Thuật Toán Giải

CHƯƠNG 1 : THUẬT TOÁN – THUẬT GIẢI

I. TỔNG QUAN THUẬT TOÁN – THUẬT GIẢI

Trong quá trình nghiên cứu giải quyết các vấn đề – bài toán, người ta đã đưa ra những nhận xét như sau:

- Có nhiều bài toán cho đến nay vẫn chưa tìm ra một cách giải theo kiểu thuật toán và cũng không biết là có tồn tại thuật toán hay không.
- Có nhiều bài toán đã có thuật toán để giải nhưng không chấp nhận được vì thời gian giải theo thuật toán đó quá lớn hoặc các điều kiện cho thuật toán khó đáp ứng.
- Có những bài toán được giải theo những cách giải vi phạm thuật toán nhưng vẫn chấp nhận được.

Từ những nhận định trên, người ta thấy rằng cần phải có những đổi mới cho khái niệm thuật toán. Người ta đã mở rộng hai tiêu chuẩn của thuật toán: tính xác định và tính đúng đắn. Việc mở rộng tính xác định đối với thuật toán đã được thể hiện qua

các giải thuật đệ quy và ngẫu nhiên. Tính đúng của thuật toán bây giờ không còn bắt buộc đối với một số cách giải bài toán, nhất là các cách giải gần đúng. Trong thực tiễn có nhiều trường hợp người ta chấp nhận các cách giải thường cho kết quả tốt (nhưng không phải lúc nào cũng tốt) nhưng ít phức tạp và hiệu quả. Chẳng hạn nếu giải một bài toán bằng thuật toán tối ưu đòi hỏi máy tính thực hiên nhiều năm thì chúng ta có thể sẵn lòng chấp nhận một giải pháp gần tối ưu mà chỉ cần máy tính chạy trong vài ngày hoặc vài giờ.

Các cách giải chấp nhận được nhưng không hoàn toàn đáp ứng đầy đủ các tiêu chuẩn của thuật toán thường được gọi là các thuật giải. Khái niệm mở rộng này của thuật toán đã mở cửa cho chúng ta trong việc tìm kiếm phương pháp để giải quyết các bài toán được đặt ra.

Một trong những thuật giải thường được đề cập đến và sử dụng trong khoa học trí tuệ nhân tạo là các cách giải theo kiểu Heuristic

II. THUẬT GIẢI HEURISTIC

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau:

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

Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ bản như sau:

- **Nguyên lý vét cạn thông minh:** Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.
- **Nguyên lý tham lam (Greedy):** Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.
- **Nguyên lý thứ tự:** Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.
- **Hàm Heuristic:** Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các hàm Heuristic. Đó là các hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.

Bài toán hành trình ngắn nhất – ứng dụng nguyên lý Greedy

Bài toán: Hãy tìm một hành trình cho một người giao hàng đi qua n điểm khác nhau, mỗi điểm đi qua một lần và trở về điểm xuất phát sao cho tổng chiều dài đoạn đường cần đi là ngắn nhất. Giả sử rằng có con đường nối trực tiếp từ giữa hai điểm bất kỳ.

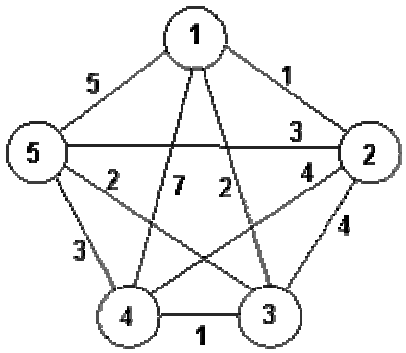
Tất nhiên ta có thể giải bài toán này bằng cách liệt kê tất cả con đường có thể đi, tính chiều dài của mỗi con đường đó rồi tìm con đường có chiều dài ngắn nhất. Tuy nhiên, cách giải này lại có độ phức tạp 0(n!) (một hành trình là một *hoán vị* của n điểm, do đó, tổng số hành trình là số lượng hoán vị của một tập n phần tử là n!). Do đó, khi số đại lý tăng thì số con đường phải xét sẽ tăng lên rất nhanh.

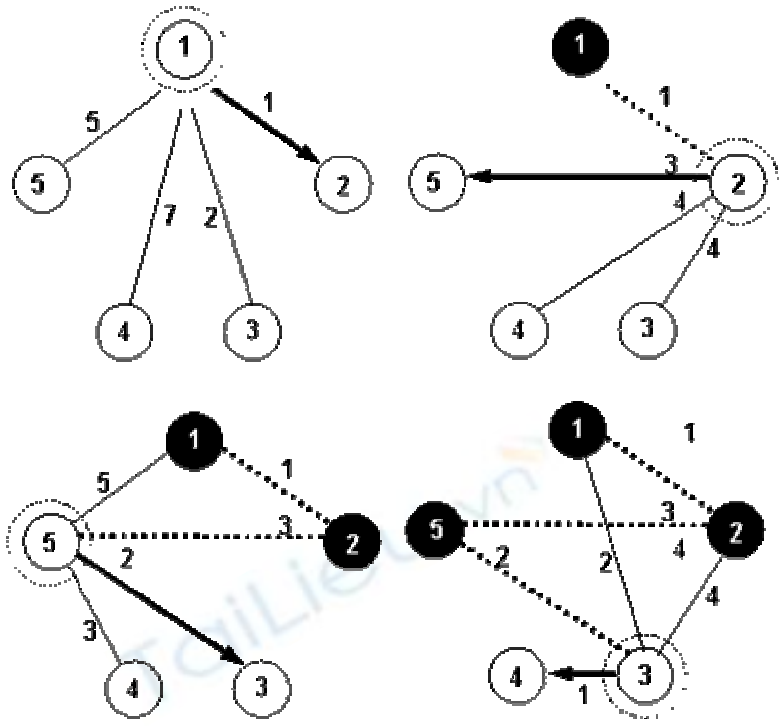
Một cách giải đơn giản hơn nhiều và thường cho kết quả tương đối tốt là dùng một thuật giải Heuristic ứng dụng nguyên lý Greedy. Tư tưởng của thuật giải như sau:

- Từ điểm khởi đầu, ta liệt kê tất cả quãng đường từ điểm xuất phát cho đến n đại lý rồi chọn đi theo con đường ngắn nhất.

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

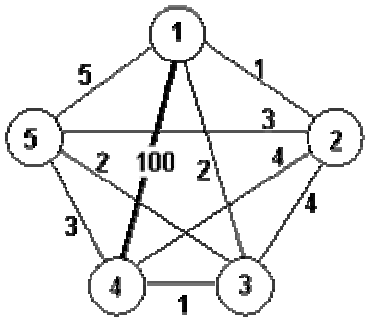
Bạn có thể quan sát hình sau để thấy được quá trình chọn lựa. Theo nguyên lý Greedy, ta lấy tiêu chuẩn hành trình ngắn nhất của bài toán làm tiêu chuẩn cho chọn lựa cục bộ. *Ta hy vọng rằng, khi đi trên n đoạn đường ngắn nhất thì cuối cùng ta sẽ có một hành trình ngắn nhất.* Điều này không phải lúc nào cũng đúng. Với điều kiện trong hình tiếp theo thì thuật giải cho chúng ta một hành trình có chiều dài là 14 trong khi hành trình tối ưu là 13. Kết quả của thuật giải Heuristic trong trường hợp này chỉ lệch 1 đơn vị so với kết quả tối ưu. Trong khi đó, độ phức tạp của thuật giải Heuristic này chỉ là $0(n^2)$.





Hình : Giải bài toán sử dụng nguyên lý Greedy

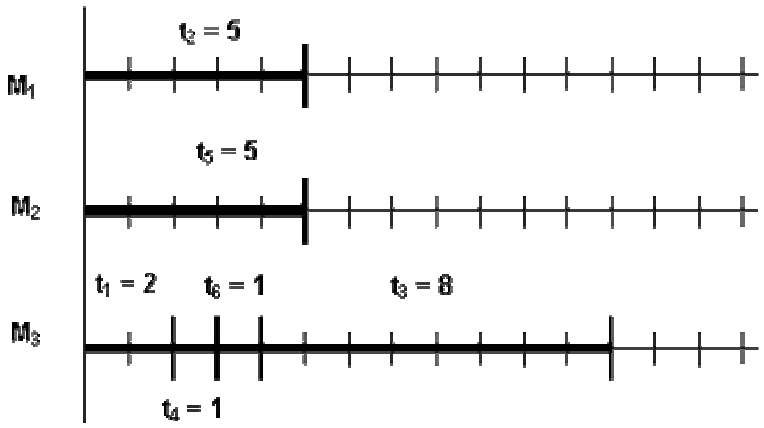
Tất nhiên, thuật giải theo kiểu Heuristic đôi lúc lại đưa ra kết quả không tốt, thậm chí rất tệ như trường hợp ở hình sau.



Bài toán phân việc – ứng dụng của nguyên lý thứ tự

Một công ty nhận được hợp đồng gia công m chi tiết máy J_1, J_2, … Jm. Công ty có n máy gia công lần lượt là P_1, P_2, … Pn. Mọi chi tiết đều có thể được gia công trên bất kỳ máy nào. Một khi đã gia công một chi tiết trên một máy, công việ sẽ tiếp tục cho đến lúc hoàn thành, không thể bị cắt ngang. Để gia công một việc J_1 trên một máy bất kỳ ta cần dùng một thời gian tương ứng là t_1. Nhiệm vụ của công ty là phải làm sao gia công xong toàn bộ n chi tiết trong thời gian sớm nhất.

Chúng ta xét bài toán trong trường hợp có 3 máy P_1, P_2, P_3 và 6 công việc với thời gian là t_1=2, t_2=5, t_3=8, t_4=1, t_5=5, t_6=1. ta có một phương án phân công (L) như hình sau:

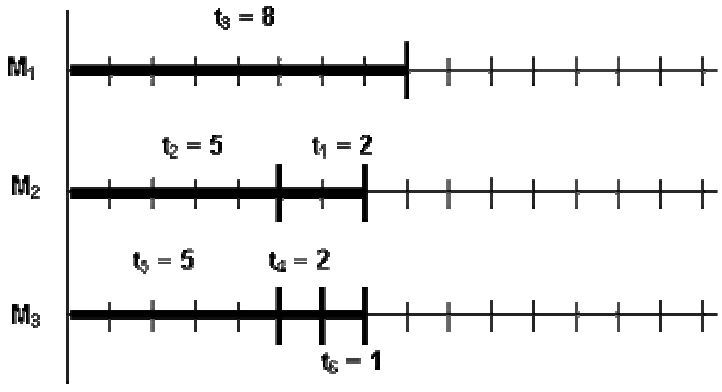


Theo hình này, tại thời điểm t=0, ta tiến hành gia công chi tiết J_2 trên máy P_1, J_5 trên P_2 và J_1 tại P_3. Tại thời điểm t=2, công việc J_1 được hoàn thành, trên máy P_3 ta gia công tiếp chi tiết J_4. Trong lúc đó, hai máy P_1 và P2 vẫn đang thực hiện công việc đầu tiên mình … Sơ đồ phân việc theo hình ở trên được gọi là lược đồ GANTT. Theo lược đồ này, ta thấy thời gian để hoàn thành toàn bộ 6 công việc là 12. Nhận xét một cách cảm tính ta thấy rằng phương án (L) vừa thực hiện là một phương án không tốt. Các máy P_1 và P_2 có quá nhiều thời gian rãnh.

Thuật toán tìm phương án tối ưu L_0 cho bài toán này theo kiểu vét cạn có độ phức tạp cỡ O(mn) (với m là số máy và n là số công việc). Bây giờ ta xét đến một thuật giải Heuristic rất đơn giản (độ phức tạp O(n)) để giải bài toán này.

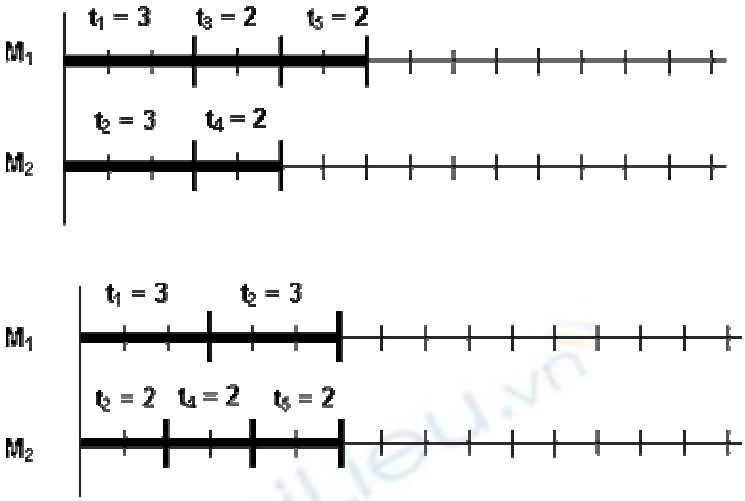
- Sắp xếp các công việc theo thứ tự giảm dần về thời gian gia công.
- Lần lượt sắp xếp các việc theo thứ tự đó vào máy còn dư nhiều thời gian nhất.

Với tư tưởng như vậy, ta sẽ có một phương án L* như sau:



Rõ ràng phương án L* vừa thực hiện cũng chính là phương án tối ưu của trường hợp này vì thời gian hoàn thành là 8, đúng bằng thời gian của công việc J_3. Ta hy vọng rằng một giải Heuristic đơn giản như vậy sẽ là một thuật giải tối ưu. Nhưng tiếc thay,

ta dễ dàng đưa ra được một trường hợp mà thuật giải Heuristic không đưa ra được kết quả tối ưu.



Nếu gọi T* là thời gian để gia công xong n chi tiết máy do thuật giải Heuristic đưa ra và T^0 là thời gian tối ưu thì người ta đã chứng minh được rằng

$$\frac{T^*}{T^0} \le \frac{4}{3} - \frac{1}{M}$$, M là số máy

Với kết quả này, ta có thể xác lập được sai số mà chúng ta phải gánh chịu nếu dùng Heuristic thay vì tìm một lời giải tối ưu. Chẳng hạn với số máy là 2 (M=2) ta có $\frac{T^*}{T^0} \le \frac{7}{6}$, và đó chính là sai số cực đại mà trường hợp ở trên đã gánh chịu. Theo công thức này, số máy càng lớn thì sai số càng lớn.

Trong trường hợp M lớn thì tỷ số 1/M xem như bằng 0 . Như vậy, sai số tối đa mà ta phải chịu là T*□□ 4/3 T^0, nghĩa là sai số tối đa là 33%. Tuy nhiên, khó tìm ra được những trường hợp mà sai số đúng bằng giá trị cực đại, dù trong trường hợp xấu nhất. Thuật giải Heuristic trong trường hợp này rõ ràng đã cho chúng ta những lời giải tương đối tốt.

III. CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC

Qua các phần trước chúng ta tìm hiểu tổng quan về ý tưởng của thuật giải Heuristic (nguyên lý Greedy và sắp thứ tự). Trong mục này, chúng ta sẽ đi sâu vào tìm hiểu một số kỹ thuật tìm kiếm Heuristic – một lớp bài toán rất quan trọng và có nhiều ứng dụng trong thực tế.

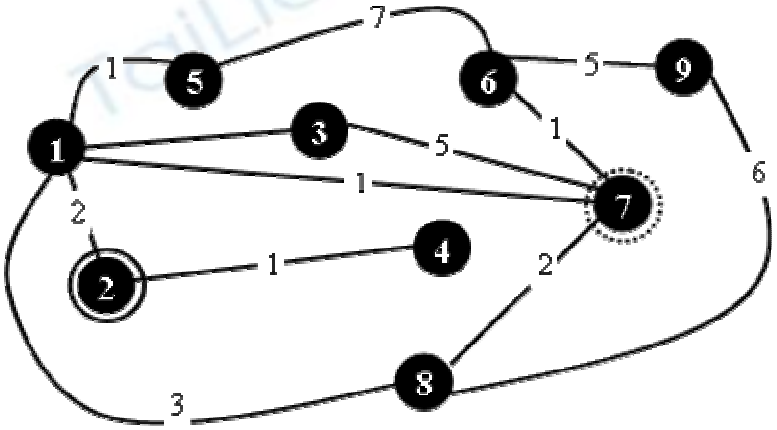
III.1. Cấu trúc chung của bài toán tìm kiếm

Để tiện lợi cho việc trình bày, ta hãy dành chút thời gian để làm rõ hơn "đối tượng" quan tâm của chúng ta trong mục này. Một cách chung nhất, nhiều vấn đề-bài toán phức tạp đều có dạng "tìm đường đi trong đồ thị" hay nói một cách hình thức hơn là *"xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến một đỉnh nào đó"*. Một phát biểu khác thường gặp của dạng bài toán này là :

Cho trước hai trạng thái T_0 và TG hãy xây dựng chuỗi trạng thái T_0, **T_1, T_2, ..., Tn_{-1}**, Tn = TG sao cho :

$\sum_1^n cost(T_{i-1}, T_i)$ thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).

Trong đó, Ti thuộc tập hợp S (gọi là không gian trạng thái – state space) bao gồm tất cả các trạng thái có thể có của bài toán và **cost(T_{i-1}, T_i)** là *chi phí* để *biến đổi* từ trạng thái Ti_{-1} sang trạng thái Ti. Dĩ nhiên, từ một trạng thái Ti ta có nhiều cách để biến đổi sang trạng thái Ti_{+1}. Khi nói đến một biến đổi cụ thể từ Ti_{-1} sang Ti ta sẽ dùng thuật ngữ *hướng đi (*với ngụ ý nói về sự lựa chọn).



Hình : Mô hình chung của các vấn đề-bài toán phải giải quyết bằng phương pháp tìm kiếm lời giải. Không gian tìm kiếm là một tập hợp trạng thái - tập các nút của đồ thị. Chi phí cần thiết để chuyển từ trạng thái T này sang trạng thái Tk được biểu diễn dưới dạng các con số nằm trên cung nối giữa hai nút tượng trưng cho hai trạng thái.

Đa số các bài toán thuộc dạng mà chúng ta đang mô tả đều có thể được biểu diễn dưới dạng đồ thị. Trong đó, một trạng thái là một đỉnh của đồ thị. Tập hợp S bao gồm tất cả các trạng thái chính là tập hợp bao gồm tất cả đỉnh của đồ thị. Việc biến đổi từ trạng thái Ti_{-1} sang trạng thái Ti là việc đi từ đỉnh đại diện cho Ti_{-1} sang đỉnh đại diện cho Ti theo cung nối giữa hai đỉnh này.

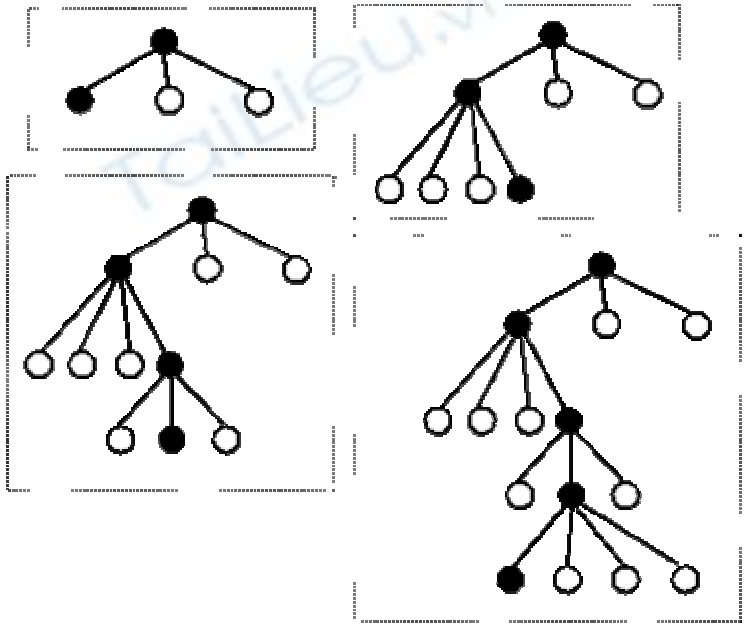
III.2. Tìm kiếm chiều sâu và tìm kiếm chiều rộng

Để bạn đọc có thể hình dung một cách cụ thể bản chất của thuật giải Heuristic, chúng ta nhất thiết phải nắm vững hai *chiến lược* tìm kiếm cơ bản là tìm kiếm theo chiều sâu (Depth First Search) và tìm kiếm theo chiều rộng (Breath First Search). Sở dĩ chúng ta dùng từ *chiến lược* mà không phải là *phương pháp* là bởi vì trong thực tế,

người ta hầu như chẳng bao giờ vận dụng một trong hai kiểm tìm kiếm này một cách trực tiếp mà không phải sửa đổi gì.

III.2.1. Tìm kiếm chiều sâu (Depth-First Search)

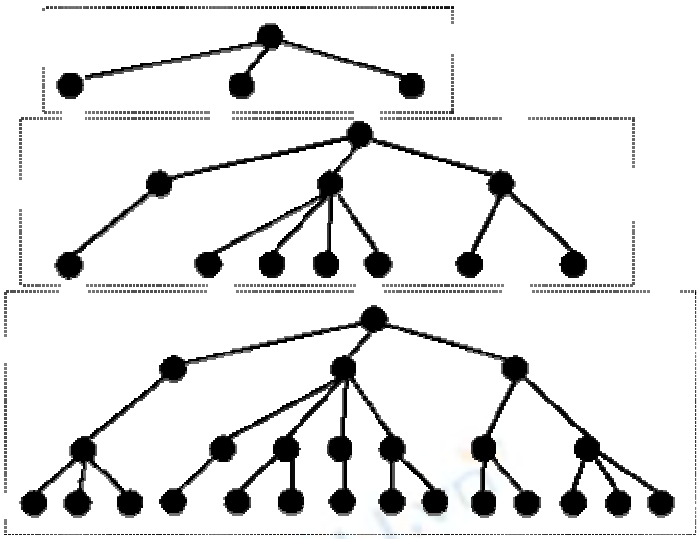
Trong tìm kiếm theo chiều sâu, tại trạng thái (đỉnh) hiện hành, ta chọn một trạng thái kế tiếp (trong tập các trạng thái có thể biến đổi thành từ trạng thái hiện tại) làm trạng thái hiện hành cho đến lúc trạng thái hiện hành là trạng thái đích. Trong trường hợp tại trạng thái hiện hành, ta không thể biến đổi thành trạng thái kế tiếp thì ta sẽ quay lui (back-tracking) lại trạng thái trước trạng thái hiện hành (trạng thái biến đổi thành trạng thái hiện hành) để chọn đường khác. Nếu ở trạng thái trước này mà cũng không thể biến đổi được nữa thì ta quay lui lại trạng thái trước nữa và cứ thế. Nếu đã quay lui đến trạng thái khởi đầu mà vẫn thất bại thì kết luận là không có lời giải. Hình ảnh sau minh họa hoạt động của tìm kiếm theo chiều sâu.



Hình : Hình ảnh của tìm kiếm chiều sâu. Nó chỉ lưu ý "mở rộng" trạng thái được chọn mà không "mở rộng" các trạng thái khác (nút màu trắng trong hình vẽ).

III.2.2. Tìm kiếm chiều rộng (Breath-First Search)

Ngược lại với tìm kiếm theo kiểu chiều sâu, tìm kiếm chiều rộng mang hình ảnh của vết dầu loang. Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, *ứng với mỗi* trạng thái Tk trong tập S, ta xây dựng tập Sk bao gồm các trạng thái kế tiếp của Tk rồi lần lượt bổ sung các Sk vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả Sk.



Hình : Hình ảnh của tìm kiếm chiều rộng. Tại một bước, mọi trạng thái đều được mở rộng, không bỏ sót trạng thái nào.

	Chiều sâu	Chiều rộng
Tính hiệu quả	Hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác. Hiệu quả của chiến lược phụ thuộc vào phương án chọn hướng đi. Phương án càng kém hiệu quả thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm chỉ một lời giải.	Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm. Hiệu quả của chiến lược phụ thuộc vào độ sâu của lời giải. Lời giải càng xa gốc thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm nhiều lời giải.
Lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Chỉ lưu lại các trạng thái chưa xét đến.	Phải lưu toàn bộ các trạng thái.
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ.
Trường hợp tốt nhất	Phương án chọn hướng đi *tuyệt đối* chính xác. Lời giải được xác định một cách trực tiếp.	Vét cạn toàn bộ.

Tìm kiếm chiều sâu và tìm kiếm chiều rộng đều là các phương pháp tìm kiếm có hệ thống và chắc chắn tìm ra lời giải. Tuy nhiên, do bản chất là vét cạn nên với những bài toán có không gian lớn thì ta không thể dùng hai chiến lược này được. Hơn nữa,

hai chiến lược này đều có tính chất "mù quáng" vì chúng không chú ý đến những thông tin (tri thức) ở trạng thái hiện thời và thông tin về đích cần đạt tới cùng mối quan hệ giữa chúng. Các tri thức này vô cùng quan trọng và rất có ý nghĩa để thiết kế các thuật giải hiệu quả hơn mà ta sắp sửa bàn đến.

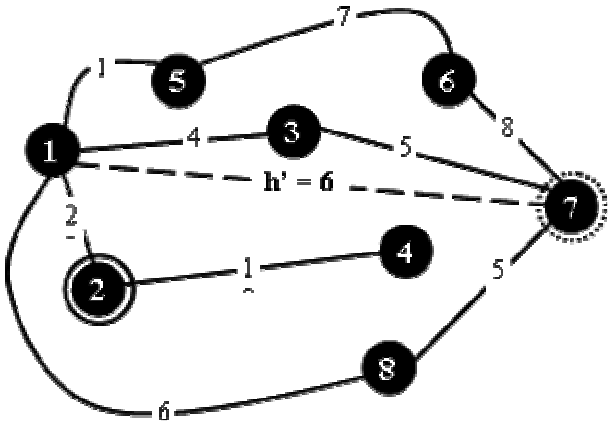
III.3. Tìm kiếm leo đồi

III.3.1. Leo đồi đơn giản

Tìm kiếm leo đồi theo đúng nghĩa, nói chung, thực chất chỉ là một trường hợp đặc biệt của tìm kiếm theo chiều sâu nhưng không thể quay lui. Trong tìm kiếm leo đồi, việc lựa chọn trạng thái tiếp theo được quyết định dựa trên một hàm Heuristic.

Hàm Heuristic là gì ?

Thuật ngữ "hàm Heuristic" muốn nói lên điều gì? Chẳng có gì ghê gớm. Bạn đã quen với nó rồi! Đó đơn giản chỉ là một ***ước lượng về khả năng dẫn đến lời giải*** tính từ trạng thái đó (*khoảng cách* giữa trạng thái hiện tại và trạng thái đích). Ta sẽ quy ước gọi hàm này là **h** trong suốt giáo trình này. Đôi lúc ta cũng đề cập đến ***chi phí tối ưu thực sự*** từ một trạng thái dẫn đến lời giải. Thông thường, giá trị này là không thể tính toán được (vì tính được đồng nghĩa là đã biết con đường đến lời giải !) mà ta chỉ dùng nó như một cơ sở để suy luận về mặt lý thuyết mà thôi ! Hàm **h**, ta quy ước rằng, luôn trả ra kết quả là một số không âm. Để bạn đọc thực sự nắm được ý nghĩa của hai hàm này, hãy quan sát hình sau trong đó minh họa chi phí tối ưu thực sự và chi phí ước lượng.



Hình Chi phí ước lượng h’ = 6 và chi phí tối ưu thực sự h = 4+5 = 9 (đi theo đường 1-3-7)

> Bạn đang ở trong một thành phố xa lạ mà không có bản đồ trong tay và ta muốn đi vào khu trung tâm? Một cách suy nghĩ đơn giản, chúng ta sẽ nhắm vào *hướng* những tòa cao ốc của khu trung tâm!

Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T_0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi không tồn tại một trạng thái tiếp theo hợp lệ (Tk) của trạng thái hiện hành :

a. Đặt Tk là một trạng thái tiếp theo hợp lệ của trạng thái hiện hành Ti

b. Đánh giá trạng thái Tk mới :

b.1. Nếu là trạng thái kết thúc thì trả về trị này và thoát.

b.2. Nếu không phải là trạng thái kết thúc nhưng **tốt** hơn trạng thái hiện hành thì cập nhật nó thành trạng thái hiện hành.

b.3. Nếu nó không tốt hơn trạng thái hiện hành thì tiếp tục vòng lặp.

Mã giả

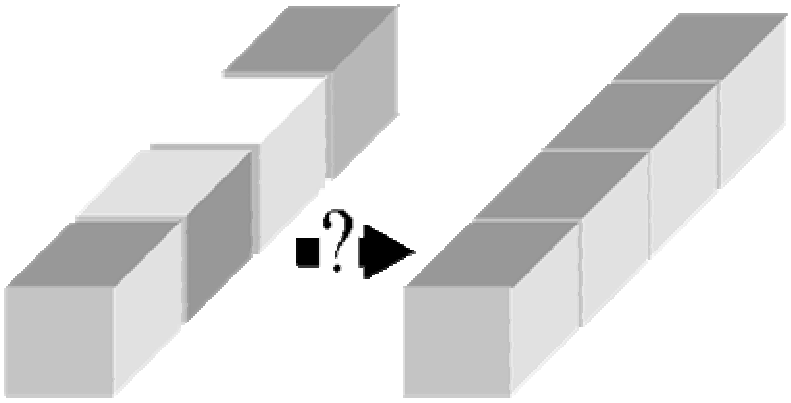
```
Ti := T0; Stop :=FALSE;
WHILE Stop=FALSE DO BEGIN
    IF Ti ☐ TG THEN BEGIN
            <tìm được kết quả >; Stop:=TRUE;
    END;
    ELSE BEGIN
        Better:=FALSE;
        WHILE (Better=FALSE) AND (STOP=FALSE) DO BEGIN
            IF <không tồn tại trạng thái kế tiếp hợp lệ của Ti>
            THEN BEGIN
                <không tìm được kết quả >; Stop:=TRUE; END;
            ELSE BEGIN
                Tk := <một trạng thái kế tiếp hợp lệ của Ti>;
                IF <h(Tk) tốt hơn h(Ti)> THEN BEGIN
                    Ti :=Tk; Better:=TRUE;
```

```
                    END;

                END;

            END; {WHILE}

        END; {ELSE}

    END;{WHILE}
```

Mệnh đề "h'(Tk) tốt hơn h'(Ti)" nghĩa là gì? Đây là một khái niệm chung chung. Khi cài đặt thuật giải, ta phải cung cấp một định nghĩa tường minh về *tốt hơn.* Trong một số trường hợp, tốt hơn là nhỏ hơn : h'(Tk) < h'(Ti); một số trường hợp khác tốt hơn là lớn hơn h'(Tk) > h'(Ti)...Chẳng hạn, đối với bài toán tìm đường đi ngắn nhất giữa hai điểm. Nếu dùng hàm h' là hàm cho ra *khoảng cách theo đường chim bay* giữa vị trí hiện tại (trạng thái hiện tại) và đích đến (trạng thái đích) thì tốt hơn nghĩa là nhỏ hơn.

Vấn đề cần làm rõ kế tiếp là thế nào là <một trạng thái kế tiếp hợp lệ của Ti>? Một trạng thái kế tiếp hợp lệ là trạng thái chưa được xét đến. Giả sử h của trạng thái hiện tại Ti có giá trị là h(Ti) = 1.23 và từ Ti ta có thể biến đổi sang một trong 3 trạng thái kế tiếp lần lượt là Tk_1, Tk_2, Tk_3 với giá trị các hàm h tương ứng là $h(Tk_1) = 1.67$, $h(Tk_2) = 2.52$, $h'(Tk_3) = 1.04$. Đầu tiên, Tk sẽ được gán bằng Tk_1, nhưng vì $h'(Tk) = h'(T_{k1}) > h'(Ti)$ nên Tk không được chọn. Kế tiếp là Tk sẽ được gán bằng Tk_2 và cũng không được chọn. Cuối cùng thì Tk_3 được chọn. Nhưng giả sử $h'(Tk_3) = 1.3$ thì cả Tk_3 cũng không được chọn và mệnh đề *<không thể sinh ra trạng thái kế tiếp của Ti>* sẽ có giá trị TRUE. Giải thích này có vẻ hiển nhiên nhưng có lẽ cần thiết để tránh nhầm lẫn cho bạn đọc.

Để thấy rõ hoạt động của thuật giải leo đồi. Ta hãy xét một bài toán minh họa sau. Cho 4 khối lập phương *giống nhau* A, B, C, D. Trong đó các mặt (M1), (M2), (M3), (M4), (M5), (M6) có thể được tô bằng 1 trong 6 màu (1), (2), (3), (4), (5), (6). Ban đầu các khối lập phương được xếp vào một hàng. Mỗi một bước, ta chỉ được xoay một khối lập phương quanh một trục (X,Y,Z) 90^0 theo chiều bất kỳ (nghĩa là ngược chiều hay thuận chiều kim đồng hồ cũng được). Hãy xác định số bước quay ít nhất sao cho tất cả các mặt của khối lập phương trên 4 mặt của hàng là có cùng màu như hình vẽ.



Hình : Bài toán 4 khối lập phương

Để giải quyết vấn đề, trước hết ta cần định nghĩa một hàm **G** dùng để đánh giá một tình trạng cụ thể có phải là lời giải hay không? Bạn đọc có thể dễ dàng đưa ra một cài đặt của hàm G như sau :

```
IF (Gtrái + Gphải + Gtrên + Gdưới + Gtrước + Gsau) = 16 THEN
    G:=TRUE
ELSE
    G:=FALSE;
```

Trong đó, Gphải là *số lượng các mặt* có cùng màu của mặt bên phải của hàng. Tương tự cho Gtrái, Gtrên, Ggiữa, Gtrước, Gsau. Tuy nhiên, do các khối lập phương A,B,C,D là hoàn toàn tương tự nhau nên tương quan giữa các mặt của mỗi khối là giống nhau. Do đó, nếu có 2 mặt không đối nhau trên hàng đồng màu thì 4 mặt còn lại của hàng cũng đồng màu. Từ đó ta chỉ cần hàm G được định nghĩa như sau là đủ :

```
IF Gphải + Gdưới = 8 THEN
    G:=TRUE
ELSE
    G:=FALSE;
```

Hàm **h** (ước lượng khả năng dẫn đến lời giải của một trạng thái) sẽ được định nghĩa như sau :

```
h = Gtrái + Gphải + Gtrên + Gdưới
```

Bài toán này đủ đơn giản để thuật giải leo đồi có thể hoạt động tốt. Tuy nhiên, không phải lúc nào ta cũng may mắn như thế!

Đến đây, có thể chúng ta sẽ nảy sinh một ý tưởng. Nếu đã chọn trạng thái *tốt hơn* làm trạng thái hiện tại thì tại sao không chọn trạng thái *tốt nhất* ? Như vậy, *có lẽ* ta sẽ nhanh chóng dẫn đến lời giải hơn! Ta sẽ bàn luận về vấn đề: "liệu cải tiến này có thực sự giúp chúng ta dẫn đến lời giải nhanh hơn hay không?" ngay sau khi trình bày xong thuật giải leo đồi dốc đứng.

III.3.2. Leo đồi dốc đứng

Về cơ bản, leo đồi dốc đứng cũng giống như leo đồi, chỉ khác ở điểm là leo đồi dốc đứng sẽ duyệt tất cả các hướng đi có thể và chọn đi theo trạng thái *tốt nhất* trong số các trạng thái kế tiếp có thể có (trong khi đó leo đồi chỉ chọn đi theo trạng thái kế tiếp đầu tiên *tốt hơn* trạng thái hiện hành mà nó tìm thấy).

Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T_0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi (Ti) không tồn tại một trạng thái kế tiếp (Tk) nào tốt hơn trạng thái hiện tại (Ti)

a) Đặt S bằng tập tất cả trạng thái kế tiếp có thể có của T_i *và* tốt hơn Ti.

b) Xác định Tkmax là trạng thái tốt nhất trong tập S

Đặt Ti = Tkmax

Mã giả

```
Ti := T0;
Stop :=FALSE;
WHILE Stop=FALSE DO BEGIN
    IF Ti ☐ TG THEN BEGIN
        <tìm được kết quả >;
        STOP :=TRUE;
        END;
    ELSE BEGIN
        Best:=h'(Ti);
        Tmax := Ti;
        WHILE <tồn tại trạng thái kế tiếp hợp lệ của Ti> DO BEGIN
            Tk := <một trạng thái kế tiếp hợp lệ của Ti>;
            IF <h'(Tk) tốt hơn Best> THEN BEGIN
                Best :=h'(Tk);
                Tmax := Tk;
            END;
```

```
                END;
                IF (Best>Ti) THEN
                        Ti := Tmax;
                ELSE BEGIN
                        <không tìm được kết quả >;
                        STOP:=TRUE;
                END;
            END; {ELSE IF}
        END;{WHILE STOP}
```

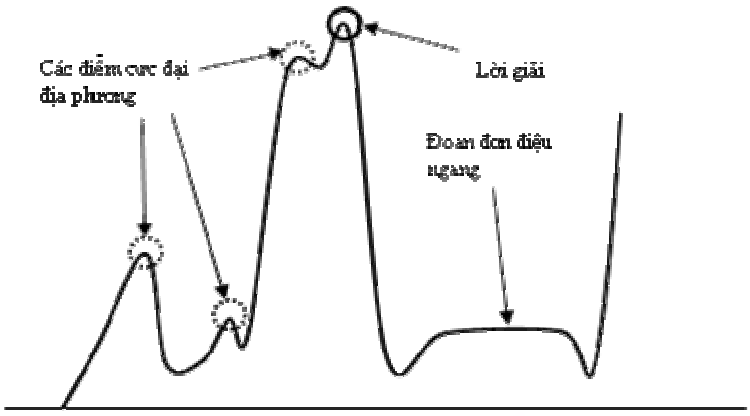
III.3.3. Đánh giá

So với leo đồi đơn giản, leo đồi dốc đứng có ưu điểm là luôn luôn chọn hướng có triển vọng nhất để đi. Liệu điều này có đảm bảo leo đồi dốc đứng luôn tốt hơn leo đồi đơn giản không? Câu trả lời là không. Leo đồi dốc đứng chỉ tốt hơn leo đồi đơn giản trong một số trường hợp mà thôi. Để chọn ra được hướng đi tốt nhất, leo đồi dốc đứng phải duyệt qua *tất cả* các hướng đi có thể có tại trạng thái hiện hành. Trong khi đó, leo đồi đơn giản chỉ chọn đi theo trạng thái *đầu tiên* tốt hơn (so với trạng thái hiện hành) mà nó tìm ra được. Do đó, thời gian cần thiết để leo đồi dốc đứng chọn được một hướng đi sẽ lớn hơn so với leo đồi đơn giản. Tuy vậy, do lúc nào cũng chọn hướng đi tốt nhất nên leo đồi dốc đứng thường sẽ tìm đến lời giải sau một số bước ít hơn so với leo đồi đơn giản. Nói một cách ngắn gọn, leo đồi dốc đứng sẽ tốn nhiều thời gian hơn cho một bước nhưng lại đi ít bước hơn; còn leo đồi đơn giản tốn ít thời gian hơn cho một bước đi nhưng lại phải đi nhiều bước hơn. Đây chính là yếu tố được và mất giữa hai thuật giải nên ta phải cân nhắc kỹ lưỡng khi lựa chọn thuật giải.

Cả hai phương pháp leo núi đơn giản và leo núi dốc đứng đều có khả năng thất bại trong việc tìm lời giải của bài toán mặc dù lời giải đó thực sự hiện hữu. Cả hai giải thuật đều có thể kết thúc khi đạt được một trạng thái mà không còn trạng thái nào tốt hơn nữa có thể phát sinh nhưng trạng thái này không phải là trạng thái đích. Điều này sẽ xảy ra nếu chương trình đạt đến một điểm cực đại địa phương, một đoạn đơn điệu ngang.

Điểm cực đại địa phương (a local maximum) : là một trạng thái tốt hơn tất cả lân cận của nó nhưng không tốt hơn một số trạng thái khác ở xa hơn. Nghĩa là tại một điểm cực đại địa phương, mọi trạng thái *trong một lân cận* của trạng thái hiện tại đều *xấu hơn* trạng thái hiện tại. Tuy có dáng vẻ của lời giải nhưng các cực đại địa phương không phải là lời giải thực sự. Trong trường hợp này, chúng được gọi là những ngọn đồi thấp.

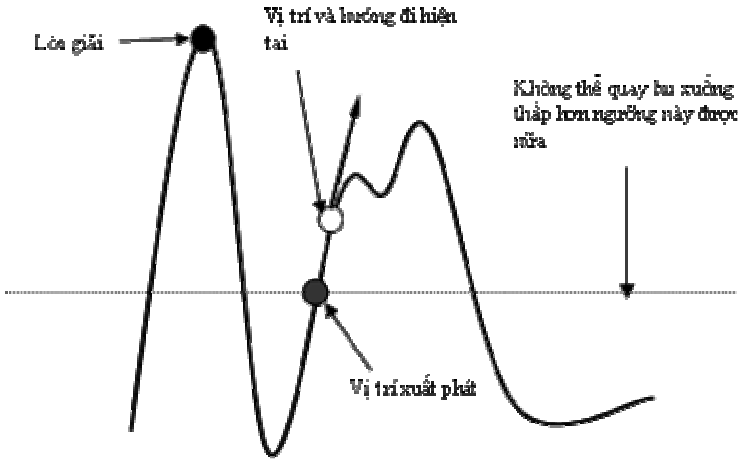
Đoạn đơn điệu ngang (a plateau) : là một vùng bằng phẳng của không gian tìm kiếm, trong đó, toàn bộ các trạng thái lân cận đều có cùng giá trị.



Hình : Các tình huống khó khăn cho tìm kiếm leo đèo.

Để đối phó với các các điểm này, người ta đã đưa ra một số giải pháp. Ta sẽ tìm hiểu 2 trong số các giải pháp này. Những giải này, không thực sự giải quyết trọn vẹn vấn đề mà chỉ là một phương án cứu nguy tạm thời mà thôi.

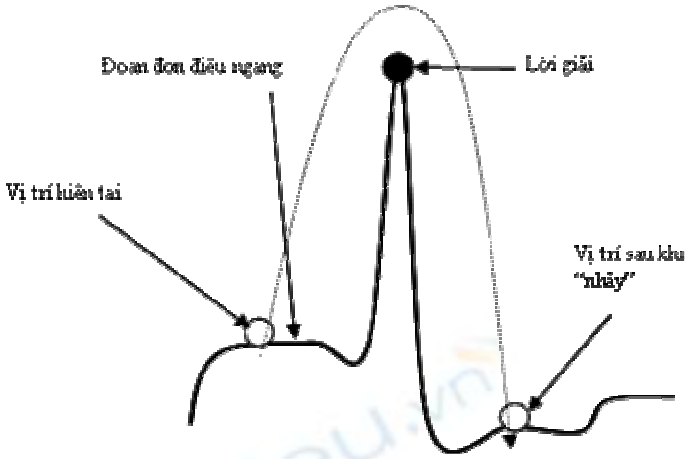
Phương án đầu tiên là kết hợp leo đồi và quay lui. Ta sẽ quay lui lại các trạng thái trước đó và thử đi theo hướng khác. Thao tác này hợp lý nếu tại các trạng thái trước đó có một hướng đi tốt mà ta đã bỏ qua trước đó. Đây là một cách khá hay để đối phó với các điểm cực đại địa phương. Tuy nhiên, do đặc điểm của leo đồi là "bước sau cao hơn bước trước" nên phương án này sẽ thất bại khi ta xuất phát từ một điểm quá cao hoặc xuất phát từ một đỉnh đồi mà để đến được lời giải cần phải đi qua một "thung lũng" thật sâu như trong hình sau.



Hình : Một trường hợp thất bại của leo đèo kết hợp quay lui.

Cách thứ hai là thực hiện một bước *nhảy vọt* theo hướng nào đó để thử đến một vùng mới của không gian tìm kiếm. Nôm na là "bước" liên tục nhiều "bước" (chẳng hạn 5,7,10, …) mà tạm thời "quên" đi việc kiểm tra "bước sau cao hơn bước trước". Tiếp cận có vẻ hiệu quả khi ta gặp phải một đoạn đơn điệu ngang. Tuy nhiên, nhảy vọt cũng có nghĩa là ta đã bỏ qua cơ hội để tiến đến lời giải thực sự. Trong trường hợp chúng ta đang đứng khá gần lời giải, việc nhảy vọt sẽ đưa chúng ta sang một vị trí hoàn toàn xa lạ, mà từ đó, có thể sẽ dẫn chúng ta đến một rắc rối kiểu khác. Hơn

nữa, số bước nhảy là bao nhiêu và nhảy theo hướng nào là một vấn đề phụ thuộc rất nhiều vào đặc điểm không gian tìm kiếm của bài toán.



Hình Một trường hợp khó khăn cho phương án "nhảy vọt".

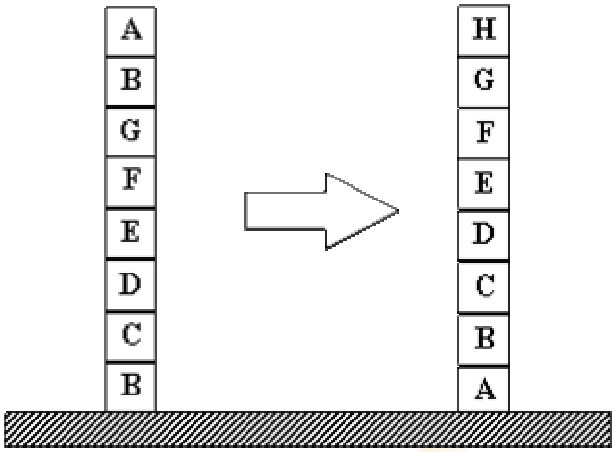
Leo núi là một phương pháp cục bộ bởi vì nó quyết định sẽ làm gì tiếp theo dựa vào một đánh giá về trạng thái hiện tại và các trạng thái kế tiếp có thể có (*tốt hơn* trạng thái hiện tại, trạng thái *tốt nhất* tốt hơn trạng thái hiện tại) thay vì phải xem xét một cách toàn diện trên tất cả các trạng thái đã đi qua. Thuận lợi của leo núi là ít gặp sự bùng nổ tổ hợp hơn so với các phương pháp toàn cục. Nhưng nó cũng giống như các phương pháp cục bộ khác ở chỗ là không chắc chắn tìm ra lời giải trong trường hợp xấu nhất.

Một lần nữa, ta khẳng định lại vai trò quyết định của hàm Heuristic trong quá trình tìm kiếm lời giải. Với cùng một thuật giải (như leo đồi chẳng hạn), nếu ta có một hàm Heuristic tốt hơn thì kết quả sẽ được tìm thấy nhanh hơn. Ta hãy xét bài toán về các khối được trình bày ở hình sau. Ta có hai thao tác biến đổi là:

+ Lấy một khối ở đỉnh một cột bất kỳ và đặt nó lên một chỗ trống tạo thành một cột mới. Lưu ý là chỉ có thể tạo ra tối đa 2 cột mới.

+ Lấy một khối ở đỉnh một cột và đặt nó lên đỉnh một cột khác

Hãy xác định số thao tác ít nhất để biến đổi cột đã cho thành cột kết quả.



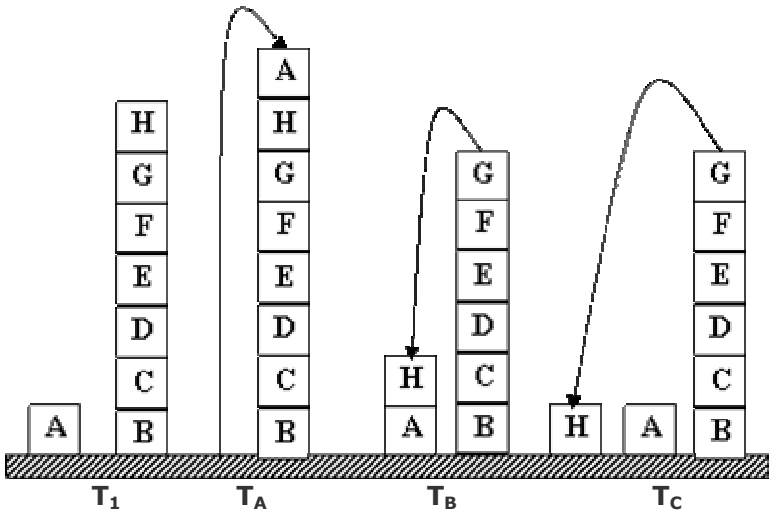
Hình : Trạng thái khởi đầu và trạng thái kết thúc

Giả sử ban đầu ta dùng một hàm Heuristic đơn giản như sau :

> **H_1** : Cộng 1 điểm cho mỗi khối ở vị trí đúng so với trạng thái đích. Trừ 1 điểm cho mỗi khối đặt ở vị trí sai so với trạng thái đích.

Dùng hàm này, trạng thái kết thúc sẽ có giá trị là 8 vì cả 8 khối đều được đặt ở vị trí đúng. Trạng thái khởi đầu có giá trị là 4 (vì nó có 1 điểm cộng cho các khối C, D, E, F, G, H và 1 điểm trừ cho các khối A và B). Chỉ có thể có một di chuyển từ trạng thái khởi đầu, đó là dịch chuyển khối A xuống tạo thành một cột mới (T_1).

Điều đó sinh ra một trạng thái với số điểm là **6** (vì vị trí của khối A bây giờ sinh ra 1 điểm cộng hơn là một điểm trừ). Thủ tục leo núi sẽ chấp nhận sự dịch chuyển đó. Từ trạng thái mới T_1, có ba di chuyển có thể thực hiện dẫn đến ba trạng thái **Ta, Tb, Tc** được minh họa trong hình dưới. Những trạng thái này có số điểm là : h'(Ta)= 4; h'(Tb) = 4 và h'(Tc) = 4



Hình Các trạng thái có thể đạt được từ T_1

Thủ tục leo núi sẽ tạm dừng bởi vì tất cả các trạng thái này có số điểm thấp hơn trạng thái hiện hành. Quá trình tìm kiếm chỉ dừng lại ở một trạng thái cực đại địa phương mà không phải là cực đại toàn cục.

Chúng ta có thể đổ lỗi cho chính giải thuật leo đồi vì đã thất bại do không đủ tầm nhìn tổng quát để tìm ra lời giải. Nhưng chúng ta cũng có thể đổ lỗi cho hàm Heuristic và cố gắng sửa đổi nó. Giả sử ta thay hàm ban đầu bằng hàm Heuristic sau đây :

> **H_2 :** Đối với mỗi khối phụ trợ đúng (khối phụ trợ là khối nằm bên dưới khối hiện tại), cộng 1 điểm, ngược lại trừ 1 điểm.

Dùng hàm này, trạng thái kết thúc có số điểm là **28** vì B nằm đúng vị trí và không có khối phụ trợ nào, C đúng vị trí được 1 điểm cộng với 1 điểm do khối phụ trợ B nằm đúng vị trí nên C được 2 điểm, D được 3 điểm, ...Trạng thái khởi đầu có số điểm là **–28**. Việc di chuyển A xuống tạo thành một cột mới làm sinh ra một trạng thái với số điểm là $h'(T_1)$ = **–21** vì A không còn 7 khối sai phía dưới nó nữa. Ba trạng thái có thể phát sinh tiếp theo bây giờ có các điểm số là : h'(Ta)=–28; h'(Tb)=–16 và h'(Tc) = –15. Lúc này thủ tục leo núi dốc đứng sẽ chọn di chuyển đến trạng thái Tc, ở đó có một khối đúng. Qua hàm H_2 này ta rút ra một nguyên tắc : *tốt hơn* không chỉ có nghĩa là có *nhiều ưu điểm* hơn mà còn phải *ít khuyết điểm* hơn. Hơn nữa, khuyết điểm không có nghĩa chỉ là sự sai biệt ngay tại một vị trí mà còn là sự khác biệt trong tương quan giữa các vị trí. Rõ ràng là đứng về mặt kết quả, cùng một thủ tục leo đồi nhưng hàm H_1 bị thất bại (do chỉ biết đánh giá ưu điểm) còn hàm H_2 mới này lại hoạt động một cách hoàn hảo (do biết đánh giá cả ưu điểm và khuyết điểm).

Đáng tiếc, không phải lúc nào chúng ta cũng thiết kế được một hàm Heuristic hoàn hảo như thế. Vì việc đánh giá ưu điểm đã khó, việc đánh giá khuyết điểm càng khó và tinh tế hơn. Chẳng hạn, xét lại vấn đề muốn đi vào khu trung tâm của một thành phố *xa lạ*. Để hàm Heuristic hiệu quả, ta cần phải đưa các thông tin về các đường một chiều và các ngõ cụt, mà trong trường hợp một thành phố hoàn toàn xa lạ thì ta khó hoặc không thể biết được những thông tin này.

Đến đây, chúng ta hiểu rõ bản chất của hai thuật giải tiếp cận theo chiến lược tìm kiếm chiều sâu. Hiệu quả của cả hai thuật giải leo đồi đơn giản và leo đồi dốc đứng phụ thuộc vào :

+ Chất lượng của hàm Heuristic.

+ Đặc điểm của không gian trạng thái.

+ Trạng thái khởi đầu.

Sau đây, chúng ta sẽ tìm hiểu một tiếp cận theo mới, kết hợp được sức mạnh của cả tìm kiếm chiều sâu và tìm kiếm chiều rộng. Một thuật giải rất linh động và có thể nói là một thuật giải kinh điển của Heuristic.