedge C)$

c) $A \Leftrightarrow B$

d) $(A \wedge B) \wedge (C \wedge D)$

TÀI LIỆU THAM KHẢO

1. S. Russell, P. Norvig. Artificial intelligence: a modern approach. 3nd edition. Prentice Hall. 2010.
2. T. M. Jones. Artificial intelligence a system approach. Infinity science press. 2008.
3. Đinh Mạnh Tường. Trí tuệ nhân tạo. Nhà xuất bản Khoa học kỹ thuật. 2002.
4. Nguyễn Thanh Thủy. Trí tuệ nhân tạo. Nhà xuất bản khoa học kỹ thuật 1999.
5. T. Mitchell. Machine learning. McGrawhill. 1997.
6. D. Koller, N. Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press. 2009.
7. Đỗ Trung Tuấn. Nhập môn trí tuệ nhân tạo. Nhà xuất bản Đại học quốc gia Hà Nội. 2010.
8. Từ Minh Phương. Bài giảng trí tuệ nhân tạo. Học viện Công nghệ bưu chính viễn thông. 2010.