Giáo trình

TRÍ TUỆ NHÂN TẠO

ARTIFICIAL INTELLIGENCE

Phạm Thọ Hoàn, Phạm Thị Anh Lê

Khoa Công nghệ thông tin

Trường Đại học Sư phạm Hà Nội

Hà nội, 2011

MỤC LỤC

Chương 1 – Giới thiệu

1. Trí tuệ nhân tạo là gì?

Để hiểu trí tuệ nhân tạo (artificial intelligence) là gì chúng ta bắt đầu với khái niệm sự bay nhân tạo (flying machines), tức là cái máy bay.

Đã từ lâu, loài người mong muốn làm ra một cái máy mà có thể di chuyển được trên không trung mà không phụ thuộc vào địa hình ở dưới mặt đất, hay nói cách khác là máy có thể bay được. Không có gì ngạc nhiên khi những ý tưởng đầu tiên làm máy bay là từ nghiên cứu cách con chim bay. Những chiếc máy biết bay được thiết kế theo nguyên lý "vỗ cánh" như con chim chỉ có thể bay được quãng đường rất ngắn và lịch sử hàng không thực sự sang một trang mới kể từ anh em nhà Wright thiết kế máy bay dựa trên các nguyên lý của khí động lực học (aerodynamics).

Các máy bay hiện nay, như đã thấy, có sức trở rất lớn và bay được quãng đường có thể vòng quanh thế giới. Nó không nhất thiết phải có nguyên lý bay của con chim nhưng vẫn bay được như chim (dáng vẻ), và còn tốt hơn chim.

Quay lại câu hỏi Trí tuệ nhân tạo là gì. Trí tuệ nhân tạo là trí thông minh của máy do con người tạo ra. Ngay từ khi chiếc máy tính điện tử đầu tiên ra đời, các nhà khoa học máy tính đã hướng đến phát hiển hệ thống máy tính (gồm cả phần cứng và phần mềm) sao cho nó có khả năng thông minh như loài người. Mặc dù cho đến nay, theo quan niệm của người viết, ước mơ này vẫn còn xa mới thành hiện thực, tuy vậy những thành tựu đạt được cũng không hề nhỏ: chúng ta đã làm được các hệ thống (phần mềm chơi cờ vua chạy trên siêu máy tinh GeneBlue) có thể thắng được vua cờ thế giới; chúng ta đã làm được các phần mềm có thể chứng minh được các bài toán hình học; v.v. Hay nói cách khác, trong một số lĩnh vực, máy tính có thể thực hiện tốt hơn hoặc tương đương con người (tất nhiên không phải tất cả các lĩnh vực). Đó chính là các hệ thống thông minh.

Có nhiều cách tiếp cận để làm ra trí thông minh của máy (hay là trí tuệ nhân tạo), chẳng hạn là nghiên cứu cách bộ não người sản sinh ra trí thông minh của loài người như

thế nào rồi ta bắt chước nguyên lý đó, nhưng cũng có những cách khác sử dụng nguyên lý hoàn toàn khác với cách sản sinh ra trí thông minh của loài người mà vẫn làm ra cái máy thông minh như hoặc hơn người; cũng giống như máy bay hiện nay bay tốt hơn con chim do nó có cơ chế bay không phải là giống như cơ chế bay của con chim.

Như vậy, trí tuệ nhân tạo ở đây là nói đến khả năng của máy khi thực hiện các công việc mà con người thường phải xử lý; và khi dáng vẻ ứng xử hoặc kết quả thực hiện của máy là tốt hơn hoặc tương đương với con người thì ta gọi đó là máy thông minh hay máy đó có trí thông minh. Hay nói cách khác, đánh giá sự thông minh của máy không phải dựa trên nguyên lý nó thực hiện nhiệm vụ đó có giống cách con người thực hiện hay không mà dựa trên kết quả hoặc dáng vẻ ứng xử bên ngoài của nó có giống với kết quả hoặc dáng vẻ ứng xử của con người hay không.

Các nhiệm vụ của con người thường xuyên phải thực hiện là: ***giải bài toán*** (tìm kiếm, chứng minh, lập luận), ***học***, ***giao tiếp***, ***thể hiện cảm xúc***, ***thích nghi với môi trường xung quanh***, v.v., và dựa trên kết quả thực hiện các nhiệm vụ đó để kết luận rằng một ai đó có là thông minh hay không. Môn học Trí tuệ nhân tạo nhằm cung cấp các phương pháp luận để làm ra hệ thống có khả năng thực hiện các nhiệm vụ đó: giải toán, học, giao tiếp, v.v. bất kể cách nó làm có như con người hay không mà là kết quả đạt được hoặc dáng vẻ bên ngoài như con người.

Trong môn học này, chúng ta sẽ tìm hiểu các phương pháp để làm cho máy tính biết cách giải bài toán, biết cách lập luận, biết cách học, v.v.

2. Lịch sử

Vào năm 1943, Warren McCulioch và Walter Pitts bắt đầu thực hiện nghiên cứu ba cơ sở lý thuyết cơ bản: triết học cơ bản và chức năng của các noron thần kinh; phân tích các mệnh đề logic; và lý thuyết dự đoán của Turing. Các tác giả đã nghiên cứu đề xuất mô hình noron nhân tạo, mỗi noron đặc trưng bởi hai trạng thái "bật", "tắt" và phát hiện mạng noron có khả năng học.

Thuật ngữ “Trí tuệ nhân tạo” (Artificial Intelligence - AI) được thiết lập bởi John McCarthy tại Hội thảo đầu tiên về chủ đề này vào mùa hè năm 1956. Đồng thời, ông cũng đề xuất ngôn ngữ lập trình Lisp – một trong những ngôn ngữ lập trình hàm tiêu biểu, được sử dụng trong lĩnh vực AI. Sau đó, Alan Turing đưa ra "Turing test" như là một phương pháp kiểm chứng hành vi thông minh.

Thập kỷ 60, 70 Joel Moses viết chương trình Macsyma - chương trình toán học sử dụng cơ sở tri thức đầu tiên thành công. Marvin Minsky và Seymour Papert đưa ra các chứng minh đầu tiên về giới hạn của các mạng nơ-ron đơn giản. Ngôn ngữ lập trình logic Prolog ra đời và được phát triển bởi Alain Colmerauer. Ted Shortliffe xây dựng thành công một số hệ chuyên gia đầu tiên trợ giúp chẩn đoán trong y học, các hệ thống này sử dụng ngôn ngữ luật để biểu diễn tri thức và suy diễn.

Vào đầu những năm 1980, những nghiên cứu thành công liên quan đến AI như các hệ chuyên gia (expert systems) – một dạng của chương trình AI mô phỏng tri thức và các kỹ năng phân tích của một hoặc nhiều chuyên gia con người

Vào những năm 1990 và đầu thế kỷ 21, AI đã đạt được những thành tựu to lớn nhất, AI được áp dụng trong logic, khai phá dữ liệu, chẩn đoán y học và nhiều lĩnh vực ứng dụng khác trong công nghiệp. Sự thành công dựa vào nhiều yếu tố: tăng khả năng tính toán của máy tính, tập trung giải quyết các bài toán con cụ thể, xây dựng các mối quan hệ giữa AI và các lĩnh vực khác giải quyết các bài toán tương tự, và một sự chuyển giao mới của các nhà nghiên cứu cho các phương pháp toán học vững chắc và chuẩn khoa học chính xác.

3. Các lĩnh vực ứng dụng của AI

- Bài toán lập luận, suy diễn

Khái niệm lập luận (reasoning), và suy diễn (reference) được sử dụng rất phổ biến trong lĩnh vực AI. Lập luận là suy diễn logic, dùng để chỉ một tiến trình rút ra kết luận (tri thức mới) từ những giả thiết đã cho (được biểu diễn dưới dạng cơ sở tri thức). Như vậy, để thực hiện lập luận người ta cần có các phương pháp lưu trữ cơ sở tri thức và các thủ tục lập luận trên cơ sở tri thức đó.

- Biểu diễn tri thức

Muốn máy tính có thể lưu trữ và xử lý tri thức thì cần có các phương pháp biểu diễn tri thức. Các phương pháp biểu diễn tri thức ở đây bao gồm các ngôn ngữ biểu diễn và các kỹ thuật xử lý tri thức. Một ngôn ngữ biểu diễn tri thức được đánh giá là "tốt" nếu nó có *tính biểu đạt* cao và các *tính hiệu quả* của thuật toán lập luận trên ngôn ngữ đó. Tính biểu đạt của ngôn ngữ thể hiện khả năng biểu diễn một phạm vi rộng lớn các thông tin trong một miền ứng dụng. Tính hiệu quả của các thuật toán lập luận thể hiện chi phí về thời gian và không gian dành cho việc lập luận. Tuy nhiên, hai yếu tố này dường như đối nghịch nhau, tức là nếu ngôn ngữ có tính biểu đạt cao thì thuật toán lập luận trên đó sẽ có độ phức tạp lớn (tính hiệu quả thấp) và ngược lại (ngôn ngữ đơn giản, có tính biểu đạt thấp thì thuật toán lập luận trên đó sẽ có hiệu quả cao). Do đó, một thách thức lớn trong lĩnh vực AI là xây dựng các ngôn ngữ biểu diễn tri thức mà có thể cân bằng hai yếu tố này, tức là ngôn ngữ có tính biểu đạt đủ tốt (tùy theo từng ứng dụng) và có thể lập luận hiệu quả.

- Lập kế hoạch: khả năng suy ra các mục đích cần đạt được đối với các nhiệm vụ đưa ra, và xác định dãy các hành động cần thực hiện để đạt được mục đích đó.
- Học máy: là một lĩnh vực nghiên cứu của AI đang được phát triển mạnh mẽ và có nhiều ứng dụng trong các lĩnh vực khác nhau như khai phá dữ liệu, khám phá tri thức,…
- Xử lý ngôn ngữ tự nhiên: là một nhánh của AI, tập trung vào các ứng dụng trên ngôn ngữ của con người. Các ứng dụng trong nhận dạng tiếng nói, nhận dạng chữ viết, dịch tự động, tìm kiếm thông tin,…
- Hệ chuyên gia: cung cấp các hệ thống có khả năng suy luận để đưa ra những kết luận. Các hệ chuyên gia có khả năng xử lý lượng thông tin lớn và cung cấp các kết luận dựa trên những thông tin đó. Có rất nhiều hệ chuyên gia nổi tiếng như các hệ chuyên gia y học MYCIN, đoán nhận cấu trúc phân tử từ công thức hóa học DENDRAL, …

- Robotics
- …

4. *Nội dung môn học*

Giáo trình này được viết với các nội dung nhập môn về AI cho các sinh viên chuyên ngành Tin học và Công nghệ thông tin. Các tác giả có tham khảo một số tài liệu, giáo trình của các trường Đại học Quốc gia Hà nội, Đại học Bách khoa Hà nội, … Nội dung gồm các phần sau:

- *Chương 1. Giới thiệu*: trình bày tổng quan về AI, lịch sử ra đời và phát triển và các lính vực ứng dụng của AI.
- *Chương 2. Các phương pháp tìm kiếm lời giải*: trình bày các kỹ thuật tìm kiếm cơ bản được áp dụng để giải quyết các vấn đề và được áp dụng rộng rãi trong các lĩnh vực của trí tuệ nhân tạo.
- *Chương 3. Các giải thuật tìm kiếm lời giải cho trò chơi*: trình bày một số kỹ thuật tìm kiếm trong các trò chơi có đối thủ.
- *Chương 4. Các phương pháp lập luận trên logic mệnh đề*: trình bày cú pháp, ngữ nghĩa của logic mệnh đề và một số thuật toán lập luận trên logic mệnh đề.
- *Chương 5. Các phương pháp lập luận trên logic vị từ cấp một*: trình bày cú pháp, ngữ nghĩa của logic vị từ cấp một và một số thuật toán lập luận cơ bản trên logic vị từ cấp một.
- *Chương 6. Prolog*: Giới thiệu chung về ngôn ngữ Prolog, cú pháp, ngữ nghĩa và cấu trúc chương trình trong Prolog, một số phiên bản mới của Prolog như SWI Prolog,…
- *Chương 7. Lập luận với tri thức không chắc chắn*: Giới thiệu về tri thức không chắc chắn và một số cách tiếp cận biểu diễn và xử lý tri thức không chắc chắn.
- *Chương 8. Học mạng noron nhân tạo*: Giới thiệu về phương pháp và các kỹ thuật cơ bản trong lập luận sử dụng mạng noron nhân tạo.

Chương 2 – Các phương pháp tìm kiếm lời giải

Một lớp các nhiệm vụ mà máy tính (với phần mềm trí tuệ nhân tạo phù hợp) có thể thực hiện tốt hơn con người nhờ vào tốc độ thực hiện của CPU và bộ nhớ dung lượng lớn là ***tìm kiếm lời giải*** trong không gian (hữu hạn hoặc vô hạn) các lời giải tiềm năng của một bài toán cụ thể. Một lời giải tiềm năng có thể là một đường đi trong không gian đồ thị trạng thái của bài toán hoặc là một trạng thái của bài toán. Với tốc độ xử lý nhanh, máy tính chỉ cần sinh ra và duyệt các lời giải tiềm năng (cây tìm kiếm) một cách có hệ thống (tìm kiếm mù, tìm kiếm theo chiều rộng hoặc theo chiều sâu) và kiểm tra nó có là lời giải thực sự (tối ưu) của bài toán đã cho không. Trong nhiều trường hợp, máy tính có thể sử dụng một hàm đánh giá/định hướng (hàm heuristic) để hạn chế không sinh ra các phần của cây tìm kiếm mà khả năng sẽ không chứa lời giải thực sự.

Rất nhiều bài toán từ đơn giản đến phức tạp có thể giải được bằng các phương pháp tìm kiếm đơn giản như trên, như các bài toán chơi cờ, các bài toán tìm đường đi trên đồ thị, các bài toán duyệt các tổ hợp, chỉnh hợp, v.v. Các bài toán có cùng đặc trưng là có thể biểu diễn bởi 4 thành tố sinh ra không gian các lời giải tiềm năng của bài toán: trạng thái đầu, trạng thái đích, các thao tác chuyển trạng thái, chi phí các phép chuyển trạng thái. Với các bài toán này, rõ ràng máy tính có khả năng giải quyết tốt hơn con người.

2.1 Hình thành bài toán

Trong thực tế, nhiều bài toán có thể đưa về bài toán tìm kiếm. Chẳng hạn, muốn tìm nghiệm của phương trình, ta đi tìm những giá trị của biến số trong miền xác định mà khi thay vào phương trình được thỏa mãn; với các bài toán chứng minh, ta đi tìm dãy các lập luận sao cho xuất phát từ giả thiết có thể đi đến kết luận; hoặc muốn đi từ Hà nội đến Sài gòn, người ta phải tra cứu trên bản đồ để tìm những thành phố mà theo đó có thể đi đến Sài gòn nếu xuất phát từ Hà nội.

Bài toán tìm kiếm là xác định trong không gian tìm kiếm (miền) những đối tượng mà thỏa mãn các điều kiện đặt ra.

Trong lĩnh vực AI, chúng ta nghiên cứu hành trình của các agent thông minh. Cơ sở của các bài toán là: trạng thái đầu (trạng thái xuất phát), các hành động biến đổi trạng thái và trạng thái kết thúc (trạng thái đích). Vấn đề đặt ra là xác định dãy các trạng thái hợp lý để sao cho từ trạng thái xuất phát có thể đến được trạng thái đích.

Không gian trạng thái: là tập các trạng thái có thể đạt được bằng cách thực hiện chuỗi các hành động xuất phát từ trạng thái ban đầu. Một hành trình không gian trạng thái là thực hiện dãy các hành động từ trạng thái này đến trạng thái khác.

Giải bài toán : xác định trạng thái xuất phát, tìm dãy các hành động hoặc phép biến đổi (toán tử) các trạng thái sao cho từ trạng thái xuất phát có thể dẫn đến trạng thái đích. Với mỗi bài toán, có thể có một hoặc nhiều cách giải, trong đó, người ta luôn mong muốn tìm lời giải "tốt nhất" dựa vào việc tình chi phí thực hiện.

Hàm chi phí: Giá trị đánh giá chi phí thực hiện biến đổi trạng thái.

Hiệu quả của việc tìm kiếm thể hiện qua việc đánh giá:

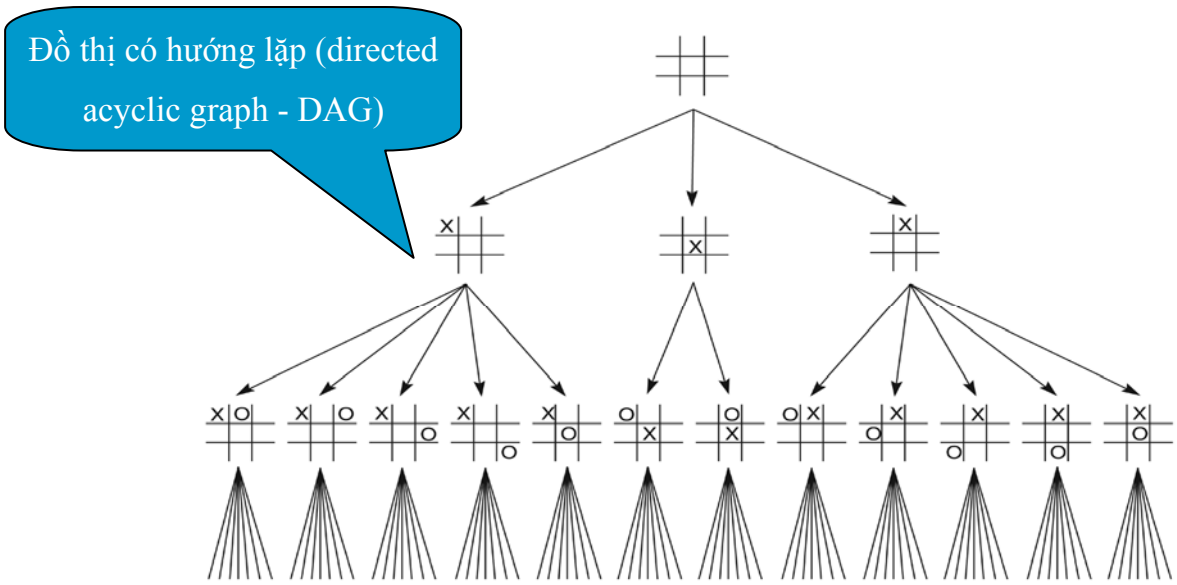
- Việc tìm kiếm có kết thúc không?
- Có tìm thấy lời giải của bài toán không?
- Có tìm được lời giải tối ưu không?

Để thực hiện tìm kiếm, trước hết phải tìm cách biểu diễn bài toán trong *không gian tìm kiếm*. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm tìm kiếm (có thể là không gian liên tục, không gian các đối tượng rời rạc, không gian vecto,…). Không gian tìm kiếm được thể hiện bởi *không gian trạng thái*. Việc biểu diễn bài toán trong không gian trạng thái, cần xác định các yếu tố sau:

- Trạng thái xuất phát
- Tập hợp các toán tử
- Tập hợp các trạng thái kết thúc (trạng thái đích).

Không gian trạng thái có thể được biểu diễn bởi đồ thị có hướng: mỗi đỉnh của đồ thị tương ứng với 1 trạng thái, nếu toán tử R biến đổi từ trạng thái u đến trạng thái v thì có 1 cung gán nhãn R nối hai đỉnh u và v.

Ví dụ:



Hình 2.1: Không gian trạng thái của trò chơi tic-tac-toe

Khi biểu diễn một vấn đề như một đồ thị không gian trạng thái, chúng ta có thể sử dụng lý thuyết đồ thị để phân tích cấu trúc và độ phức tạp của các vấn đề cũng như thực hiện các thủ tục tìm kiếm. Quá trình tìm kiếm là đi xây dựng *cây tìm kiếm* với gốc của cây tương ứng với trạng thái xuất phát, các đỉnh tương ứng với các trạng thái trong đồ thị không gian trạng thái.

Các kỹ thuật tìm kiếm được áp dụng rộng rãi trong lĩnh vực TTNT :

- Tìm kiếm mù : không có hiểu biết gì về các đối tượng để hướng dẫn tìm kiếm

- Tìm kiếm kinh nghiệm (heuristic) : dựa vào kinh nghiệm và hiểu biết về vấn đề cần giải quyết để xây dựng *hàm đánh giá* hướng dẫn sự tìm kiếm.

+ Tìm kiếm tối ưu

+ Tìm kiếm có đối thủ : tìm kiếm nước đi trong các trò chơi hai người (cờ vua, cờ tướng,...)

2.2 Tìm kiếm có hệ thống

Tìm kiếm mù là chiến lược tìm kiếm không có sự hướng dẫn nào cho tìm kiếm, chỉ phát triển các trạng thái từ trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có hai thuật toán tìm kiếm đơn giản:

2.2.1 Tìm kiếm theo chiều rộng:

- Ý tưởng: Bắt đầu mở rộng từ nút gốc, sau đó lần lượt mở rộng các nút được *sinh ra* từ nút gốc, tiếp đến những nút kế tiếp của các nút này, và cứ như vậy cho đến khi tìm thấy một nút đích nào đó hoặc không còn nút nào được sinh ra. Trong đó, nút B gọi là được sinh ra từ nút A nếu B là kề với A trong đồ thị không gian trạng thái. Do đó, tại mỗi bước, trạng thái chọn để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác

- Thuật toán:

Procedure *Breadth_first_Search*;

begin

1. Khởi tạo dsách L chỉ chứa trạng thái ban đầu;

2. **Loop do**

2.1 **if** L rỗng **then** {thông báo tìm kiếm thất bại; stop};

2.2 Loại trạng thái u đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo tìm kiếm thành công; stop};

2.4 **for** mỗi trạng thái v kề u **do**

{Đặt v vào cuối danh sách L;

father(v) ← u};

end;

- Đánh giá thuật toán: Danh sách L được xử lý như hàng đợi

+ Nếu bài toán có nghiệm (tồn tại đường đi từ trạng thái ban đầu tới trạng thái đích) thì thuật toán sẽ tìm ra nghiệm và đường đi là ngắn nhất.

+ Nếu bài toán vô nghiệm, không gian trạng thái hữu hạn, thuật toán dừng và thông báo vô nghiệm.

+ Gọi b là nhân tố nhánh, nghiệm của bài toán là đường đi có độ dài d, độ phức tạp $O(b^d)$.

- Ví dụ:

2.2.2 Tìm kiếm theo chiều sâu

- Ý tưởng: Bắt đầu mở rộng từ nút gốc, sau đó mở rộng một trong các nút ở mức sâu nhất của cây. Nếu tìm đến một điểm cụt (tức là đến nút không phải nút đích và lại không mở rộng được nữa). Như vậy, tại mỗi bước, trạng thái chọn để phát triển là trạng thái được sinh ra sau các trạng thái chờ phát triển khác

- Thuật toán:

Procedure *Deepth_first_Search*;

begin

1. Khởi tạo dsách L chỉ chứa trạng thái ban đầu;

2. **Loop do**

2.1 **if** L rỗng **then** {thông báo tìm kiếm thất bại; stop};

2.2 Loại trạng thái u đầu danh sách L;

2.3 **if** u là trạng thái kết thúc **then**

{thông báo tìm kiếm thành công; stop};

2.4 **for** mỗi trạng thái v kề u **do**

{Đặt v vào đầu danh sách L;

father(v) ← u};

end;

- Ví dụ:

- Đánh giá thuật toán: Danh sách L được xử lý như ngăn xếp

+ Nếu bài toán có nghiệm, không gian trạng thái hữu hạn thì thuật toán sẽ tìm ra nghiệm. Nếu không gian trạng thái vô hạn thì có thể không tìm ra nghiệm→ không nên dùng thuật toán này với bài toán có cây tìm kiếm chứa các nhánh vô hạn

+ Nghiệm bài toán là đường đi có độ dài d, cây tìm kiếm có nhân tố nhánh b, độ phức tạp trong trường hợp tồi nhất $O(b^d)$, độ phức tạp không gian là O(db).

2.3 Tìm kiếm có sử dụng hàm đánh giá

Tìm kiếm kinh nghiệm (heuristic) là kỹ thuật tìm kiếm dựa vào kinh nghiệm, sự hiểu biết, trực giác để đưa các phỏng đoán, ước chừng trong các bước tìm kiếm.

Lĩnh vực AI giải quyết các bài toán trong hai tình huống cơ bản sau:

- Bài toán được định nghĩa chính xác nhưng *chi phí tìm lời giải bằng tìm kiếm vét cạn là không thể chấp nhận*. Ví dụ: sự bùng nổ không gian tìm kiếm trong trò chơi cờ vua,...

- Lời phát biểu bài toán hay dữ liệu cũng như tri thức sẵn có của bài toán mang *tính mơ hồ*. Ví dụ: chẩn đoán trong y học, các sự cố hỏng hóc của máy móc,…

Việc giải quyết bài toán bằng tìm kiếm heuristic thường thực hiện ba bước chính sau:

- Tìm biểu diễn thích hợp mô tả các trạng thái và các toán tử biến đổi trạng thái của bài toán

- Xây dựng hàm đánh giá *(evaluation function)*, dùng để đánh giá các đặc điểm của một trạng thái trong KGTT
- Thiết kế chiến lược chọn các trạng thái để phát triển ở mỗi bước: *Tìm kiếm tốt nhất-đầu tiên (best-first search) và tìm kiếm leo đồi (hill-climbing).*

Hàm đánh giá: Không có một phương pháp tổng quát để xác định hàm đánh giá, mà tùy vào từng bài toán cụ thể và dựa vào kinh nghiệm người ta có thể đưa ra các đánh giá khác nhau. Dưới đây là một số ví dụ:

Ví dụ*:* trò chơi 8 số (8-puzzle): các con số liên tiếp từ 1 đến 8 được đặt trong một bàn cờ 3x3 (9 ô), như vậy sẽ có 1 ô trống. Người ta di chuyển ô trống này sao cho có thể đưa bàn cờ về trạng thái mà tất cả các con số đều xếp theo thứ tự lien tiếp theo từ ngoài vào trong.

1	5	2
8	3	4
6	7	

a.Trạng thái đầu

1	2	3
8		4
7	6	5

b.Trạng thái đích

Hình 2.2: Ví dụ về trò chơi 8-puzzle

Một số hàm đánh giá có thể được xác định như sau:

- Hàm đánh giá h_1 = số quân cờ nằm sai vị trí. Trong hình 2.2.a) thì h_1= 5

- Hàm đánh giá h_2 = tổng số khoảng cách của các quân cờ so với vị trí mục tiêu (trạng thái đích). Khoảng cách các quân cờ được tính bằng số di chuyển theo chiều ngang hoặc theo chiều dọc của các quân cờ để đến vị trí mục tiêu. Ví dụ trong hình 2.2.a) thì h_2 = 1+2+3+1+1= 8.

2.3.1 Tìm kiếm tốt nhất đầu tiên

- Ý tưởng: tìm kiếm theo chiều rộng kết hợp với hàm đánh giá

- Thuật toán:

Procedure Best-first search;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

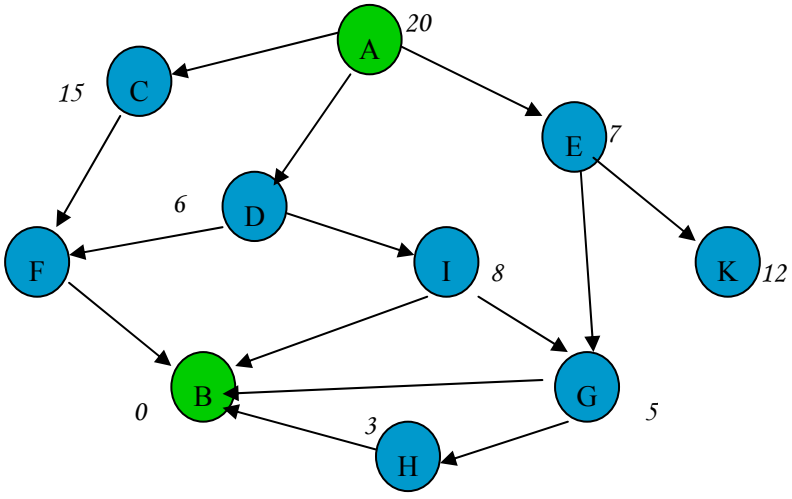
{thông báo thành công; stop};

2.4 **For** mỗi trạng thái v kề u **do**

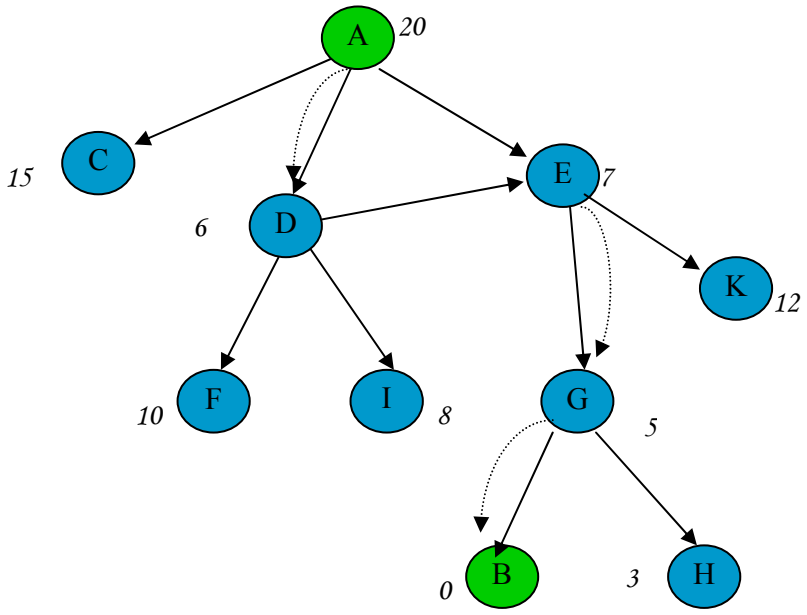
Chèn v vào danh sách L sao cho L được sắp theo thứ tự tăng dần của hàm đánh giá;

End;

- Ví dụ:



Hình 2.2: Đồ thị không gian trạng thái



Hình 2.3: Cây tìm kiếm tốt nhất-đầu tiên

2.3.2 Tìm kiếm leo đồi

- Ý tưởng: Tìm kiếm theo chiều sâu kết hợp với hàm đánh giá. Mở rộng trạng thái hiện tại và đánh giá các trạng thái con của nó bằng hàm đánh giá heuristic. Tại mỗi bước, con "tốt nhất" sẽ được chọn để đi tiếp.

- Thuật toán:

Procedure Hill-Climbing_search;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

{thông báo thành công; stop};

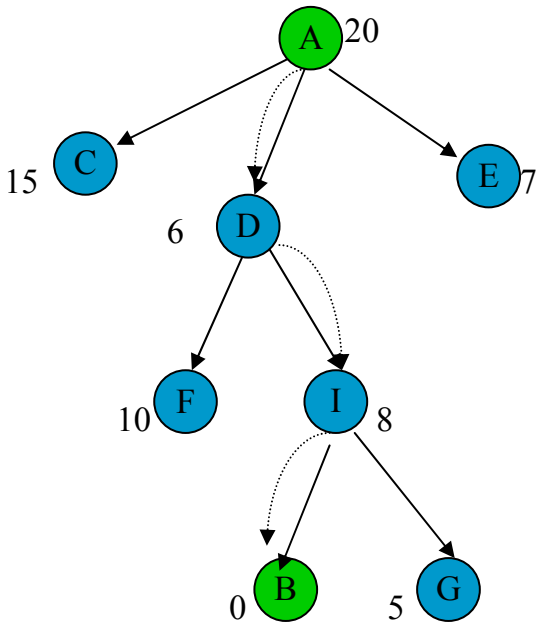
2.4 **For** mỗi trạng thái v kề u **do** đặt v vào L1;

2.5 Sắp xếp L1 theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái tốt nhất ở đầu danh sách L1;

2.6 Chuyển danh sách L1vào đầu danh sách L;

End;

Ví dụ : Với ví dụ đồ thị không gian trạng thái như hình 2.2 thì cây tìm kiếm leo đồi tương ứng như hình 2.4 :



Hình 2.4: Cây tìm kiếm leo đồi

Hạn chế của thuật toán :

- Giải thuật có khuynh hướng bị sa lầy ở những cực đại cục bộ:

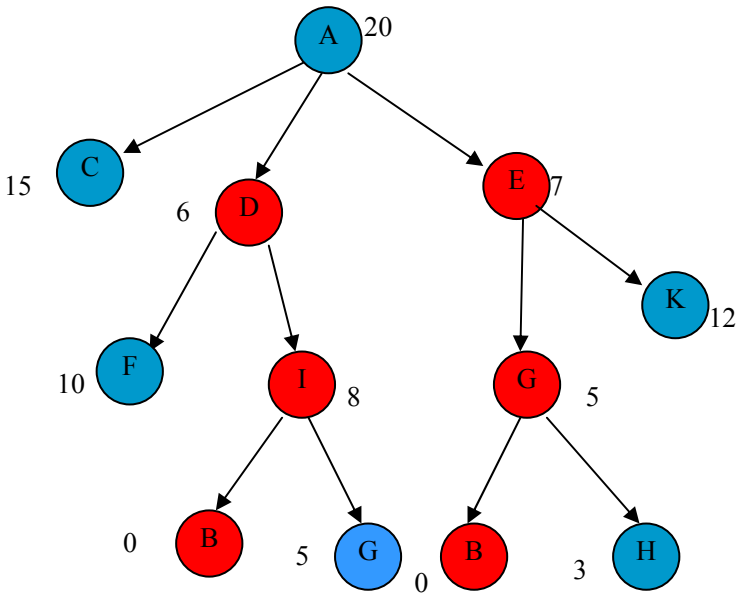
+ Lời giải tìm được không tối ưu

+ Không tìm được lời giải mặc dù có tồn tại lời giải

- Giải thuật có thể gặp vòng lặp vô hạn do không lưu giữ thông tin về các trạng thái đã duyệt.

2.3.3 Tìm kiếm Beam

Để hạn chế không gian tìm kiếm, người ta đưa ra phương pháp tìm kiếm Beam. Đây là phương pháp tìm kiếm theo chiều rộng nhưng có hạn chế số đỉnh phát triển ở mỗi mức. Trong tìm kiếm theo chiều rộng, tại mỗi mức ta phát triển tất cả các đỉnh, còn tìm kiếm Beam thì chọn *k* đỉnh tốt nhất để phát triển. Các đỉnh này được xác định bởi hàm đánh giá. Ví dụ, với đồ thị không gian trạng thái như hình 2.2 và lấy *k*=2 thì cây tìm kiếm Beam như hình 2.5. Các đỉnh được chọn ở mỗi mức là các đỉnh được tô màu đỏ:



Hình 2.5: Cây tìm kiếm Beam

2.4 Tìm kiếm tối ưu

Tìm kiếm tối ưu là tối thiểu chi phí ước lượng để đi đến mục tiêu. Một cách tổng quát, trong không gian tìm kiếm, mỗi đối tượng x được gán một giá trị hàm giá f(x), ta cần tìm đối tượng x mà f(x) là lớn nhất (hoặc nhỏ nhất). Hàm f(x) được gọi là *hàm mục tiêu*. Trong phần này, chúng tôi trình bày một số thuật toán trong bài toán tìm đường

đi ngắn nhất (thuật toán A*, thuật toán nhánh-cận) ; tìm đối tượng tốt nhất và thuật toán di truyền.

2.4.1 Tìm đường đi ngắn nhất

Vấn đề tìm kiếm như đã trình bày trong phần trước là tìm đường đi từ trạng thái xuất phát đến trạng thái đích trong không gian trạng thái. Trong thực tế, thường người ta phải tính đến chi phí di chuyển các trạng thái, từ trạng thái u đến trạng thái v, biểu diễn bởi một số không âm. Giá trị này được gọi là trọng số của cung (u, v) trong đồ thị không gian trạng thái. Trọng số này được xác định tùy thuộc từng bài toán cụ thể. Chẳng hạn, trong bài toán tìm đường đi trên bản đồ, trọng số của cung (u,v) là độ dài của đoạn đường từ địa điểm u đến địa điểm v. Độ dài đường đi được xác định bằng tổng độ dài các cung trên đường đi. Mục tiêu của chúng ta là tìm đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái đích.

Không gian tìm kiếm là tất cả các đường đi từ trạng thái xuất phát đến trạng thái đích, hàm mục tiêu là độ dài đường đi. Người ta có thể giải quyết bài toán bằng các phương pháp tìm kiếm mù (tìm tất cả các đường đi từ trạng thái xuất phát đến trạng thái đích), sau đó so sánh độ dài của chúng và tìm ra đường đi ngắn nhất. Tuy nhiên, trong thực tế không thể áp dụng kỹ thuật này vì với bài toán có không gian tìm kiếm lớn thì đòi hỏi chi phí về thời gian rất cao. Mặt khác, các phương pháp heuristic như tìm kiếm tốt nhất - đầu tiên cho kết quả là đường đi ngắn nhất nhưng có thể kém hiệu quả, hay tìm kiếm leo đồi cũng chỉ cho kết quả là đường đi «tương đối tốt » chứ không đảm bảo đường đi là ngắn nhất. Do đó, để tăng hiệu quả tìm kiếm, chúng ta cần các phương pháp tìm kiếm heuristic sử dụng hàm đánh giá kết hợp :

- Hàm g(u) : đánh giá độ dài đường đi ngắn nhất từ đỉnh xuất phát u_0 đến u. Trong đó, đường đi từ u_0 đến u không phải là trạng thái đích thì được gọi là đường đi một phần, đường đi từ u_0 đến trạng thái đích u gọi là đường đi đầy đủ.

- Hàm h(u) : đánh giá độ dài đường đi ngắn nhất từ đỉnh u đến đích.

Trong đó, hàm h(u) gọi là *chấp nhận được* (đánh giá thấp) nếu với mọi trạng thái *u*, h(u) <= độ dài đường đi ngắn nhất trong thực tế từ *u* đến đích. Ví dụ, trong bài toán tìm đường đi thì h(u) là độ dài đường chim bay từ địa điểm *u* đến đích.

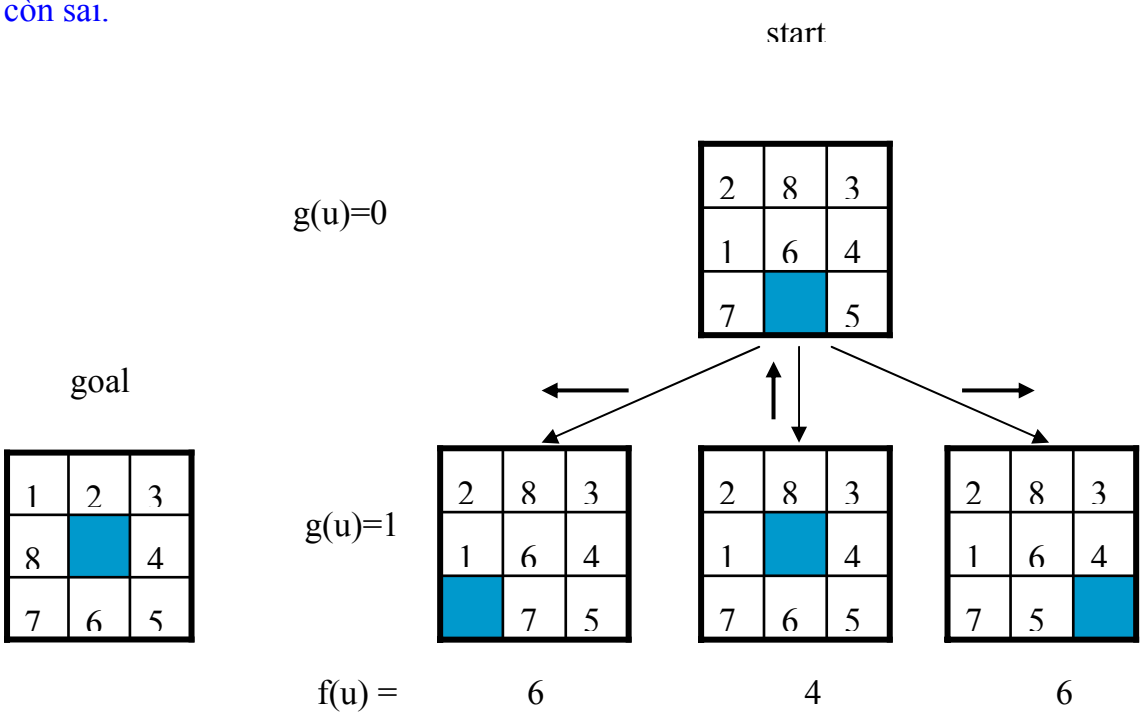
Như vậy, hàm đánh giá là f(u) = g(u) + h(u), đánh giá độ dài đường đi ngắn nhất từ trạng thái xuất phát đến trạng thái đích mà đi qua trạng thái *u*.

Ví dụ về cài đặt hàm đánh giá : Xét trò chơi 8-puzzle.

Cho mỗi trạng thái u một giá trị f(u) = g(u) + h(u)

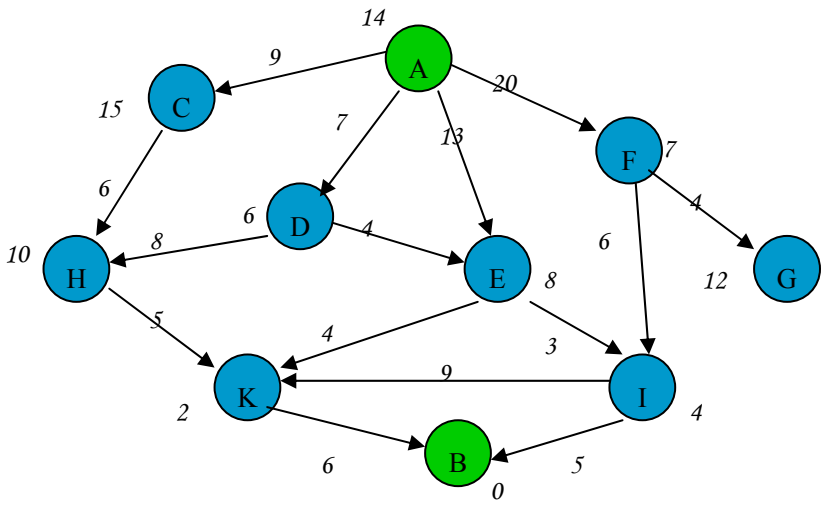
g(u) là khoảng cách thực sự từ u đến trạng thái bắt đầu

h(u) là hàm heuristic đánh giá khoảng cách từ trạng thái u đến mục tiêu, là số vị trí còn sai.



Hình 2.6: Minh họa hàm đánh giá cho trò chơi 8-puzzle

Ví dụ : hình 2.7 minh họa một đồ thị không gian trạng thái với hàm đánh giá: các số ghi cạnh các đỉnh là giá trị của hàm *h*, các số ghi trên các cung là độ dài cung đó.



Hình 2.7: Đồ thị không gian trạng thái với hàm đánh giá

a)- Thuật toán A*

- Ý tưởng : thuật toán tìm kiếm tốt nhất – đầu tiên với hàm đánh giá f(u)

- Thuật toán :

Procedure A*;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

{thông báo thành công; stop};

2.4 **For** mỗi trạng thái v kề u **do**

{g(v)←g(u)+k(u,v)

f(v)←g(v)+h(v);

đặt v vào danh sách L;}

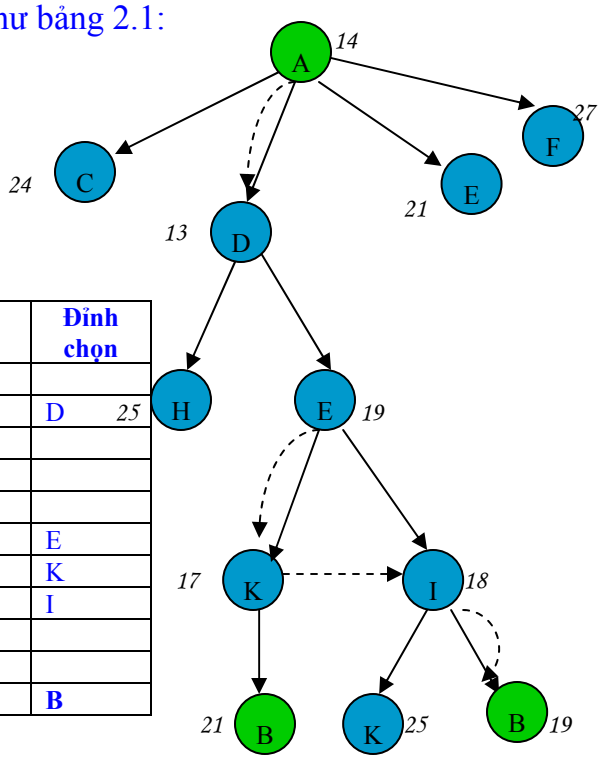
2.5 Sắp xếp L theo thứ tự tăng dần của hàm f;

End;

- Ví dụ : Với đồ thị không gian trạng thái như hình 2.7, đỉnh xuất phát A và đỉnh đích B. Áp dụng thuật toán A*, ta xây dựng được cây tìm kiếm như hình 2.8 và giá trị của hàm *f* tại các đỉnh được tính như bảng 2.1:

Đỉnh phát triển (u)	Đỉnh con (v)	g(v)	f(v)	Đỉnh chọn
A	C	9	9+15=24	
	D	7	7+6=13	D
	E	13	13+8=21	
	F	20	20+7=27	
D	H	7+8=15	15+10=25	
	E	7+4=11	11+8=19	E
E	K	11+4=15	15+2=17	K
	I	11+3=14	14+4=18	I
K	B	15+6=21	21+0=21	
I	K	14+9=23	23+2=25	
	B	14+5=19	19+0=19	**B**

*Bảng 2.1: Tính giá trị hàm f cho thuật toán A**



*Hình 2.8: Cây tìm kiếm A**

- Nhận xét :

+ Nếu h(u) là đánh giá thấp (đặc biệt h(u)=0 với mọi trạng thái *u*), thì A* là thuật toán tối ưu, tức là nghiệm tìm được là tối ưu. Nếu độ dài các cung không nhỏ hơn một số dương δ nào đó thì A* là thuật toán đầy đủ, tức là nó luôn dừng và tìm ra nghiệm.

+ Trường hợp hàm đánh giá h(u)=0 với mọi *u*, thuật toán A* chính là thuật toán tìm kiếm tốt nhất – đầu tiên với hàm đánh giá g(u).

+ Thuật toán A* đã được chứng minh là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho bài toán tìm đường đi ngắn nhất.

b)- Thuật toán nhánh - cận

- Ý tưởng : thuật toán tìm kiếm leo đồi kết hợp với hàm đánh giá *f(u)*. Tại mỗi bước, khi phát triển trạng thái *u*, chọn trạng thái con v tốt nhất (*f(v)* nhỏ nhất) của *u* để phát triển ở bước sau. Quá trình tiếp tục như vậy cho đến khi gặp trạng thái *w* là đích, hoặc *w* không có đỉnh kề, hoặc *w* có *f(w)* lớn hơn độ dài đường đi tối ưu tạm thời (đường đi đầy đủ ngắn nhất trong số những đường đi đầy đủ đã tìm được). Trong các trường hợp này, chúng ta không phát triển đỉnh *w* nữa, tức là cắt bỏ những nhánh xuất phát từ *w*, và quay lên cha của *w* để tiếp tục đi xuống trạng thái tốt nhất trong số những trạng thái còn lại chưa được phát triển.

- Thuật toán :

Procedure Branch-and-Bound;

Begin

1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;

Gán giá trị ban đầu cho cost; */*cost là giá trị đường đi tối ưu tạm thời*/*

2. **Loop do**

2.1 **If** L rỗng **then** {thông báo thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **If** u là trạng thái kết thúc **then**

if g(u)<=cost then {cost ←g(u); quay lại 2.1};

2.4 **if** f(u)>cost **then** quay lại 2.1;

2.5 **For** mỗi trạng thái v kề u **do**

{g(v) ←g(u)+k(u,v);

f(v) ←g(v) +h(v);

đặt v vào danh sách L1};

2.6 Sắp xếp L1 theo thứ tự tăng dần của hàm f;

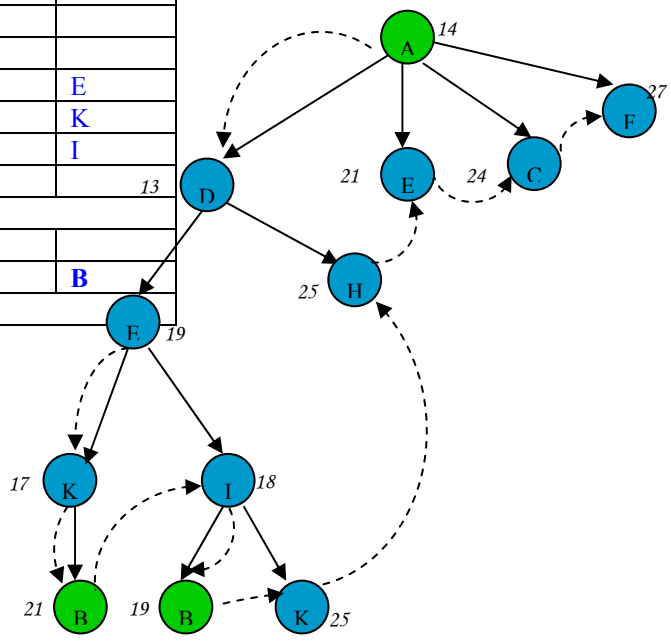
2.7 Chuyển danh sách L1vào đầu danh sách L sao cho L1 ở đầu danh sách L;

End;

- Ví dụ : Với đồ thị không gian trạng thái như hình 2.7, đỉnh xuất phát A và đỉnh đích B. Áp dụng thuật toán nhánh – cận, ta xây dựng được cây tìm kiếm như hình 2.9 và giá trị của hàm *f* tại các đỉnh được tính như bảng 2.2:

Đỉnh phát triển (u)	Đỉnh con (v)	g(v)	f(v)	Đỉnh chọn
A	C	9	9+15=24	
	D	7	7+6=13	D
	E	13	13+8=21	
	F	20	20+7=27	
D	H	7+8=15	15+10=25	
	E	7+4=11	11+8=19	E
E	K	11+4=15	15+2=17	K
	I	11+3=14	14+4=18	I
K	B	15+6=21	21+0=21	
B	**cost := 21**			
I	K	14+9=23	23+2=25	
	B	14+5=19	19+0=19	**B**
B	**cost := 19**			

Bảng 2.2: Tính giá trị hàm f cho thuật toán nhánh-cận



Hình 2.9 : Cây tìm kiếm nhánh-cận

- Nhận xét : Thuật toán nhánh-cận cũng là thuật toán đầy đủ và tối ưu nếu h(u) là hàm đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó

2.4.2 Tìm đối tượng tốt nhất

Chương 3 – Các giải thuật tìm kiếm lời giải cho trò chơi

Chương trình chơi cờ đầu tiên được viết bởi Claude Shannon vào năm 1950 đã là một minh chứng cho khả năng máy tính có thể làm được những việc đòi hỏi trí thông minh của con người. Từ đó người ta nghiên cứu các chiến lược chơi cho máy tình với các trò chơi có đối thủ (có hai người tham gia). Việc giải quyết bài toán này có thể đưa về bài toán tìm kiếm trong không gian trạng thái, tức là tìm một chiến lược chọn các nước đi hợp lệ cho máy tính. Tuy nhiên, vấn đề tìm kiếm ở đây phức tạp hơn so với vấn đề tìm kiếm trong chương trước, vì người chơi không biết trước đối thủ sẽ chọn nước đi nào tiếp theo. Chương này sẽ trình bày một số chiến lược tìm kiếm phổ biến như Minimax, phương pháp cắt cụt α-β.

3.1 Cây trò chơi đầy đủ

Các trò chơi có đối thủ có các đặc điểm: hai người thay phiên nhau đưa ra các nước đi tuân theo các luật của trò chơi (các nước đi hợp lệ), các luật này là như nhau đối với cả hai người chơi, chẳng hạn các trò chơi cờ: cờ vua, cờ tướng, cờ ca rô (tic-tăc-toe), …. Ví dụ, trong chơi cờ vua, một người điều khiển quân Trắng và một người điều khiển quân Đen. Người chơi có thể lựa chọn các nước đi theo các luật với các quân tốt, xe, mã,… Luật đi quân tốt Trắng, xe Trắng, mã Trắng,… giống luật đi quân tốt Đen, xe Đen, mã Đen,…Hơn nữa, cả hai người chơi đều biết đầy đủ các thông tin về tình thế cuộc chơi. Thực hiện trò chơi là người chơi tìm kiếm nước đi *tốt nhất* trong số rất nhiều nước đi hợp lệ, tại mỗi lượt chơi của mình, sao cho sau một dãy nước đi đã thực hiện người chơi phải thắng cuộc.

Vấn đề chơi cờ có thể được biểu diễn trong không gian trạng thái, ở đó, mỗi trạng thái là một tình thế của cuộc chơi (sự sắp xếp các quân cờ trên bàn cờ):

- Trạng thái xuất phát là sự sắp xếp các quân cờ của hai bên khi bắt đầu cuộc chơi (chưa ai đưa ra nước đi)
- Các toán tử biến đổi trạng thái là các nước đi hợp lệ

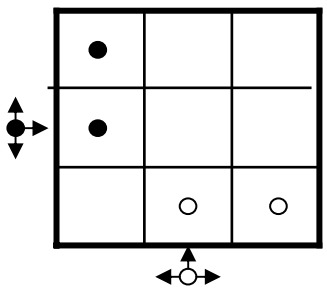
- Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường được xác định bởi một số điều kiện dừng (chẳng hạn, quân Trắng thắng hoặc quân Đen thắng hoặc hai bên hòa nhau)
- Hàm kết cuộc: mang giá trị tương ứng với mỗi trạng thái kết thúc. Chẳng hạn, trong cờ vua, hàm kết cuộc có giá trị là 1 tại các trạng thái mà Trắng thắng, -1 tại các trạng thái mà Trắng thua và 0 tại các trạng thái hai bên hòa nhau. Trong các trò chơi tính điểm khác thì hàm kết cuộc có thể nhận các giá trị nguyên trong đoạn [-m, m], với m là một số nguyên dương nào đó.

Như vậy, trong các trò chơi có đối thủ, người chơi (điều khiển quân Trắng – gọi tắt là Trắng) luôn tìm một dãy các nước đi xen kẽ với các nước đi của đối thủ (điều khiển quân Đen – gọi tắt là Đen) để tạo thành một đường đi từ trạng thái ban đầu đến trạng thái kết thúc là thắng cho Trắng.

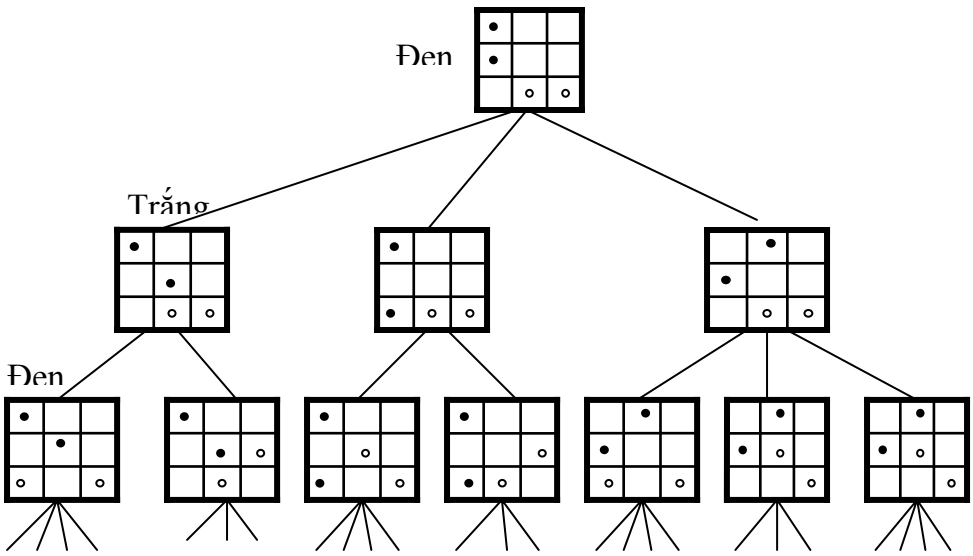
Không gian tìm kiếm đối với các trò chơi này có thể được biểu diễn bởi *cây trò chơi* như sau: gốc của cây ứng với trạng thái xuất phát, các đỉnh trên cây tương ứng với các trạng thái của bàn cờ, các cung (u, v) nếu có biến đổi từ trạng thái u đến trạng thái v. Các đỉnh trên cây được gán nhãn là đỉnh Trắng (Đen) ứng với trạng thái mà quân Trắng (Đen) đưa ra nước đi. Nếu một đỉnh u được gán nhãn là Trắng (Đen) thì các đỉnh con v của nó là tất cả các trạng thái nhận được từ u do Trắng (Đen) thực hiện một nước đi hợp lệ nào đó. Do đó, các đỉnh trên cùng một mức của cây đều có nhãn là Trắng hoặc đều có nhãn là Đen, các lá của cây ứng với trạng thái kết thúc.

Ví dụ: trò chơi Dodgem:

Có hai quân Trắng và hai quân Đen được xếp vào bàn cờ 3x3. Ban đầu các quân cờ được xếp như hình bên. Quân Đen có thể đi đến ô trống bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi đến ô trống bên trên, bên trái hoặc bên phải. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa được cả hai quân của mình ra khỏi bàn cờ hoặc tạo ra tình thế mà đối phương không đi được là thắng cuộc.



Trò chơi Dodgem



Cây trò chơi Dodgem với Đen đi trước

3.2 Giải thuật Minimax

Quá trình chơi cờ là quá trình mà Trắng và Đen thay phiên nhau đưa ra các nước đi hợp lệ cho đến khi dẫn đến trạng thái kết thúc cuộc chơi. Quá trình này biểu diễn bởi đường đi từ nút gốc tới nút lá trên cây trò chơi. Giả sử tại một đỉnh u nào đó trên đường đi, nếu u là đỉnh Trắng (Đen) thì cần chọn một nước đi nào đó đến một trong các đỉnh con Đen (Trắng) v của u. Tại đỉnh Đen (Trắng) v sẽ chọn đi tiếp đến một đỉnh con Trắng (Đen) w của v. Quá trình này tiếp tục cho đến khi đạt đến một đỉnh lá của cây.

Chiến lược tìm nước đi của Trắng hay Đen là luôn tìm những nước đi dẫn tới trạng thái tốt nhất cho mình và tồi nhất cho đối thủ. Giả sử Trắng cần tìm nước đi tại đỉnh u, nước đi tối ưu cho Trắng là nước đi dẫn tới đỉnh con v sao cho v là tốt nhất trong số các đỉnh con của u. Đến lượt Đen chọn nước đi từ v, Đen cũng chọn nước đi tốt nhất cho mình. Để chọn nước đi tối ưu cho Trắng tại đỉnh u, cần xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá ứng với giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất (các đỉnh lá)

lên gốc u. Giả sử cần xác định giá trị của đỉnh v mà các đỉnh con của nó đã xác định. Khi đó, nếu v là đỉnh Trắng thì giá trị của nó là giá trị lớn nhất trong các đỉnh con, nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các đỉnh con.

Sau đây là thủ tục chọn nước đi cho Trắng tại đỉnh u Minimax(u, v), trong đó v là đỉnh con được chọn của u:

```
Procedure Minimax(u, v);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w)  then
            {val ← MinVal(w);  v ←  w}
end;
Function MinVal(u); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc  then  MinVal(u) ← f(u)
    else MinVal(u) ← min{MaxVal(v) | v là đỉnh con của u}
end;
Function MaxVal(u); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc  then  MaxVal(u) ← f(u)
    else MaxVal(u) ← max{MinVal(v) | v là đỉnh con của u}
end;
```

Trong các thủ tục và hàm trên, f(u) là giá trị của hàm kết cuộc tại đỉnh kết thúc u.

Thuật toán Minimax là thuật toán tìm kiếm theo chiều sâu. Về lý thuyết, chiến lược Minimax cho phép tìm nước đi tối ưu cho Trắng. Tuy nhiên trong thực tế, ta không có đủ thời gian để tính toán nước đi tối ưu này. Bởi vì thuật toán tính toán trên toàn bộ cây trò chơi (xem xét tất cả các đỉnh của cây theo kiểu vét cạn). Trong các trò chơi hay thì kích thước của cây trò chơi là cực lớn. Chẳng hạn, trong cờ vua, chỉ tính đến độ sâu 40 thì cây trò chơi đã có đến 10^{120} đỉnh. Nếu cây có độ cao m và tại mỗi đỉnh có b nước đi thì độ phức tạp về thời gian của thuật toán Minimax là $O(b^m)$.

Trong thực tế, các trò chơi đều có giới hạn về thời gian. Do đó, để có thể tìm nhanh nước đi tốt (không phải tối ưu) thay vì sử dụng hàm kết cuộc và xét tất cả các đỉnh của cây trò chơi, ta sử dụng hàm đánh giá và chỉ xem xét một bộ phận của cây trò chơi.

3.3 Giải thuật Minimax với độ sâu hạn chế

3.3.1 Hàm đánh giá

Hàm đánh giá eval cho mỗi đỉnh u là đánh giá "mức độ lợi thế" của trạng thái u. Giá trị của eval(u) là số dương càng lớn thì trạng thái u càng có lợi cho Trắng, giá trị của eval(u) là số dương càng nhỏ thì trạng thái u càng có lợi cho Đen, eval(u)=0 thì trạng thái u không có lợi cho đối thủ nào, eval(u)=+∞ thì u là trạng thái thắng cuộc cho Trắng, eval(u)=-∞ thì u là trạng thái thắng cuộc cho Đen.

Hàm đánh giá đóng vai trò rất quan trọng trong các trò chơi, nếu hàm đánh giá tốt sẽ định hướng chính xác việc lựa chọn các nước đi tốt. Việc thiết kế hàm đánh giá phụ thuộc vào nhiều yếu tố: các quân cờ còn lại của hai bên, sự bố trí các quân cờ này,… Để đưa ra hàm đánh giá chính xác đòi hỏi nhiều thời gian tính toán, tuy nhiên, trong thực tế người chơi bị giới hạn thời gian đưa ra nước đi. Vì vậy, việc đưa ra hàm đánh giá phụ thuộc vào kinh nghiệm của người chơi. Sau đây là một số ví dụ về cách xây dựng hàm đánh giá:

Ví dụ 1: Hàm đánh giá cho cờ vua. Mỗi loại quân được gán một giá trị số phù hợp với "sức mạnh" của nó. Chẳng hạn, quân tốt Trắng (Đen) được gán giá trị 1 (-1), mã hoặc

tượng Trắng (Đen) được gán giá trị 3 (-3), xe Trắng (Đen) được gán giá trị 5 (-5) và hậu Trắng (Đen) được gán giá trị 9 (-9). Hàm đánh giá của một trạng thái được tính bằng cách lấy tổng giá trị của tất cả các quân cờ trong trạng thái đó. Hàm đánh giá này được gọi là hàm tuyến tính có trọng số, vì có thể biểu diễn dưới dạng:

$$s_1w_1 + s_2w_2 + \ldots + s_nw_n$$

Trong đó, w_i là giá trị của quân cờ loại i, s_i là số quân loại đó.

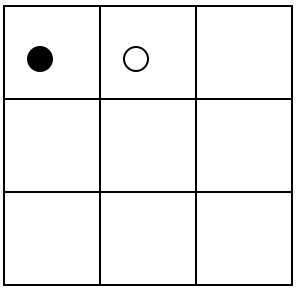
Đây là cách đánh giá đơn giản, vì nó không tính đến sự bố trí của các quân cờ, các mối tương quan giữa chúng.

Ví dụ 2: Hàm đánh giá trạng thái trong trò chơi Dodgem. Mỗi quân Trắng được gán giá trị tương ứng với các vị trí trên bàn cờ như trong hình bên trái. Mỗi quân Đen được gán giá trị ở các vị trí tương ứng nhu hình bên phải:

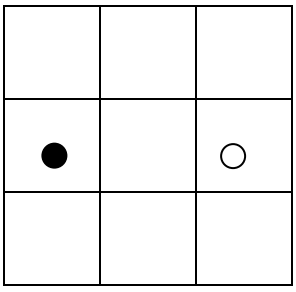
30	35	40
15	20	25
0	5	10

-10	-25	-40
-5	-20	-35
0	-15	-30

Ngoài ra, nếu quân Trắng cản trực tiếp một quân Đen, nó được thêm 40 điểm, nếu cản gián tiếp được thêm 30 điểm (xem hình dưới). Tương tự, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, cản gián tiếp được thêm -30 điểm.

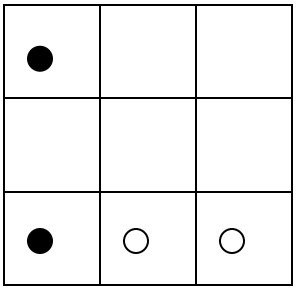


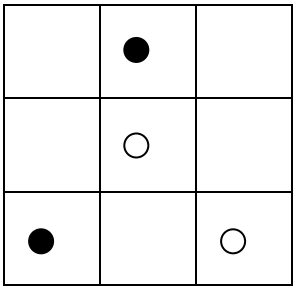
Trắng cản trực tiếp Đen được thêm 40 điểm



Trắng cản gián tiếp Đen được thêm 30 điểm

Áp dụng cách tính hàm đánh giá nêu trên, ta tính được giá trị của các trạng thái ở các hình dưới như sau:





Giá trị hàm đánh giá:75=
(-10+0+5+10)+(40+30)

Giá trị hàm đánh giá:-5=
(-25+0+20+10)+(-40+30)

3.3.2 Thuật toán

Để hạn chế không gian tìm kiếm, khi xác định nước đi cho Trắng tại u, ta chỉ xem xét cây gốc u tại độ cao h nào đó. Áp dụng thủ tục Minimax cho cây trò chơi gốc u, độ cao h và sử dụng hàm đánh giá để xác định giá trị cho các lá của cây.

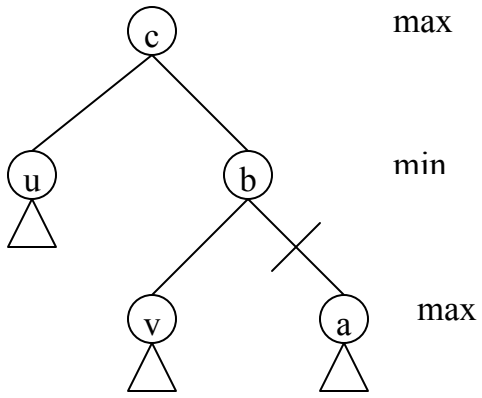
```
Procedure Minimax(u, v, h);
begin
    val ←-∝;
    for  mỗi w là đỉnh con của u  do
    if  val(u) <= MinVal(w, h-1)  then
            {val ← MinVal(w, h-1);  v ←  w}
end;
Function MinVal(u, h); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc or h = 0 then  MinVal(u, h) ← eval(u)
    else MinVal(u, h) ← min{MaxVal(v, h-1) | v là đỉnh con của u}
```

```
end;
Function MaxVal(u, h); { hàm xác định giá trị cho các đỉnh Trắng}
begin
    if  u là đỉnh kết thúc or h =0  then  MaxVal(u, h) ← eval(u)
    else MaxVal(u, h) ← max{MinVal(v, h-1) | v là đỉnh con của u}
end;
```

3.4 Giải thuật Minimax với cắt tỉa alpha-beta

Trong chiến lược Minimax với độ sâu hạn chế thì số đỉnh của cây trò chơi phải xét vẫn còn rất lớn với h>=3. Khi đánh giá đỉnh u tới độ sâu h, thuật toán Minimax đòi hỏi phải đánh giá tất cả các đỉnh của cây gốc u với độ sâu h. Tuy nhiên, phương pháp cắt cụt alpha-beta cho phép cắt bỏ những nhánh không cần thiết cho việc đánh giá đỉnh u. Phương pháp này làm giảm bớt số đỉnh phải xét mà không ảnh hưởng đến kết quả đánh giá đỉnh u.

Ý tưởng: Giả sử tại thời điểm hiện tại đang ở đỉnh Trắng a, đỉnh a có anh em là v đã được đánh giá. Giả sử cha của đỉnh a là b, b có anh em là u đã được đánh giá, và cha của b là c như hình sau:



Cắt bỏ cây con gốc a nếu eval(u)>eval(v)

Khi đó ta có giá trị đỉnh Trắng *c* ít nhất là giá trị của *u*, giá trị của đỉnh Đen *b* nhiều nhất là giá trị của *v*. Do đó, nếu eval(u) > eval(v) ta không cần đi xuống để đánh giá đỉnh *a* nữa mà vẫn không ảnh hưởng đến đánh giá đỉnh *c*. Hay nói cách khác, ta có thể cắt bỏ cây con gốc *a*.

Lập luận tương tự cho trường hợp *a* là đỉnh Đen, trường hợp này nếu eval(u)<eval(v) ta cũng cắt bỏ cây con gốc *a*.

Để cài đặt kỹ thuật này, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Trắng, tham số β để ghi lại giá trị nhỏ nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Đen.

Thuật toán:

```
Procedure   Alpha_beta(u, v);
begin
    α←-∞; β←-∞;
    for  mỗi w là đỉnh con của u  do
    if  α <= MinVal(w, α, β)  then
            {α ← MinVal(w, α, β);  v ←  w}
end;
Function MinVal(u, α, β); {hàm xác định giá trị cho các đỉnh Đen}
begin
    if  u là đỉnh kết thúc or u là lá của cây hạn chế then
        MinVal(u, α, β) ← eval(u)
    else for  mỗi đỉnh v là con của u  do
            {β ← min{β, MaxVal(v, α, β)} ;
```

```
            If α >= β  then  exit};
        /*cắt bỏ các cây con từ các đỉnh v còn lại */
        MinVal(u, α, β) ← β;
    end;
    Function MaxVal(u, α, β); { hàm xác định giá trị cho các đỉnh Trắng}
    begin
        if  u là đỉnh kết thúc or là lá của cây hạn chế  then
            MaxVal(u, α, β) ← eval(u)
        Else for mỗi đỉnh v là con của u  do
            α ← max{α, MinVal(v, α, β)} ;
            If α >= β then exit};
        /*cắt bỏ các cây con từ các đỉnh v còn lại */
        MaxVal(u, α, β) ← α
end;
```