Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

TRƯỜNG ĐẠI HỌC QUY NHƠN
KHOA TIN HỌC

LÊ XUÂN VINH

Giáo Trình

TRÍ TUỆ NHÂN TẠO

Quy Nhơn – 2008

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

MỤC LỤC

Nội dung chương trình

Thời gian: 60 tiết

Nội dung: 3 phần

Phần 1: Giải quyết vấn đề bằng tìm kiếm

Phần 2: Biểu diễn tri thức và lập luận

Phần 3: Một số kỹ thuật trong TTNT hiện đại

Tài liệu tham khảo

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

[1] Trí tuệ nhân tạo – Đinh Mạnh Tường

[2] Trí tuệ nhân tạo – George F. Luger

[3] Trí tuệ nhân tạo – Nguyễn Thanh Thủy

[4] Giáo trình nhập môn Trí tuệ nhân tạo – Hoàng Kiếm

[5] Artificial Intelligence, A Modern Approach - Stuart J. Russell

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

CHƯƠNG 1. GIỚI THIỆU VỀ TRÍ TUỆ NHÂN TẠO
(2 tiết)

1. Một số ứng dụng của trí tuệ nhân tạo

Những năm gần đây chúng ta thường nghe nói nhiều về máy tính thế hệ 5, hệ chuyên gia, lập trình Prolog, logic mờ, mạng nơron nhân tạo, giải thuật di truyền,…Đây là một số thuật ngữ trong một ngành mới của khoa học máy tính. Đó là: Trí tuệ nhân tạo (TTNT). Để hình dung TTNT giải quyết những vấn đề gì, chúng ta hãy xem những ứng dụng với những đòi hỏi cụ thể của nó.

Trò chơi: cờ carô, cờ vua, các ô số,…mỗi một bước đi trên bàn cờ là một quyết định trong số rất nhiều khả năng có thể lựa chọn. Tất cả các khả năng sẽ sinh ra một không gian quá lớn và phức tạp. Sẽ rất khó khăn nếu như sòng phẳng xét hết tất cả các khả năng. Vì lý do thời gian, một người đánh cờ chỉ có thể cảm nhận khả năng tốt trong lựa chọn. Chương trình thông minh phải có khả năng như vậy. Chiến lược lựa chọn mang tính cảm nhận nhưng có cơ sở sẽ được gọi là heuristic, nó không chắc chắn mang lại kết quả nhưng nhiều khả năng mang đến thành công, tuy nhiên vẫn có thể hàm chứa sự rủi ro dẫn đến thất bại. Năm 1953, Samuel đã viết chương trình chơi cờ gây ấn tượng lớn khi nó được công chiếu trên tivi. Chương trình của Ông có khả năng học và khi được huấn luyện có khả năng chơi hay hơn người viết ra nó. Chương trình đánh cờ cho máy tính Deep-Blue (1997) cũng là một ứng dụng của TTNT vào trò chơi.

Hệ chuyên gia: Một chuyên gia phải có nhiều tri thức chuyên môn và kỹ năng sử dụng những tri thức đó để giải quyết vấn đề. Một hệ chương trình thay thế cho chuyên gia được gọi là hệ chuyên gia. Nó bao gồm cơ sở tri thức và các quy tắc suy luận. Hệ chuyên gia đang được ứng dụng rộng rãi trong các lĩnh vực y tế, giáo dục, thiết kế, kinh doanh, khoa học,…Những hệ chuyên gia nổi tiếng như DENDRAL (Stanford, 1960) dùng để phỏng đoán cấu trúc các phân tử hữu cơ từ công thức hóa học của chúng; MYCIN (Stanford, 1970) chẩn đoán và kê đơn điều trị cho bệnh viêm màng não và nhiễm trùng máu; PROSPECTOR (MIT, 1979) xác định vị trí, loại quặng mỏ dựa trên thông tin địa lý. Đòi hỏi cơ bản của mọi hệ chuyên gia là biểu diễn tri thức ngôn ngữ như thế nào và tiếp cận cách suy luận của con người ra sao. Cho đến nay, đó vẫn là những vấn đề nan giải cần giải quyết.

Lập kế hoạch và robot: Lập kế hoạch là xác định một dãy thao tác để đạt được mục đích đặt ra. Đối với con người đây đã là một yêu cầu phức tạp, tuy nhiên có thể giải quyết được do con người có khả năng phán đoán, suy luận. Khi trang bị chương trình như vậy cho robot chúng ta gặp phải những khó khăn: biểu diễn tri thức về không gian, môi trường tác động luôn biến động, số lượng các chuỗi thao tác là rất lớn, thông tin không đầy đủ, thao tác sửa chữa hành vi khi gặp bất lợi,…Các robot của Nhật Bản là những minh chứng cho sự thành công trong việc giải quyết những vấn đề trên.

Điều khiển mờ: Tích hợp các thiết bị điều khiển mờ tự động vào các sản phẩm công nghệ phục vụ đời sống bắt đầu từ những năm 1990 tại Nhật Bản. Điển hình là

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

các sản phẩm như máy giặt, máy điều hòa nhiệt độ của Toshiba; máy ảnh, máy quay phim kỹ thuật số của Canon; hướng dẫn lùi xe tự động của Nissan, Mitshubisi và các ứng dụng trong điều khiển tàu điện không người lái, trong các dây chuyền công nghiệp, sản xuất xi măng,…Đặc trưng của kỹ thuật này dựa trên lý thuyết mờ của L. A. Zadeh (1965) với quan điểm mờ hóa đầu vào các tác động của môi trường nhằm đạt được kết quả liên tục tốt nhất, phù hợp với quan điểm sử dụng ngôn ngữ để mô tả cho dữ liệu của con người.

Hiểu và mô hình hóa ngữ nghĩa ngôn ngữ tự nhiên: Khi đọc cùng một bài viết, mỗi người hiểu một mức độ khác nhau. Điều này phụ thuộc vào tri thức, khả năng nắm bắt, suy luận, xử lý linh hoạt vấn đề, ngữ cảnh của người đọc. Những ứng dụng như dịch máy phải phát triển các kỹ thuật để cấu trúc hóa ý nghĩa ngữ nghĩa. Việc hiểu ngôn ngữ tổng quát theo cách con người vẫn nằm ngoài tầm tay của chúng ta thậm chí chỉ là mức phương pháp luận.

2. Khái niệm về trí tuệ nhân tạo

Trí tuệ nhân tạo (Artificial Intelligence – AI) là thuật ngữ do McCathy đưa ra tại hội thảo Dartmount năm 1956 dùng để chỉ cho một ngành khoa học mới trong lĩnh vực khoa học máy tính. Nghiên cứu những vấn đề liên quan đến tư duy của con người, TTNT kế thừa nhiều ý tưởng, quan điểm, kỹ thuật từ nhiều ngành khoa học khác như Triết học, Toán học, Tâm lý học,…Triết học được nhắc đến như nguồn gốc xa xưa của TTNT khi mà qui tắc suy diễn "modus ponens" (tam đoạn luận) được sử dụng trong suy luận hình thức ngày nay đã được Aristotle đưa ra từ vài nghìn năm trước và nhiều công trình nghiên cứu về tư duy của nhiều nhà triết học khác. Descartes cũng là nhân vật trung tâm trong sự phát triển các khái niệm hiện đại về tư duy và tinh thần với câu nói nổi tiếng "tôi tư duy nghĩa là tôi tồn tại". Các ngành logic, lý thuyết đồ thị, xác suất của Toán học đóng góp rất nhiều cho TTNT. Logic kinh điển Boole, logic vị từ Frege là những cơ sở quan trọng để biểu diễn tri thức. Lý thuyết đồ thị cung cấp công cụ để mô hình một vấn đề, tìm kiếm lời giải, phân tích tính chính xác, tính hiệu quả của các chiến lược tìm kiếm lời giải.

Khác với các ngành khoa học khác nghiên cứu về trí tuệ, TTNT nghiên cứu và tạo ra những thực thể có mang tính trí tuệ và ứng dụng trong sản xuất các thiết bị phục vụ cho đời sống, một xu thế tất yếu của thời đại công nghệ tri thức. Vì vậy, có một số định nghĩa về TTNT được chấp nhận nhiều hơn như sau:

+ TTNT là sự nghiên cứu, thiết kế các chương trình máy tính ứng xử một cách thông minh.

+ TTNT là sự nghiên cứu, thiết kế các tác nhân thông minh (Intelligent Agents).

Các định nghĩa về TTNT tập trung quanh hai quan điểm: suy nghĩ, hành động như con người (think, act like human): quan tâm đến yếu tố kinh nghiệm; và suy nghĩ và hành động hợp lý (think, act rationally) quan tâm đến yếu tố logic của vấn đề.

Thế nào là máy tính hay một chương trình máy tính thông minh? Trắc nghiệm Turing đưa ra dựa trên việc so sánh với khả năng của con người, một đối tượng được

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

coi là có hành vi thông minh và chuẩn mực nhất về trí tuệ. Trắc nghiệm này có người thẩm vấn cách ly với người trả lời thẩm vấn và máy tính. Nếu người thẩm vấn không phân biệt được câu trả lời của máy tính và người thì máy tính đó được coi là thông minh.

Trắc nghiệm như trên không dễ thực hiện và để hiểu đánh giá sự thông minh của chương trình chúng ta tìm hiểu khái niệm tác nhân thông minh (intelligent agent).

Tác nhân là bất cứ cái gì có khả năng nhận thức và tác động phản ứng lại đối với môi trường. Ví dụ robot tiếp nhận các trạng thái của môi trường thông qua các bộ cảm nhận, hành động theo quyết định điều khiển.

Tác nhân được gọi là thông minh nếu nó phản ứng lại môi trường để đạt được mục tiêu hoặc ít nhất là có cơ sở sẽ đạt được mục tiêu đề ra. Một tác nhân thông minh phải có các khả năng sau đây:

- Hiểu ngôn ngữ tự nhiên: tiếng Anh hay một ngôn ngữ nào đó.
- Có khả năng biểu diễn tri thức: thu thập, sử dụng tri thức.
- Lập luận tự động: xử lý tri thức và đưa ra kết luận.
- Học máy: thích nghi với hoàn cảnh và khả năng ngoại suy.

3. Những đặc điểm của công nghệ xử lý thông tin dựa trên TTNT

Với những yêu cầu mới đối với những ứng dụng của TTNT, bản thân công nghệ xử lý thông tin phải có những đặc điểm sau:

Phải có những công cụ hình thức hóa như các mô hình logic ngôn ngữ, logic mờ, mạng ngữ nghĩa,…để biểu diễn tri thức trên máy tính và quá trình giải quyết bài toán được tiến hành hữu hiệu hơn.

Phải có tính mềm dẻo, thích nghi với những tình huống mới nảy sinh, chẳng hạn như các hệ chuyên gia. Các cơ chế suy diễn cũng phải mềm dẻo được áp dụng tùy tình huống, chưa chắc cơ chế nào tốt hơn cơ chế nào.

Phải được trang bị tri thức heuristic do chuyên gia con người cung cấp khác với chuơng trình thông thuờng chỉ cần dựa trên thuật toán là đủ.

Việc xây dựng các chương trình TTNT phải có sự tham gia của các kỹ sư xử lý tri thức: phân tích phương pháp giải quyết bài toán theo chuyên gia con người, diễn đạt tri thức và cơ chế suy diễn để dễ mã hóa trong máy tính.

Nhìn chung, xử lý thông tin dựa trên TTNT có những đặc điểm riêng khác với công nghệ truyền thống trước đây:

Lập trình truyền thống	**Lập trình theo TTNT**
Xử lý dữ liệu	Xử lý tri thức – dữ liệu thường mang tính định tính hơn là định lượng

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

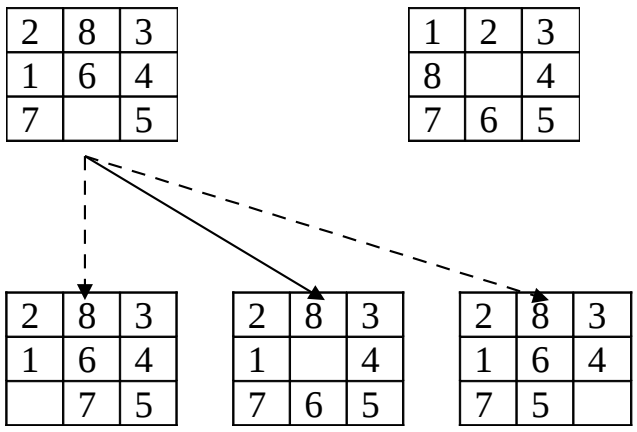
Dữ liệu được đánh địa chỉ	Tri thức được biểu diễn theo một số cấu trúc nhất định
Xử lý theo thuật toán	Xử lý theo các giải thuật heuricstics và cơ chế lập luận
Xử lý tuần tự, theo lô	Tính tương tác cao (robot)
Không cần giải thích việc thực hiện	Có thể giải thích lý do thực hiện

$2. CÁC CHIẾN LƯỢC TÌM KIẾM MÙ

4. Biểu diễn vấn đề trong không gian trạng thái.

Vấn đề là một bài toán nào đó cần giải quyết, chẳng hạn như một trò chơi, chứng minh một định lý,…Một lớp rộng các bài toán có thể giải quyết bằng cách biểu diễn bởi các trạng thái và các tóan tử (phép biến đổi trạng thái). Để dễ hình dung các khái niệm này, chúng ta tìm hiểu ví dụ sau.

Ví dụ 1. (Trò chơi 8 số). Cho hình vuông 3x3 và 8 số từ 1,..,8 xếp ngẫu nhiên vào các ô. Hãy di chuyển các số nhờ ô trống để thu được sự sắp xếp như hình bên phải



Sự sắp xếp các số tại mỗi thời điểm là một trạng thái. Hình bên trái là trạng thái ban đầu. Hình bên phải là trạng thái kết thúc hay trạng thái đích (goal). Trạng thái đích của một bài toán có thể nhiều hơn một trạng thái. Một toán tử là một phép biển đổi hợp lệ chuyển từ trạng thái này sang trạng thái khác. Gọi u là một trạng thái tùy ý, tập hợp tất cả các trạng thái mở rộng từ trạng thái u được ký hiệu là S(u). Từ các trạng thái được mở rộng từ trạng thái ban đầu, bằng các tóan tử, ta tiếp tục mở rộng và cuối cùng thu được một không gian trạng thái (KGTT).

Như vậy, một không gian trạng thái là một bộ bốn (X, u_0, F, G), trong đó:

+ X là tập các trạng thái

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

+ u_0 là trạng thái bắt đầu

+ F là tập các toán tử, gồm các phép biến đổi.

+ G là tập trạng thái đích.

Không gian trạng thái có thể biểu diễn bằng đồ thị có hướng: mỗi đỉnh là một trạng thái, mỗi cung là một toán tử. Giải quyết được vấn đề hay tìm được nghiệm của bài toán nếu như ta tìm được đường đi từ trạng thái bắt đầu đến một trong các trạng thái đích.

Trong đồ thị của KGTT có thể xuất hiện chu trình gây khó khăn cho việc tìm kiếm, hạn chế các toán tử trong F có thể đưa đồ thị trở thành cây, một cấu trúc dễ tìm kiếm hơn. KGTT của trò chơi caro là một trường hợp như vậy.

Việc biểu diễn các trạng thái hợp lý là rất quan trọng, nó có thể làm cho KGTT nhỏ lại cũng như việc tìm kiếm trở nên đơn giản hơn.

Khi tìm kiếm lời giải, từ một trạng thái nào đó chưa phải là trạng thái đích, ta dựa theo toán tử sinh ra tập các trạng thái mới. Quá trình này gọi là quá trình mở rộng không gian trạng thái. Để được lời giải, ta phải liên tục chọn trạng thái mới, mở rộng, kiểm tra cho đến khi tìm được trạng thái đích hoặc không mở rộng được không gian trạng thái. Thông thường, tập các trạng thái được mở rộng sẽ có nhiều phần tử, việc chọn trạng thái nào để tiếp tục mở rộng được gọi là chiến lược tìm kiếm. Việc tìm kiếm lời giải cho bài toán thường sinh ra một cây gọi là cây tìm kiếm. Trong đó, mỗi nút trên cây cần giữ những thông tin sau đây:

+ Nội dung trạng thái mà nút hiện hành đang biểu diễn.

+ Nút cha đã sinh ra nó.

+ Toán tử đã được sử dụng để sinh ra nút hiện hành.

+ Độ sâu của nút.

+ Giá trị đường đi từ nút gốc đến nút hiện hành.

Để triển khai một chiến lược tìm kiếm nào đó, thông thường chúng ta phải lưu trữ các trạng thái chờ phát triển. Hàng đợi (queue) là một cấu trúc dữ liệu được ưu tiên lựa chọn với những thao tác sau:

+ Tạo hàng đợi với một số phần tử: Make-Queue(elements)

+ Kiểm tra hàng đợi rỗng: Empty(Queue)?

+ Lấy một phần tử ở đầu hàng đợi: Remove-Front(Queue)

+ Chèn một số phần tử vào hàng đợi:Insert(elements, Queue)

5. Các chiến lược tìm kiếm mù

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

Khi tìm kiếm trạng thái đích việc lựa chọn trạng thái nào để mở rộng được gọi là chiến lược tìm kiếm. Một chiến lược tìm kiếm được đánh giá dựa theo các tiêu chí sau đây:

+ Tính đầy đủ: chiến lược phải đảm bảo tìm được lời giải nếu có.

+ Độ phức tạp thời gian: mất thời gian bao lâu để tìm được lời giải.

+ Độ phức tạp không gian: tốn bao nhiêu đơn vị bộ nhớ để tìm được lời giải.

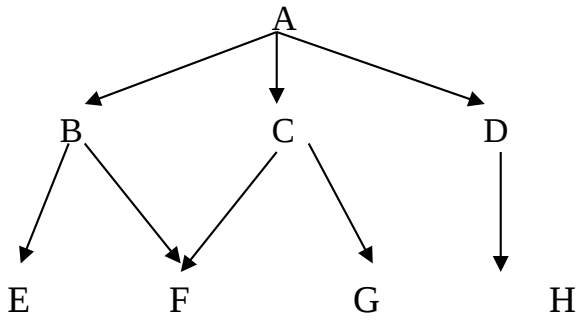
+ Tính tối ưu: tốt hơn so với một số chiến lược khác hay không.

Trong tìm kiếm mù, trạng thái được chọn để phát triển chỉ đơn thuần dựa theo cấu trúc của KGTT mà không có thông tin hướng dẫn nào khác. Cho nên, nói chung tìm kiếm mù sẽ không hiệu quả. Tuy nhiên, nó được quan tâm bởi vì đây là cơ sở để chúng ta cải tiến và thu được những chiến lược hiệu quả hơn.

Tìm kiếm theo bề rộng (BFS)

Trạng thái được ưu tiên phát triển là trạng thái được sinh ra trước. Dùng danh sách open chứa các trạng thái sinh ra đang chờ phát triển, danh sách closed chứa các trạng thái đã được khảo sát.

Ví dụ. Không gian trạng thái:….(vẽ hình)



Thuật toán:

```
procedure bfs;
begin
    open:=[start]; closed:=[];
    while open<>[] do
        begin
            loại trạng thái ngoài cùng bên trái của open, gọi nó là u
            if (u là một đích) then thông báo kết quả, thoát
            else begin
                Đưa u vào closed
```

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

Phát sinh các con v của u

Loại các con vừa phát sinh nhưng đã có mặt trong open hoặc closed

Đưa các con còn lại vào bên phải open (1)

end

end

Thông báo thất bại

End

Các trạng thái con phát sinh nhờ các toán tử hợp lệ. Danh sách open bổ sung phần tử bên phải, lấy phần tử bên trái. Thuật tóan khảo sát tất cả các nút ở độ sâu d sau đó mới đến mức d+1 nên chắc chắn tìm được nghiệm nếu có. Trong trường hợp bài tóan vô nghiệm và không gian trạng thái hữu hạn thuật toán sẽ dừng và thông báo vô nghiệm.

Để lưu được nghiệm, cần giữ nút cha của các nút được khảo sát.

Đánh giá: Giả sử mỗi trạng thái trung bình sinh ra b trạng thái con (kề), b được gọi là nhân tố nhánh. Giả sử đường đi nghiệm có độ dài d. Tình trạng xấu nhất phải khảo sát

$$1 + b + b^2 + \cdots + b^d$$

Như vậy độ phức tạp thời gian là $O(b^d)$, độ phức tạp không gian cũng là $O(b^d)$.

Giả sử mỗi giây máy tính xử lý được 1000 trạng thái, và mỗi trạng thái cần 100 bytes để lưu trữ, thời gian và bộ nhớ cần thiết cho tìm kiếm rộng được tính trong bảng sau

Độ sâu	Số nút	Thời gian	Bộ nhớ
2	111	0.1giây	11 kilobytes
4	11,111	11 giây	1 megabyte
6	10^6	18 phút	111 megabytes
8	10^8	31 giờ	11 gigabytes
10	10^{10}	128 ngày	1 terabyte
12	10^{12}	35 năm	111 terabyte
14	10^{14}	3500 năm	11.111 terabyte

Bảng 1. Thời gian, bộ nhớ cần thiết cho tìm kiếm theo bề rộng với b=10.

Tìm kiếm theo chiều sâu (DFS)

Chiến lược này mở rộng nút có độ sâu hơn trước các nút khác đang chờ xử lý. Chỉ khi nào không mở rộng được nữa thì mới quay lại nút ở độ sâu thấp hơn. Do đó, các nút mới được sinh ra chờ xử lý phải được bỏ bên trái của hàng đợi open (tại câu lệnh 1).

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

DFS sử dụng bộ nhớ dùng cho open để lưu trữ ít hơn. Nếu nhân tố nhánh là b và chiến lược đang khảo sát nút ở độ sâu d thì nó chỉ lưu trữ b*d nút, trong khi đó BFS phải lưu trữ b^d nút. Ở độ sâu d=12, tính toán cho thấy DFS chỉ sử dụng 12KB trong khi BFS dùng đến 111TB. Độ phức tạp thời gian của DFS vẫn là $O(b^d)$ vì trong trường hợp xấu nhất các nút được khảo sát vẫn như BFS. Tuy nhiên, cơ hội để tìm thấy trạng thái đích cao hơn nếu nó nằm ở phần không gian trạng thái bên trái.

DFS có những hạn chế của nó. Nó bỏ qua cơ hội tìm thấy ngay trạng thái đích khi trạng thái này nằm gần gốc. Trường hợp không gian trạng thái được mở rộng vô hạn có thể nó không tìm được trạng thái đích. Hơn nữa trong trường hợp không gian trạng thái là đồ thị đường đi nghiệm nói chung không phải là đường đi ngắn nhất. Vì vậy, DFS bị đánh giá là chiến lược không đầy đủ mà cũng không tối ưu. Do đó nó không được khuyến khích sử dụng khi KGTT có độ sâu lớn hoặc vô hạn. Hai giải pháp sau đây có thể giải quyết phần nào cho tình huống này.

Tìm kiếm với độ sâu hạn chế

Vẫn như sử dụng chiến lược tìm kiếm theo chiều sâu nhưng giới hạn độ sâu của đường đi nghiệm trên cây, tức là chúng ta sẽ không tiếp tục mở rộng nếu đã đến một độ sâu d cố định nào đó. Số d được gọi là bán kính trên KGTT. Chiến lược này sẽ tìm được nghiệm hay không phụ thuộc vào d. Nếu d được chọn thích hợp thì nó tìm được nghiệm, khi đó chiến lược là đầy đủ. Tuy nhiên nó không là chiến lược tối ưu. Tương tự như DFS, độ phức tạp thời gian là $O(b^d)$ và độ phức tạp không gian là O(bd).

Thuật toán cho chiến lược này có thêm tham số d

```
procedure DFS(d);
begin
    open:=[start]; closed:=[]; depth(start):=0;
    while open<>[] do
        begin
            loại trạng thái ngoài cùng bên trái của open, gọi nó là u
            if (u là một đích) then thông báo kết quả, thoát
            else begin
                Đưa u vào closed
                If  depth(u)<=d then begin
                    Phát sinh các con v của u
                    Loại các con vừa phát sinh nhưng đã có mặt trong open hoặc closed
                    Gán độ sâu cho các v bằng depth(u)+1
                    Đưa các con v còn lại vào bên trái open
                End;
        End;
```

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

```
        End;
    Thông báo thất bại
End.
```

Vấn đề khó khăn là xác định độ sâu hạn chế d. Hầu hết các bài tóan chúng ta không biết trước d bằng bao nhiêu. Chiến lược sau đây giải quyết vấn đề này.

Tìm kiếm sâu dần

Từng bước tăng dần độ sâu hạn chế là cách làm trong chiến lược này. Lần lượt cho d bằng 1,2,…và tìm kiếm với độ sâu hạn chế ứng với d. Cách làm này tận dụng được lợi thế của hai chiến lược tìm kiếm rộng và tìm kiếm sâu ở chỗ vì nó tìm hết cây độ sâu d đến d+1 nên giống như tìm kiếm rộng chắc chắn tìm được nghiệm và do sử dụng tìm kiếm sâu trong cây hạn chế nên ít tốn kém bộ nhớ hơn. Như vậy, chiến lược này là đầy đủ và nếu KGTT là cây thì nó là chiến lược tối ưu. Tìm kiếm sâu dần sử dụng tìm kiếm sâu hạn chế như một thủ tục con

```
procedure DFS1;
begin
    for d:=0 to max do
        begin
            DFS(d);
            If thành công then exit
        End;
End;
```

Qúa trình tìm kiếm sâu sẽ lặp lại việc khảo sát các nút trong lớp có độ sâu bé hơn. Tưởng chừng điều này sẽ làm tăng độ phức tạp thời gian và không gian, tuy nhiên không như vậy. Chúng ta xét ví dụ sau đây. Giả sử nhân tố nhánh b=10 và d=max=5. Nhớ lại rằng số nút phải khảo sát trong tìm kiếm sâu với độ sâu d=5 là

$$1+b+b^2+\cdots+b^d = 1+10+100+1.000+10.000+100.000=111.111$$

Khi tìm kiếm sâu dần, do vòng lặp For thực hiện max+1 lần nên số nút phải khảo sát là: $(d+1)1 + d*b + (d-1)*b^2 + \ldots + 2*b^{d-1} + 1*b^d = 6 + 50 + 400 + 3000 + 20.000 + 100.000 = 123.456$. Số này phụ thuộc b nhưng nói chung không vượt quá 2 nhưng tính đầy đủ của chiến lược bảo đảm như tìm kiếm rộng. Thực tế số nút tăng lên nhưng độ phức tạp của thuật tóan vẫn là $O(b^d)$, trong khi đó độ phức tạp không gian vẫn là O(bd). Vì vậy, tìm kiếm sâu dần được đánh giá là chiến lược tìm kiếm thích hợp hơn khi KGTT lớn và không biết trước được độ sâu của trạng thái đích.

Tóm lại:….

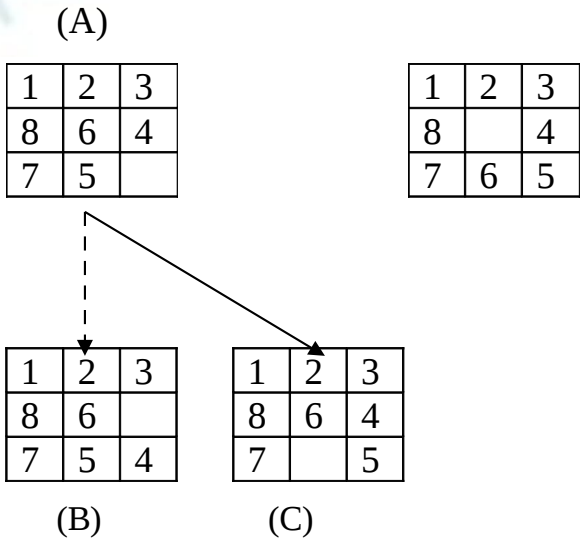
$3. CÁC CHIẾN LƯỢC TÌM KIẾM KINH NGHIỆM

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

Các chiến lược tìm kiếm mù kém hiệu quả và không thể áp dụng được trong nhiều trường hợp. Sử dụng thông tin của trạng thái kết hợp với nhận xét dựa theo kinh nghiệm để cải tiến cách lựa chọn để phát triển trạng thái là quan điểm chung của các chiến lược tìm kiếm kinh nghiệm.

1. Hàm đánh giá

Xét một số trạng thái của bài toán trò chơi 8 số. Nếu dùng các chiến lược tìm kiếm mù thì trạng thái (B) sẽ được lựa chọn để phát triển. Dễ thấy việc lựa chọn này là không hiệu quả, việc tìm đến đích sẽ lâu hơn là phát triển trạng thái (C). Nói cách khác, trạng thái (C) được phát triển có nhiều khả năng dẫn đến đích hơn trạng thái (B). Điều này phải được lượng hóa. Hàm đánh giá là một hàm ước lượng khả năng về đích của mỗi trạng thái. Để ý rằng ở đây chỉ là ước lượng vì nói chung chúng ta không thể tính toán chính xác khả năng này, bởi vì nếu tính được nghĩa là đã tìm được lời giải. Tuy nhiên, nhờ kinh nghiệm trong nhiều bài toán cụ thể chúng ta có thể ước lượng được giá trị này.



Ví dụ nếu đếm các số sai vị trí của một trạng thái so với trạng thái đích thì số này đối với (B) là 3 và đối với (C) là 1. Như vậy, số này càng nhỏ thì trạng thái đó càng có khả năng về đích và chúng ta có thể dùng số này làm giá trị cho hàm đánh giá. Một cách tổng quát, hàm đánh giá là một hàm h như sau:

$$h: X \to R^+,$$

trong đó X là không gian trạng thái của bài toán. Dễ thấy, nếu g là trạng thái đích thì $h(g)=0$. Mỗi $u \in X$, ta có một giá trị ước lượng là $h(u) \geq 0$. Hàm đánh giá còn được gọi là hàm heuristic.

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

Trong ví dụ trên, hàm h có thể xác định bởi $h(u) =$ số các số sai vị trí đối với đích

Vì được xây dựng theo kinh nghiệm nên có nhiều cách để xây dựng hàm h.

1	5	3
8		4
7	6	2

(D)

1	4	3
8		2
7	6	5

(E)

Nếu tính như trên thì $h(D) = h(E) = 2$ vì đều có 2 số sai vị trí nhưng xét về mặt đường đi qua các ô ngang dọc để di chuyển về đúng vị trí thì (D) phải di chuyển đoạn đường xa hơn (E) nên có thể là (D) không tốt bằng (E). Theo nhận xét này, ta có thể xây dựng hàm

$h_1(u) =$ tổng khoảng cách phải di chuyển của các ô sai vị trí.

Ví dụ: $h_1(D) = 3+3 = 6,\ h_1(E) = 2+2=4$

Một ví dụ khác là lấy độ dài đường chim bay làm giá trị cho hàm đánh giá….

Giải quyết vấn đề bằng chiến lược tìm kiếm kinh nghiệm cần thực hiện các công việc sau:

+ Biểu diễn bài toán bằng một KGTT thích hợp

+ Xây dựng hàm đánh giá

+ Thiết kế chiến lược chọn trạng thái

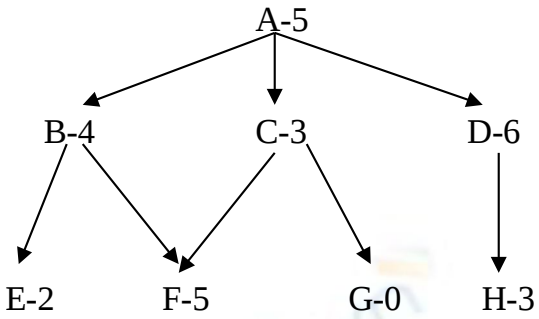
2. Các chiến lược tìm kiếm kinh nghiệm.

a) Tìm kiếm leo đồi – Hill Climbing Search (Pearl, 1984)

Chiến lược này chọn một trạng thái tốt hơn trạng thái đang khảo sát để phát triển. Trường hợp không tìm được trạng thái tốt hơn, thuật tóan phải dừng. Nếu chỉ chọn một trạng thái tốt hơn nào đó thì gọi là chiến lược leo đồi đơn giản, nếu trong số các trạng thái tốt hơn chọn trạng thái tốt nhất thì gọi là leo đồi dốc đứng. Nhắc lại rằng, chúng ta sử dụng hàm đánh giá để biết trạng thái nào tốt hơn. Khác với tìm kiếm theo chiều sâu tìm kiếm leo đồi không lưu tất cả các trạng thái con được sinh ra mà chỉ lưu đúng một trạng thái được chọn nếu có.

Ví dụ. KGTT của bài toán như sau

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang



……

Tìm kiếm leo đồi chỉ hiệu quả khi có một hàm đánh giá tốt. Khi hàm đánh giá không tốt, việc tìm kiếm có thể rơi vào bế tắc trong những trường hợp sau đây:

- Phát triển đến được nút tốt nhất trong khu vực nhưng không phải là trạng thái đích. Nút như vậy được gọi là điểm cực đại địa phương. Điều này xảy ra do thuật giải leo đồi chỉ phát triển nếu tìm được trạng thái con tốt hơn. Khi không tìm được trạng thái như vậy, việc tìm kiếm dừng lại. Một cách hình ảnh, việc tìm kiếm không vượt được qua khu vực trũng để đến khu vực chứa trạng thái đích.

- Các trạng thái con được sinh ra có độ tốt ngang bằng với trạng thái đang xét. Điều này cũng làm cho việc tìm kiếm dừng lại. Nói cách khác, khu vực bằng phẳng cũng là một trở ngại trong chiến lược tìm kiếm này.

Cũng có một vài giải pháp như xáo trộn ngẫu nghiên các trạng thái chờ đợi trong open để khắc phục các tình trạng trên. Tuy nhiên, không có một giải pháp tổng quát cho vấn đề này. Lý do xảy ra các tình huống thất bại của giải thuật leo đồi là do giải thuật này quá cực đoan, nó không lưu lại các trạng thái anh em nên không thể quay lại các trạng thái trước đó khi cần thiết. Khắc phục hạn chế này là chiến lược sau đây.

b) Tìm kiếm ưu tiên tốt nhất – Best First Search (Best-FS)

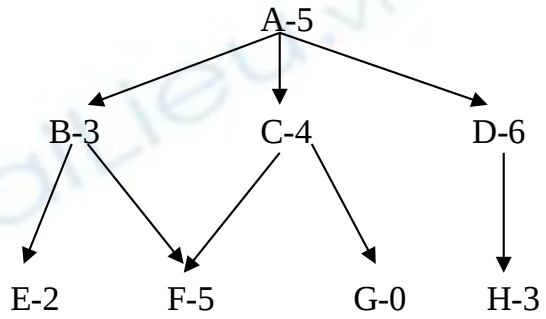
Chiến lược này chọn trạng thái tốt nhất trong các trạng thái đang chờ trong open để phát triển, kể cả trạng thái này không tốt bằng trạng thái đã sinh ra nó. Điều này giải thích vì sao nó có thể đi xuống vùng thấp trong khi giải thuật leo đồi thì không xuống được. Một điều nữa là nó lưu trữ tất cả các trạng thái anh em khi phát triển trạng thái mới vì vậy khi đi vào ngõ cụt, chiến lược này có thể lui ra được mà không bị bế tắc. So với một số chiến lược khác chúng ta có nhận xét rằng, giải thuật Best-FS kết hợp hai phương pháp tìm kiếm theo chiều sâu và chiều rộng nhằm tận dụng ưu thế của cả hai. Tìm kiếm theo chiều sâu có ưu thế là không phải quan tâm đến sự mở rộng của tất cả các nhánh, trong khi đó tìm kiếm theo chiều rộng lại không bị sa vào các nhánh bế tắc. Best-FS "cẩn thận" hơn ở chỗ khi di chuyển theo một hướng khả quan nó vẫn ghi nhớ

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

lại các một số trạng thái không tốt hơn trước đó để còn có thể quay lại nếu như những bước tiếp theo của hướng đang đi không còn khả quan như trước.

Cụ thể hơn, Best-FS sẽ lựa chọn trạng thái tiếp theo là trạng thái có khả năng tốt nhất trong các trạng thái đã gặp, khác với tìm kiếm leo đồi dốc đứng chọn trạng thái tốt nhất trong số các trạng thái kế tiếp có thể từ trạng thái hiện tại. Như vậy, Best-FS sẽ linh hoạt hơn, nó kịp thời di chuyển sang một nhánh khác khả quan hơn, không tiếp tục đi vào nhánh cụt.

Để làm điều này Best-FS cải tiến so với leo đồi ở chỗ: thứ nhất là nó giữ lại trong open tất cả các con của trạng thái đang xét – để có thể sẽ quay lại nếu bế tắc trên một hướng đi khác. Thứ hai là trong số các trạng thái đang xét nó sẽ chọn trạng thái tốt nhất, vì vậy phải sắp xếp open để chuẩn bị cho việc chọn phần tử này.



Ví dụ: Không gian trạng thái như hình 2.

Bước 1	Open	Close (bổ sung mỗi thời điểm)
1	A-5	
2	B-3, C-4, D-6	A-5
3	E-2, C-4, F-5,D-6	B-3
4	C-4, F-5,D-6	E-2
5	G-0 ,F-5,D-6	C-4

Giải thuật được mô tả như sau.

```
procedure Best_FS;
begin
    open:={u0}; closed:={ };
    while open<>{ } do
        begin
            loại trạng thái ngoài cùng bên trái của open, gọi nó là u
            if (u là một đích) then thông báo thắng lợi, thoát
            else begin
                Đưa u vào closed
                Phát sinh các con v của u
                Loại các con  v đã có mặt trong open + closed
                Đưa các con còn lại vào open
                Sắp xếp open sao cho phần tử tốt nhất nằm bên trái
                end
```

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

```
        end
    Thông báo thất bại
end
```

Một cách đầy đủ, để có được kết quả là đường đi nghiệm chúng ta phải lưu ý thêm về việc lưu giữ các trạng thái cha để truy lại vết của đường đi này.

$3. CÁC CHIẾN LƯỢC TÌM KIẾM TỐI ƯU

Trong các chiến lược tìm kiếm trên, chúng ta chỉ quan tâm đến việc làm thế nào để tìm được trạng thái đích mà không quan tâm đến độ dài hay chi phí của đường đi nghiệm. Trong thực tế, đây là một yếu tố quan trọng vì giá trị hay ý nghĩa của nghiệm tìm được phụ thuộc vào yếu tố này. Ví dụ như số bước đi của một người chơi cờ. Cũng tìm được trạng thái đích nhưng chỉ cần ít hơn đối thủ một bước thì bạn đã thắng. Chi phí một hành trình trong bài toán người du lịch cũng là một ví dụ cho yếu tố này. Vì vậy chúng ta không chỉ quan tâm đến việc tìm ra nghiệm mà còn phải quan tâm đến việc nghiệm đó có tối ưu hay không.

KGTT trong bài toán tối ưu phải có thêm trọng số của các cung. Chúng ta đã phân tích các chiến lược từ tìm kiếm mù, tìm kiếm kinh nghiệm và biết được những ưu nhược điểm của từng chiến lược. Tìm kiếm Best-FS là chiến lược tìm kiếm linh hoạt nhất cho đến lúc này. Chúng ta sẽ tiếp tục cải tiến chiến lược này để giải quyết vấn đề tối ưu.

1. Hàm đánh giá cải tiến.

Trong Best-FS một trạng thái u được ưu tiên phát triển nếu như nó có giá trị h(u) nhỏ nhất trong các trạng thái trong open. Tuy nhiên, chỉ riêng h(u) là không đủ khi xét đường đi tối ưu. Với quan điểm tối ưu, đồng thời với h(u) nhỏ để mau về đích thì độ dài đường đi từ trạng thái xuất phát đến u cũng phải nhỏ. Vì vậy, người ta dùng hàm đánh giá cải tiến sau:

$$f(u) = g(u) + h(u)$$

trong đó g(u) là độ dài đường đi ngắn nhất từ u_0 đến u, h(u) là đánh giá độ tốt theo hàm heuristic. Hàm h(u) được gọi là chấp nhận được hay hàm đánh giá thấp nếu như h(u) nhỏ hơn độ dài đường đi ngắn nhất thực sự từ u đến đích.

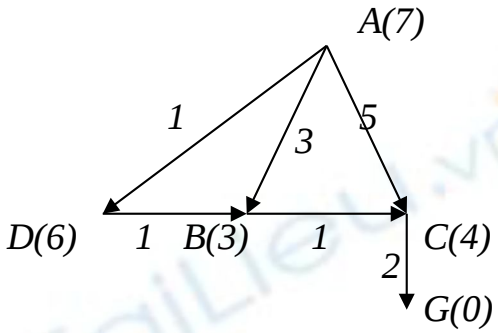
3. Giải thuật A^{KT} (Algorithm for Knowlegeable Tree Search)

4. Giải thuật A*

A* là giải thuật tổng quát hơn A^{KT}, nó tìm kiếm trên KGTT là đồ thị. Vì là đồ thị nên phát sinh nhiều vấn đề khi tìm đường đi tối ưu. Để ý rằng nghiệm là đường đi nên ta phải lưu lại vết của đường đi này. Trong các giải thuật trước, để tập trung cho tư tưởng chính của các giải thuật đó chúng ta bỏ qua chi tiết này, nhưng trong giải thuật này chi tiết này được đề cập vì nó liên quan đến nghiệm một cách trực tiếp. Trạng thái u tùy ý sẽ gồm bốn yếu tố (g(u), h(u), f(u), cha(u)). Trong đó,...

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

Bây giờ ta sẽ mô tả cách làm việc của A*. Giả sử KGTT được cho trong hình ...



Các trạng thái sẽ được lưu giữ trong open và closed trong các bước như sau

Bước	Open	closed
1	A(0,7,0,-)	
2	B(3,3,6,A), D(1,6,7,A), C(5,4,9,A)	A(0,7,0,-)
3	D(1,6,7,A), C(4,4,8,B)	~~B(3,3,6,A)~~
4	B(2,3,5,D), C(4,4,8,B)	D(1,6,7,A)
5	C(3,4,7,B)	B(2,3,5,D)
6	G(5,0,5,C)	C(3,4,7,B)

Ở bước 2, mọi việc xảy ra bình thường như với giải thuật A^{KT}

Ở bước 3, tìm được đường đi đến C qua B ngắn hơn nên các giá trị của C trong open phải được sửa đổi.

Ở bước 4, mặc dù B đã nằm trong closed, tức đã xét xong nhưng đường đi mới qua D đến B ngắn hơn nên B phải được lấy khỏi closed chuyển qua open chờ xét lại với giá trị mới.

Ở bước 5, lại xảy ra việc chỉnh sửa các giá trị của C như ở bước 3.

Giải thuật dừng ở bước 6 và đường đi thu được độ dài 5 như sau A-D-B-C-G.

Như vậy, khác với các chi tiết đã trình bày trong các giải thuật trước đây, trạng thái đã lưu trong open vẫn phải thay đổi giá trị, hoặc thậm chí lưu trong closed cũng phải bị xóa đi và chuyển qua open với các giá trị mới.

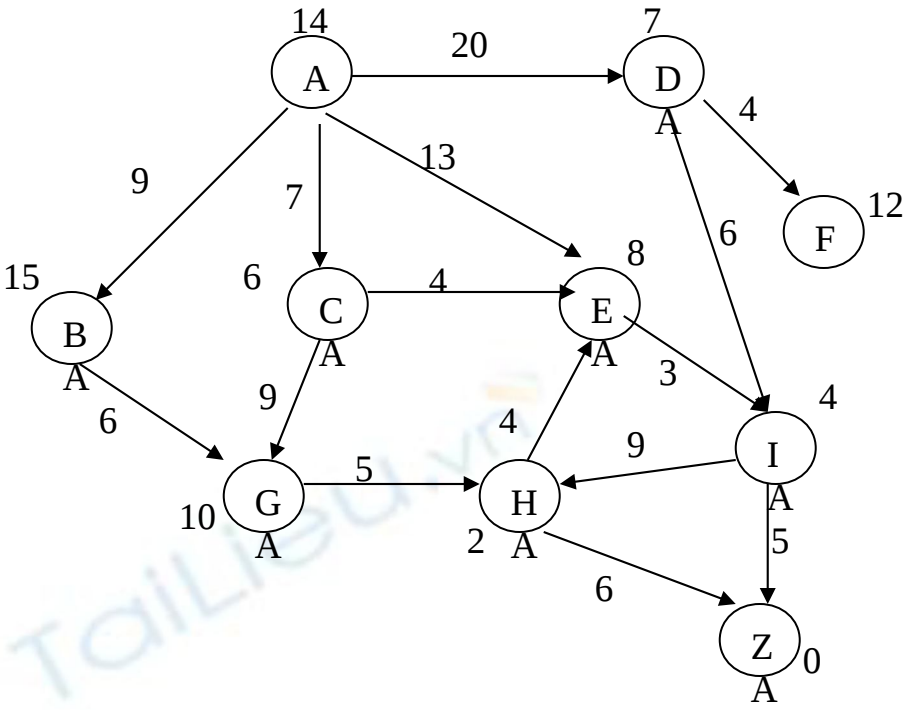
Tóm lại, giải thuật được trình bày dưới dạng giả mã sau:

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang

```
procedure A*(uo);
begin
    g(uo)=0; f(uo)=g(uo);
    open:=[uo]; closed:=[];
    while open<>[] do
        begin
            loại trạng thái ngoài cùng bên trái của open, gọi nó là u
            Đưa u vào closed;
            if (u là một đích) then thông báo thắng lợi, thoát
            else begin
                Sinh các con v của u;
                For v thuộc con(u) do
                    begin
                        g(v):=g(u)+c[u,v];
                        If v không thuộc open hay closed
                            begin
                            f(v):=g(v)+h(v);
                            cha(v):=u;
                            Bỏ v  vào open;
                        end
                        If v thuộc open (tồn tại v' thuộc open, sao cho v=v')
                            If g(v)<g(v') then
                                Begin
                                    g(v'):=g(v);
                                        f(v'):=g(v')+h(v');
                                    Cha(v'):=u;
                                End;
                        If v thuộc closed (tồn tại v' thuộc closed, sao cho v=v')
                            If g(v)<g(v') then
                                Begin
                                        f(v):=g(v)+h(v);
                                    cha(v):=u;
                                    Đưa v vào open;
                                        Loại v' khỏi closed;
                                End;
                    End;{for}
                Xếp open để trạng thái tốt nhất nằm bên trái;
            End{else}
    End;{while}
    Thông báo thất bại
End;{procedure}
```

Ví dụ. Hoạt động của giải thuật với KGTT là bản đồ giao thông...

Tài liệu chỉ xem được một số trang đầu. Vui lòng download file gốc để xem toàn bộ các trang



Kết quả.

Một hàm đánh giá h(u) được gọi là chấp nhận được hay là hàm đánh giá thấp nếu như h(u)<=h*(u) với moi u, ở đây h*(u) là đường đi ngắn nhất từ u đến đích.

Nếu hàm đánh giá h(u) là chấp nhận được thì thuật toán A* là tối ưu.

Thật vậy, giả sử G là một trạng thái đích mà thuật toán tìm được và có độ dài đường đi tử uo đến G là g(G). Do G là trạng thái đích, h(G)=0 nên f(G)=g(G).

Giả sử nghiệm tối ưu là một đường đi khác từ uo kết thúc tại G1 có độ dài l. Nếu G1 nằm trong cây thì do G được chọn trước nên f(G)<=f(G1), suy ra g(G)<=g(G1)=l. Trường hợp G1 nằm ngoài cây tìm kiếm. Giả sử I là nút lá mà đường đi tối ưu thóat ra khỏi cây tìm kiếm. Vì hàm đánh giá thấp nên f(I) = g(I)+ h(I) <=g(G1). Do G được chọn trước I nên f(G)<=f(I)<=g(G1), do đó g(G)<=g(G1). Vậy G mới là nghiệm tối ưu.

A* là thuật tóan hiệu quả nhất trong các thuật toán đầy đủ và tối ưu cho bài toán tìm đường đi ngắn nhất.