CHƯƠNG 3: CÁC PHƯƠNG PHÁP TÌM KIẾM TỐI ƯU

3.1. Tìm đường đi ngắn nhất

3.1.1. Thuật toán A*

3.1.2. Thuật toán nhánh và cận

3.2. Tìm đối tượng tốt nhất

3.2.1. Tìm kiếm leo đồi

3.2.2. Tìm kiếm radient

3.2.3. Tìm kiếm mô phỏng luyện kim

3.3. Tìm kiếm mô phỏng sự tiến hoá.

Thuật toán di truyền

3.1. TÌM ĐƯỜNG ĐI NGẮN NHẤT

Khác với quá trình tìm kiếm trước, tìm đối tượng tốt nhất x,x $\in u$ chưa xác định được đích của quá trình tìm kiếm. Một số kĩ thuật được sử dụng trong phần này gồm:

Kỹ thuật tìm kiếm leo đồi để tìm đối tượng tốt nhất

Kỹ thuật tìm kiếm Gradient (gradient search)

Kỹ thuật tìm kiếm mô phỏng luyện kim (simulate annealing)

3.1.1. THUẬT TOÁN A*

- **Ý tưởng:**
  - Sử dụng hàm Heuristics chấp nhận được + tìm kiếm theo chiều rộng → Loại bỏ những đường đi có chi phí cao.
- **Hàm lượng giá: f(u) = g(u) + h(u)**
  - g(u) = Chi phí để đến u
  - h(u) = Lượng giá từ u đến đích
  - f(u) = Ước lượng tổng giá đến đích qua u.

3.1.1. CÀI ĐẶT THUẬT TOÁN A*

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 đích **then** {thông báo thành công; stop}

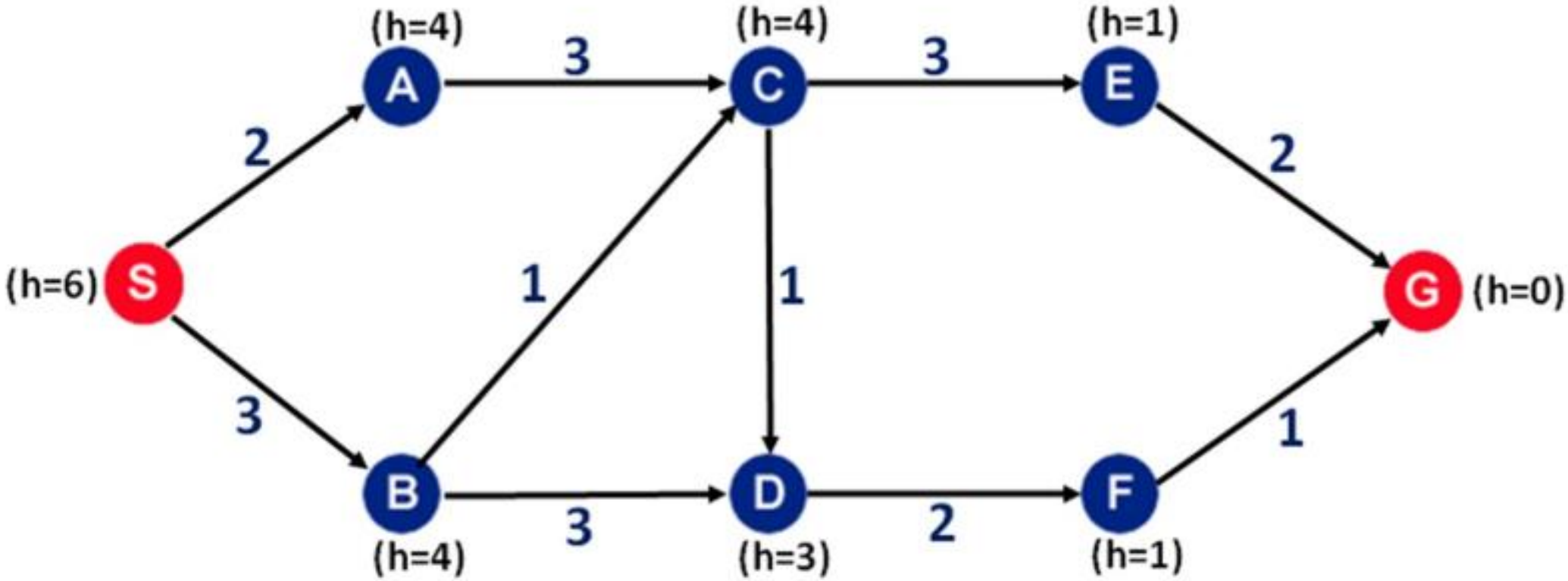
   2.4. **for** mỗi trạng thái **v** kề **u do**

   { **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 giảm dần của hàm **f** sao cho trạng thái có giá trị của hàm **f** nhỏ nhất ở đầu danh sách;

3. **End**;

3.1.1. THUẬT TOÁN A*



TT	Lấy ra	L
1		S
2	S	A_S=2+4=6, B_S=3+4=7
3	A_S	C_A=2+3+4=9, B_S=7
4	B_S	C_B=3+1+4=8, D_B=3+3+3=9, C_A=9
5	C_B	D_C=3+1+1+3=8, E_C=3+1+3+1=8,D_B=9, C_A=9
6	D_C	F_D=3+1+1+2+1=8, E_C=8, D_B=9, C_A=9
7	E_C	G_E=3+1+3+2=9, F_D=8, D_B=9, C_A=9
8	F_D	G_F=3+1+1+2+1= 8, G_E=9, D_B=9, C_A=9
9	G_F	n ∈ G → STOP
Fi		S→B→C→D→F→G=8

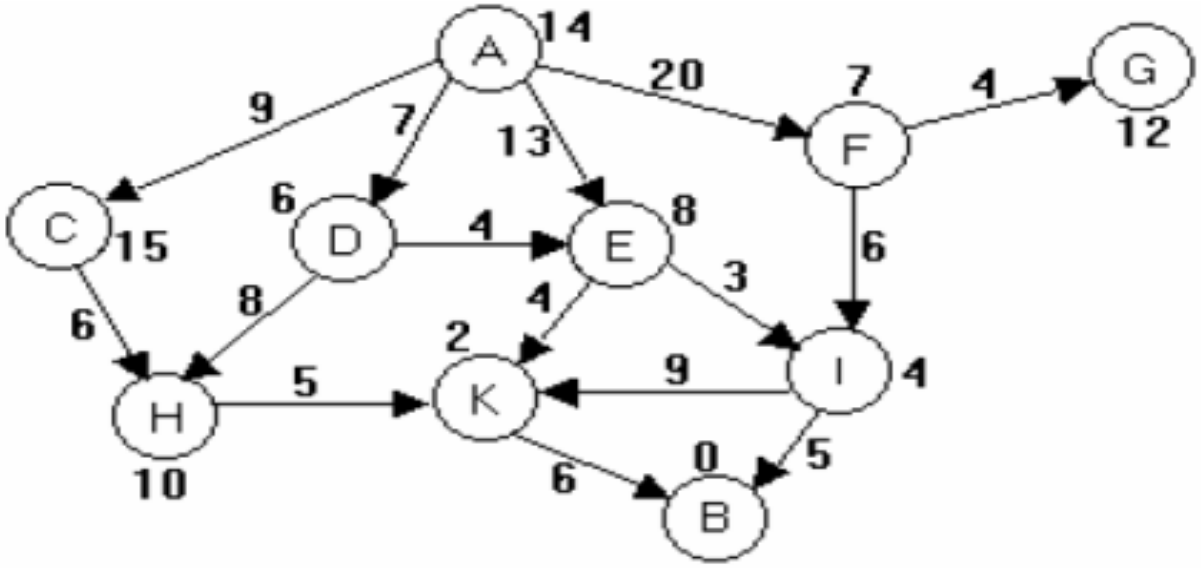
Chú ý:

+ Chữ màu đỏ là các bước lấy ra khỏi L (nhỏ nhất)
+ Ưu tiên thứ tự ABC
+ Không phân biệt vị trí trong L

3.1.1. THUẬT TOÁN A* (VÍ DỤ 2)

Đầu vào:

- Trạng thái đầu A,
- Trạng thái đích B.
- Các giá trị ghi trên cạnh là độ dài đường đi;
- Các số cạnh các đỉnh là giá trị hàm h.



Không gian trạng thái với hàm đánh giá

Yêu cầu:

- Tìm đường đi ngắn nhất từ A đến B bằng A*.

3.1.1. THUẬT TOÁN A*

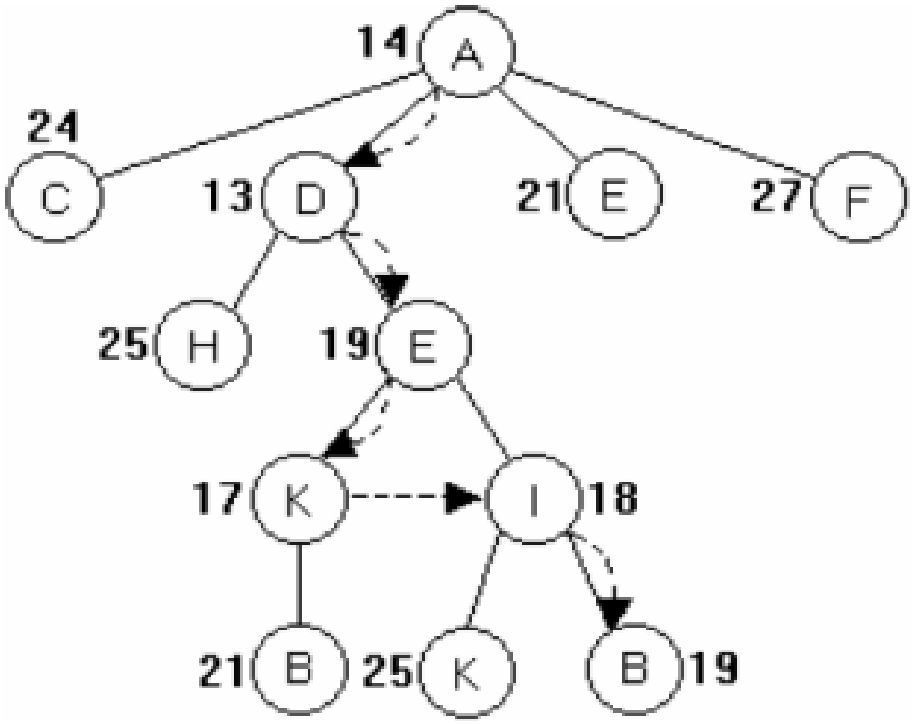
Thực hiện:

- Phát triển đỉnh A sinh ra các đỉnh con C, D, E, F.
- g(C) = 9, h(C) = 15 → f(C) = 9 + 15 = 24
- g(D) = 7, h(D) = 6 → f(D) = 7 + 6 = 13
- g(E) = 13, h(E) = 8 → f(E) = 13 + 8 = 21
- g(F) = 20, h(F) = 7 → f(F) = 20 + 7 = 27

→ Như vậy, đỉnh D được chọn để phát triển (f(D) = 13)

3.1.1. THUẬT TOÁN A*

- Phát triển D, nhận được các đỉnh con H và E (mới). Trong đó:
  - g(H) = g(D) + độ dài cung (D,H) = 7 + 8 = 15
  - h(H) = 10
  - → f(H) = g(H) + h(H) = 15 + 10 = 25.



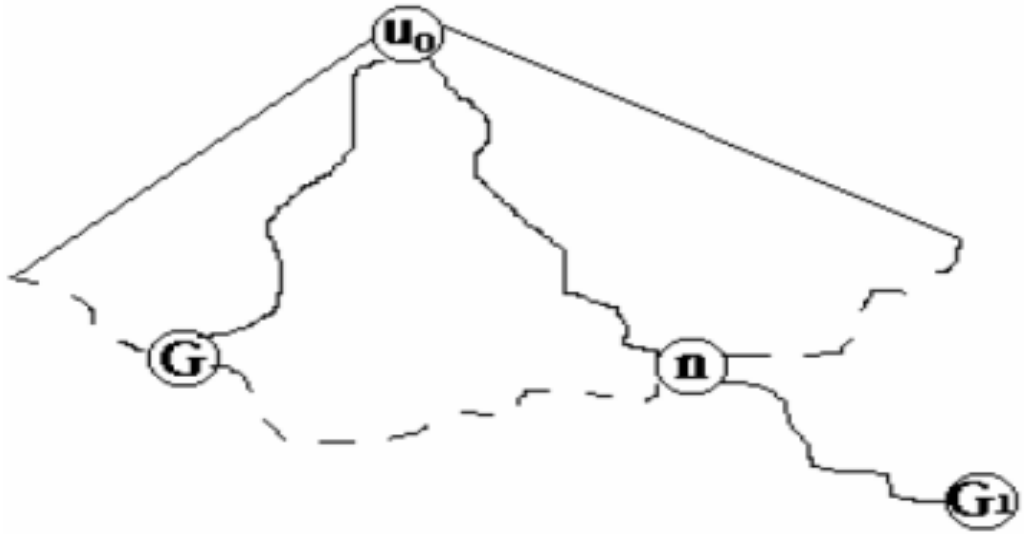
  - g(E) = g(D) + độ dài cung (D,E) = 7 +4 = 11
  - h(E) = 8

  → f(E) = g(E) + h(E) = 11 + 8 = 19.

- Như vậy đỉnh E sẽ được dùng để phát triển tiếp
- Tương tự sẽ chọn được các đỉnh K, B (đích).
- Với g(B) = 19, h(B) = 0.
- Đường đi: A → D → E → I →B

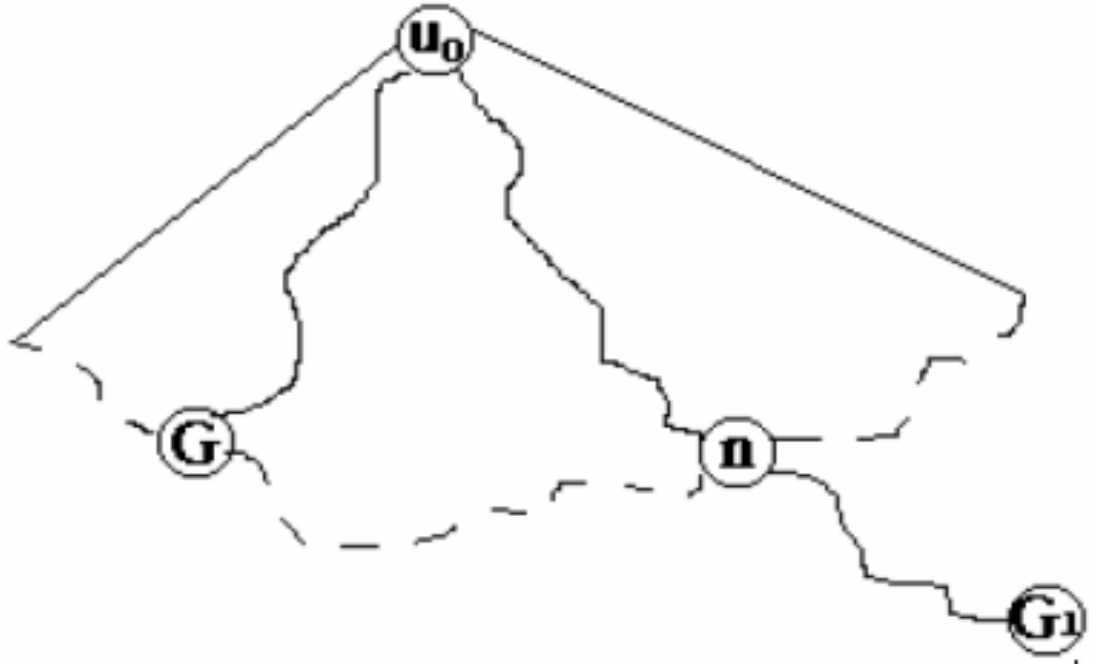
3.1.1. TÍNH TỐI ƯU CỦA THUẬT TOÁN A*

- Giả sử thuật toán dừng ở G, với độ dài đường đi là g(G) và ta có h(G)=0 nên f(G)=g(G).
- Giả sử nghiệm tối ưu không phải là G, tối ưu là G_1, với độ dài đường đi là S.
- Như vậy, đường đi này bị tách ra ở vị trí N nào đó. Khi đó có 2 khả năng:
  - N trùng G_1, hoặc
  - N không trùng G_1.



3.1.1. TÍNH TỐI ƯU CỦA THUẬT TOÁN A*

- Nếu $N \equiv G_1$:
  - G được chọn trước G_1, nên $f(G) \le f(G_1)$ vì $g(G) \le g(G_1) = S$
- Nếu $N \neq G1$:
  - Do h(u) là đánh giá thấp → $f(N) = g(N) + h(N) \le S$.
  - Do G được chọn trước → $f(G) \le f(N) \le S$.
  - Vậy G là đường đi tối ưu.



3.1.1. TÍNH TỐI ƯU CỦA THUẬT TOÁN A*

Nhận xét về A*

- Nếu hàm h(u) đánh giá thấp nhất thì thuật toán A* là tối ưu.
- Nếu các cung không nhỏ hơn một số **α** nào đó thì A* là đầy đủ.
- Nếu 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).

8.5. Phân tích A*

- **Đủ?**
  - Có (Trừ phi có vô hạn node với $f \leq f(G)$).
- **Độ phức tạp thời gian?**
  - Hàm mũ
- **Không gian?**
  - Lưu trữ tất cả các node
- **Tối ưu?** Có

3.1.2. THUẬT TOÁN TÌM KIẾM NHÁNH VÀ CẬN

Ý tưởng:

- Thuật toán nhánh và cận sử dụng tìm kiếm leo đồi với hàm đánh giá f(u).
- Tại trạng thái u, chọn trạng thái v trong số các trạng thái kề với u, với f(v) đạt min.
- Tương tự cho đến khi:
  - v là đích, hoặc
  - v không có đỉnh kề, hoặc
  - v có f(v) lớn hơn độ dài đường đi tối ưu hiện thời.

  → Không phát triển v nữa, quay về cha của v để tìm trạng thái tốt nhất trong các trạng thái còn lại chưa xét.

3.1.2. THUẬT TOÁN TÌM KIẾM NHÁNH VÀ CẬN

- Tìm kiếm leo đồi + hàm đánh giá f(u)

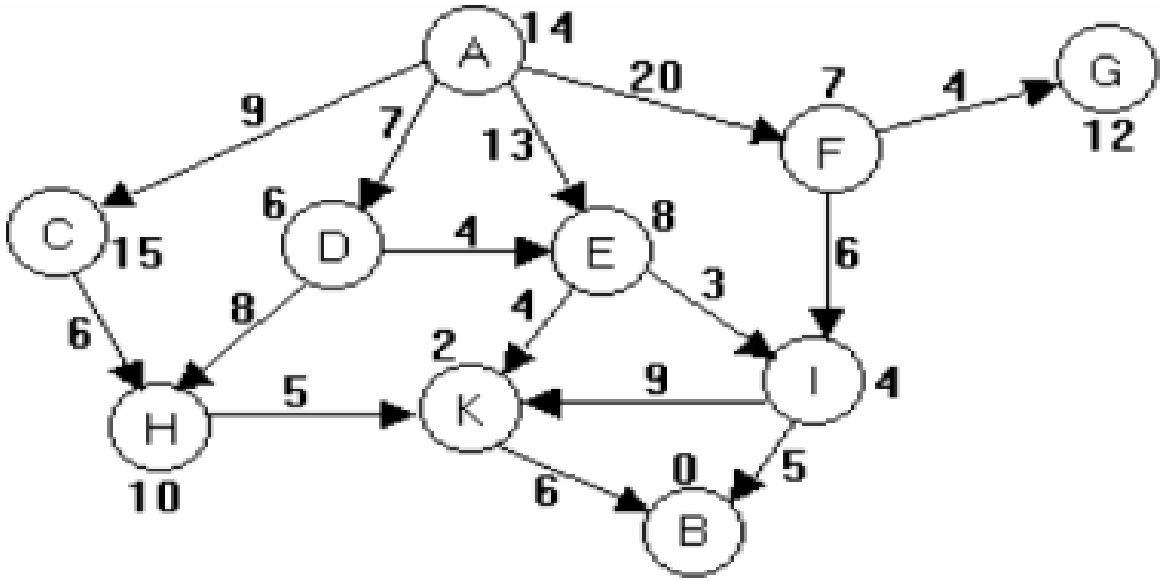
Procedure Branch-and-Bound;
Begin
1. Khởi tạo danh sách L chỉ chứa trạng thái đầu;
   Gán giá trị ban đầu cho cost;
2. **Loop do**
   - 2.1 **If** L rỗng **then** {thông báo thất bại; stop};
   - 2.2 Loại trạng thái u ở đầu danh sách L;
   - 2.3 **If** u là trạng thái kết thúc **then**
     if g(u)<=cost 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 L_1};
   - 2.6 Sắp xếp L_1 theo thứ tự tăng dần của hàm f;
   - 2.7 Chuyển danh sách L_1vào đầu danh sách L sao cho L_1 ở đầu danh sách L;

End;

3.1.2. THUẬT TOÁN TÌM KIẾM NHÁNH VÀ CẬN

Xét không gian trạng thái bên.

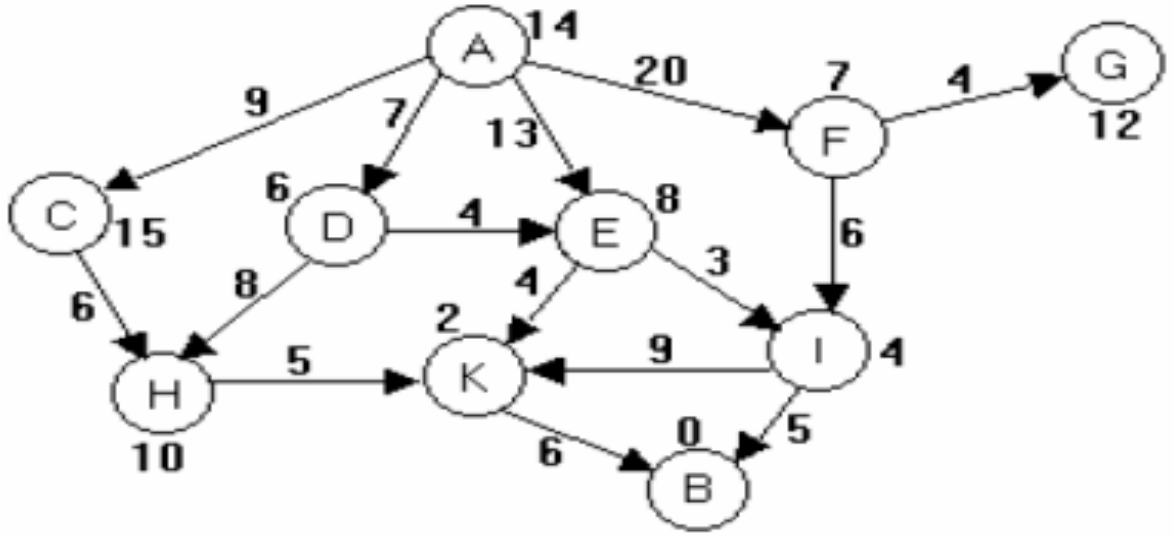
- Phát triển A, có các đỉnh con C, D, E, F với f(C) = 24; **f(D) = 13**; f(E) = 21; f(F) = 27.
- Chọn D, sinh các con H, E (mới) với f(H) = 25; **f(E) = 19**.
- Chọn E, sinh ra K, I với **f(K) = 17**; f(I) = 18.
- Chọn K, sinh ra B với **f(B) = g(B) = 21** → ***đường đi tạm thời là 21.***



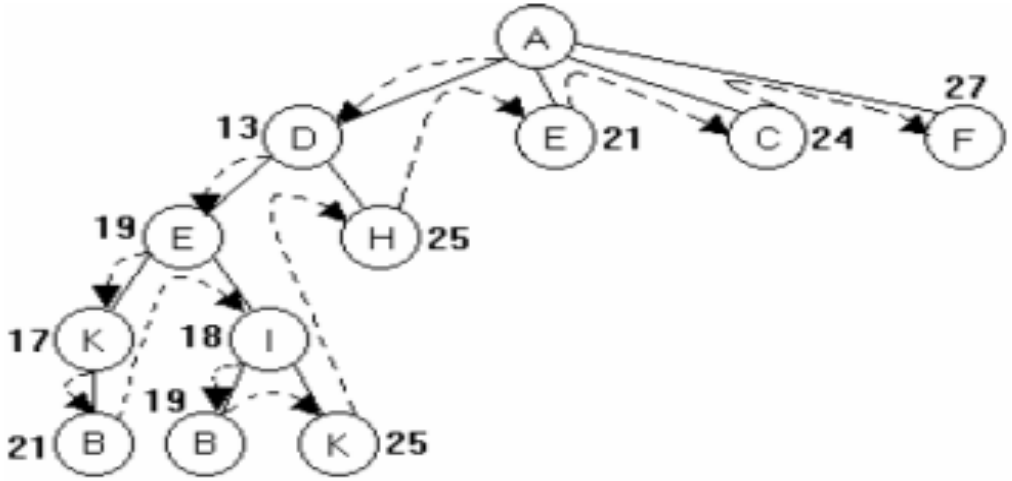
Không gian trạng thái với hàm đánh giá

3.1.2. THUẬT TOÁN TÌM KIẾM NHÁNH VÀ CẬN

- Từ B, quay về K. Từ K quay về E.
- Từ E, sang I, f(I) = 18 < độ dài tạm thời 21. Sinh ra K, B với f(K) = 25, **f(B) = g(B) = 19 → đường đi tạm thời là 19.**
- Với B, không tìm được điểm nào có chi phí tốt hơn nữa.
- ***Vậy đường đi tối ưu có độ dài 19.***



Không gian trạng thái với hàm đánh giá



3.2. TÌM ĐỐI TƯỢNG TỐT NHẤT

Trên không gian tìm kiếm U, mỗi đối tượng x được xác định với một hàm giá cost(x)→ cần tìm đối tượng mà hàm giá đạt giá trị lớn nhất, gọi là ***đối tượng tốt nhất***.

3.2. TÌM ĐỐI TƯỢNG TỐT NHẤT

- Khác với quá trình tìm kiếm trước, tìm đối tượng tốt nhất x, $x \in U$, chưa xác định được đích của quá trình tìm kiếm. Một số kỹ thuật được sử dụng trong phần này gồm:
  - Kỹ thuật tìm kiếm leo đồi để tìm đối tượng tốt nhất.
  - Kỹ thuật tìm kiếm gradient (gradient search).
  - Kỹ thuật tìm kiếm mô phỏng luyện kim (simulated annealing)

3.2.1. LEO ĐỒI TÌM ĐỐI TƯỢNG TỐT NHẤT

- Kỹ thuật này không khác nhiều so với thuật toán tìm kiếm leo đồi trước.
- Với thuật toán leo đồi, từ trạng thái **u** chuyển sang trạng thái tốt nhất **v** (theo hàm lượng giá). Nếu không đến đích và không "leo" đượ nữa, "quay về" trạng thái trước và leo lên trạng thái tốt nhất còn lại.
- Với kỹ thuật tìm kiếm leo đồi tìm đối tượng tốt nhất, từ trạng thái **u**, chuyển sang trạng thái **v** tốt hơn. Nếu không tìm được thì dừng thuật toán.

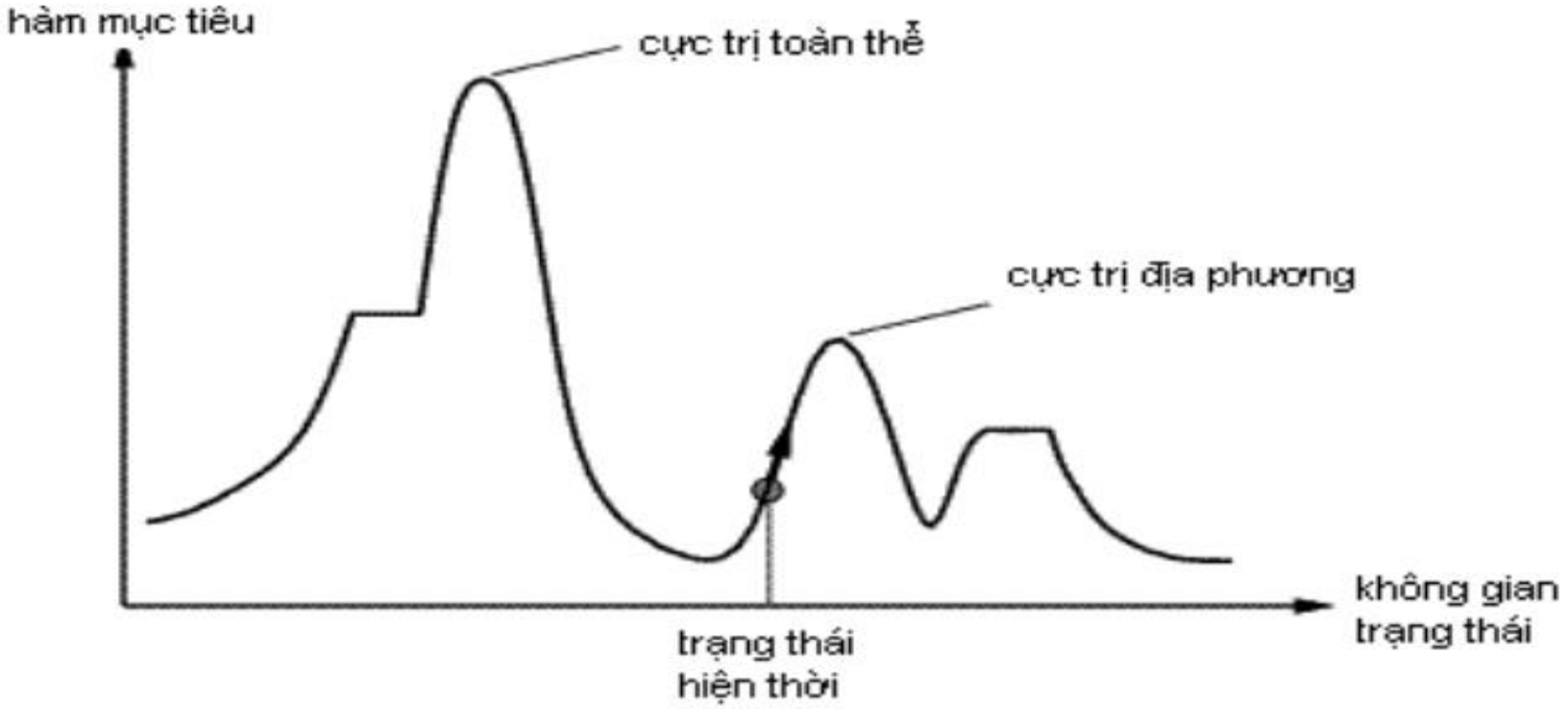
3.2.1. LEO ĐỒI TÌM ĐỐI TƯỢNG TỐT NHẤT

- Trong thủ tục leo đồi dưới đây, biến u lưu đỉnh hiện thời, biến v lưu đỉnh tốt nhất (cost(v) nhỏ nhất) trong các đỉnh ở lân cận u. Khi thuật toán dừng, biến u sẽ lưu đối tượng tìm được

procedure *Hill_Climbing;*
begin
1. u ← một đối tượng ban đầu nào đó;
*2.***loop do**
2.1 v← đối tượng tốt nhất (được xác định bởi hàm giá)trong lân cận u;
2.2 **if** *cost(u) < cost(v)* **then** *u ←v else* **stop**;
end;

3.2.1. LEO ĐỒI TÌM ĐỐI TƯỢNG TỐT NHẤT

Yếu điểm: phụ thuộc vào trạng thái khởi đầu, có thể bị tắc tại cực trị địa phương. (Vấn đề ridge, valley).



Khắc phục: Có thể lặp lại quá trình leo đồi với nhiều điểm xuất phát khác nhau

3.2.2. TÌM KIẾM GRADIENT

- **Ý tưởng**: Tìm kiếm gradient là kỹ thuật tìm kiếm leo đồi để tìm giá trị lớn nhất (hoặc nhỏ nhất) của **hàm khả vi liên tục f(x)** trong không gian các vector thực n-chiều.
- Trong lân cận đủ nhỏ của điểm $x = (x_1, x_2, \ldots, xn)$, hàm f tăng nhanh nhất theo hướng gradient, với hướng được xác định:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

3.2.2. TÌM KIẾM GRADIENT

Chọn bước tiếp theo sao cho:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

với thay đổi nhỏ của **x** ta có thể xấp xỉ hàm $F(\mathbf{x})$:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

trong đó

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_k}$$

Muốn hàm giảm thì:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0$$

Giảm nhiều nhất:

$$\mathbf{p}_k = -\mathbf{g}_k$$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k}$$

3.2.2. TÌM KIẾM GRADIENT

procedure *Gradient_Search*;
begin
$x \leftarrow$ *điểm xuất phát nào đó;*
repeat
$x \leftarrow x + \alpha \nabla f(x)$;
until $|\nabla f| < \varepsilon$;
end;

α là hằng số dương nhỏ xác định tỉ lệ của các bước, ε là hằng số dương nhỏ xác định tiêu chuẩn dừng. Bằng cách lấy các bước đủ nhỏ theo hướng của vectơ gradient chúng ta sẽ tìm được điểm cực đại địa phương, đó là điểm mà tại đó ∇f = 0, hoặc tìm được điểm rất gần với cực đại địa phương

3.2.2. TÌM KIẾM GRADIENT (VÍ DỤ)

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \alpha = 0.1$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \qquad \mathbf{g}_0 = \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.08 \end{bmatrix}$$

3.2.3. TÌM KIẾM MÔ PHỎNG LUYỆN KIM

Ý tưởng:

Với phương pháp tìm kiếm leo đồi không đảm bảo cho việc tìm kiếm nghiệm tối ưu toàn cục. Để có được nghiệm tối ưu toàn cục, kỹ thuật leo đồi sử dụng việc xuất phát từ lựa chọn ngẫu nhiên, lặp nào đó.

Tư tưởng chính của mô phỏng luyện kim là cho phép chọn cả lựa chọn xấu với sác xuất nào đó.

3.2.3. MÔ TẢ THUẬT TOÁN TÌM KIẾM MÔ PHỎNG LUYỆN KIM

Giả sử đang ở trạng thái **u** nào đó. Chọn ngẫu nhiên trạng thái **v** trong lân cận **u**.

- Nếu **v** tốt hơn **u**: sang **v**.
- Nếu **v** không tốt hơn **u**: chỉ sang v với xác suất nào đó.

Xác suất này giảm theo hàm mũ của "độ tồi" của trạng thái **v**. Xác suất phụ thuộc vào tham số **T** (nhiệt độ), **T** càng lớn khả năng sang trạng thái tồi càng cao.

Xác suất được xác định:

$$e^{\frac{\Delta}{T}}$$

$$với\ \Delta = \ cost(v) - cost(u)$$

3.2.3. CÀI ĐẶT THUẬT TOÁN TÌM KIẾM MÔ PHỎNG LUYỆN KIM

```
Procedure Simulated_Anneaning;
Begin
    t ← 0;
    u ← trạng thái ban đầu nào đó;
    T ← nhiệt độ ban đầu;
    repeat
            v ← trạng thái được chọn ngẫu nhiên trong
    lân cận u;
            if cost(v) > cost(u) then u ← v;
            else u ← v với xác suất e^(Δ/T);
            T ← g(T,t);
            t ← t + 1;
    until T đủ nhỏ;
End;
```

Chú ý:

g(T,t) thỏa mãn điều kiện

g(T,t) < T

với mọi t.

Hàm này xác định tốc độ giảm nhiệt độ.

3.2.3. ĐÁNH GIÁ THUẬT TOÁN TÌM KIẾM MÔ PHỎNG LUYỆN KIM

- Trong thủ tục trên, hàm g(T, t) thỏa mãn điều kiện g(T, t) < T với mọi t, nó xác định tốc độ giảm của nhiệt độ T.
- Người ta chứng minh được rằng, nếu nhiệt độ T giảm đủ chậm, thì thuật toán sẽ tìm được **nghiệm tối ưu toàn cục**. Thuật toán mô phỏng luyện kim đã được áp dụng thành công cho **các bài toán tối ưu cỡ lớn**.

3.1. TÌM KIẾM MÔ PHỎNG SỰ TIẾN HÓA

THUẬT TOÁN DI TRUYỀN

- Được giới thiệu bởi John Holland năm 1975 cho phép thực hiện tìm kiếm ngẫu nhiên.
- Nó mã hóa lời giải tiềm năng của bài toán bằng các nhiễm sắc thể
- Lưu trữ một quần thể các lời giải tiềm năng.
- Thực hiện các phép toán di truyền để phát sinh các cá thể mới đồng thời áp dụng chọn lọc tự nhiên trên các lời giải

THUẬT TOÁN DI TRUYỀN

TTDT bắt chước sự *chọn lọc tự nhiên* và *di truyền*:

– *Chọn lọc tự nhiên:* các cá thể khoẻ, có khả năng thích nghi tốt với môi trường sẽ được tái sinh và nhân bản ở các thế hệ sau

– *Di truyền:* Trong quá trình sinh sản, các cá thể con thừa hưởng các phẩm chất của cha mẹ và có những đột biến.

 Mỗi cá thể được mã hoá bởi một cấu trúc DL mô tả cấu trúc gien của cá thể đó, gọi là *nhiễm sắc thể*.

 Một *thế hệ* là một quần thể ứng với một giai đoạn phát triển.

 TTDT bắt chước chọn lọc tự nhiên và di truyền để biến đổi các thế hệ.

THUẬT TOÁN DI TRUYỀN-
CÁC TOÁN TỬ BIẾN ĐỔI CÁC THẾ HỆ

- **Toán tử tái sinh**: các cá thể tốt được lựa chọn để đưa vào thế hệ sau, sự chọn lọc dựa vào độ thích nghi với môi trường.
- **Toán tử lai ghép**: hai cá thể cha mẹ trao đổi gen để tạo ra hai cá thể con.
- **Toán tử đột biến**: Một cá thể thay đổi một số gen để tạo thành cá thể mới

THUẬT TOÁN DI TRUYỀN-
CÁC TOÁN TỬ BIẾN ĐỔI CÁC THẾ HỆ

Một giải thuật di truyền đơn giản bao gồm các bước sau:

Bước 1: Khởi tạo một quần thể ban đầu gồm các chuỗi nhiễm sắc thể.

Bước 2: Xác định giá trị mục tiêu cho từng nhiễm sắc thể tương ứng.

Bước 3: Tạo các nhiễm sắc thể mới dựa trên các toán tử di truyền.

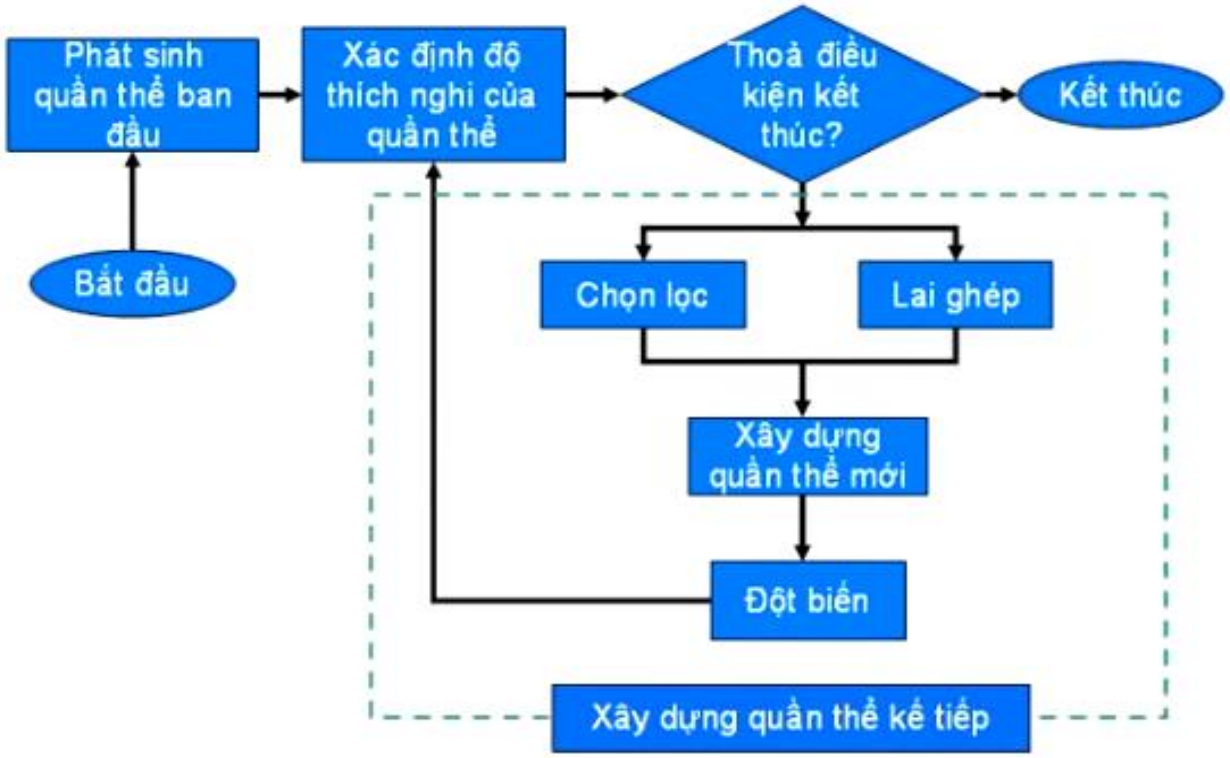
Bước 4: Xác định hàm mục tiêu cho các nhiễm sắc thể mới và đưa vào quần thể.

Bước 5: Loại bớt các nhiễm sắc thể có độ thích nghi thấp.

Bước 6: Kiểm tra thỏa mãn điều kiện dừng. Nếu điều kiện đúng, lấy ra nhiễm sắc thể tốt nhất, giải thuật dừng lại; ngược lại, quay về bước 3.

THUẬT TOÁN DI TRUYỀN- CÁC TOÁN TỬ BIẾN ĐỔI CÁC THẾ HỆ

Procedure Genetic-Algorithm;
Begin
t ← 0;
Khởi tạo thế hệ ban đầu P(t)
Đánh giá P(t) (dựa vào hàm thích nghi);
repeat
t ← t + 1;
Sinh ra thế hệ mới P(t) từ P(t-1) bởi:
Chọn lọc
Lai ghép
Đột biến;
Đánh giá P(t);
until điều kiện kết thúc được thoả mãn;
End;



TOÁN TỬ CHỌN LỌC- PHÉP TÁI SINH

- **Phép tái sinh** là quá trình các nhiễm sắc thể được sao chép trên cơ sở **độ thích nghi**. **Độ thích nghi** là một hàm được gán giá trị thực, tương ứng với mỗi nhiễm sắc thể trong quần thể. Quá trình này, được mô tả như sau:
- **Xác định độ thích nghi của từng nhiễm sắc thể** trong quần thể ở thế hệ thứ t, lập bảng cộng dồn các giá trị thích nghi (theo thứ tự gán cho từng nhiễm sắc thể). Giả sử, quần thể có n cá thể. Gọi độ thích nghi của nhiễm sắc thể i tương ứng là f_i tổng cộng dồn thứ i là f_{ti} được xác định bởi:

$$f_{ti}=\sum_{j=1}^{i} f_i$$

- Gọi ***Fn*** là tổng độ thích nghi của toàn quần thể. Chọn một số ngẫu nhiên *f* trong khoảng từ 0 tới *Fn*. Chọn cá thể thứ k đầu tiên thoả mãn $f \geq f_{tk}$ đưa vào quần thể mới.

TOÁN TỬ CHỌN LỌC- PHÉP CHỌN

- Phép chọn là quá trình loại bỏ các nhiễm sắc thể kém thích nghi trong quần thể. Quá trình này được mô tả như sau:
- - Sắp xếp quần thể theo thứ tự mức độ thích nghi giảm dần.
- - Loại bỏ các nhiễm sắc thể ở cuối dãy. Giữ lại n cá thể tốt nhất.

TOÁN TỬ GHÉP CHÉO

- **Ghép chéo** là quá trình tạo nhiễm sắc thể mới trên cơ sở các nhiễm sắc thể cha-mẹ bằng cách ghép một đoạn trên nhiễm sắc thể cha-mẹ với nhau. Toán tử ghép chéo được gán với một xác suất p_c. Quá trình được mô tả như sau:
- Chọn ngẫu nhiên một cặp nhiễm sắc thể (cha-mẹ) trong quần thể. Giả sử, nhiễm sắc thể cha-mẹ có cùng độ dài m.
- Tạo một số ngẫu nhiên trong khoảng từ 1 tới m-1 (gọi là điểm ghép chéo). Điểm ghép chéo chia nhiễm sắc thể cha-mẹ thành hai chuỗi con có độ dài *m1*, *m2*. Hai chuỗi con mới được tạo thành là: *m11+ m22 và m21+m12*.
- Đưa hai nhiễm sắc thể mới vào quần thể.

TOÁN TỬ ĐỘT BIẾN

- Đột biến là hiện tượng nhiễm sắc thể con mang một số đặc tính không có trong mã di truyền của cha-mẹ.
- • Chọn ngẫu nhiên một nhiễm sắc thể trong quần thể;
- • Tạo một số ngẫu nhiên k trong khoảng từ 1 tới m, $1 \le k \le m$;
- • Thay đổi bit thứ k. Đưa nhiễm sắc thể này vào quần thể để tham gia quá trình tiến hoá ở thế hệ tiếp theo.

VÍ DỤ

- Giải phương trình bậc 2: **$x^2=64$**
- Xác định kích thước quần thể n=4
- Chọn phương pháp mã hóa nghiệm:
  - Xác định nghiệm nguyên trong miền giá trị từ 0- 31
  - Mã hóa theo chuỗi nhị phân : số bit mã hóa: 5
- Lựa chọn hàm thích nghi:
  - Hàm thích nghi **$F(x)= 1000- x^2+64$**, chọn nghiệm có hệ số thích nghi sấp sỉ 1000

VÍ DỤ

- Phát sinh tập quần thể ban đầu:

Stt	Nhị phân	Nghiệm
1	00100	4
2	10101	21
3	01010	10
4	11000	24

VÍ DỤ

- Tính hệ số thích nghi cho quần thể:

Stt	Nhị phân	Nghiệm	X^2-64	Hệ số thích nghi
1	00100	4	-48	1048
2	10101	21	377	623
3	01010	10	36	964
4	11000	24	512	488

VÍ DỤ

- Tính hệ số thích nghi cho quần thể:

Stt	Nhị phân	Nghiệm	X^2-64	Hệ số thích nghi
1	00100	4	-48	1048
2	10101	21	377	623
3	01010	10	36	964
4	11000	24	512	488

VÍ DỤ

- Chọn lọc nghiệm và lai ghép:
- Chọn nghiệm 4 và 10 để tiến hành lai ghép với xác suất pc và vị trí position= 2
- 4 001**00** → **010****00** **8**
- 10 010**10** → **001****10** **6**

VÍ DỤ

- Đột biến một cá thể.
- Với một xác suất p_m đột biến lời giải thứ 4 với vị trí pos=4
- **00110 6 → 01110 14**

VÍ DỤ

- Tính lại hệ số thích nghi cho nghiệm mới và tiến hành chọn lọc:
- Chọn nghiệm số 3 có hệ số thích nghi = 1000 vừa khít với hệ số thích nghi

Stt	Nhị phân	Nghiệm	X^2-64	Hệ số thích nghi
1	00100	4	-48	1048
2	01010	10	36	964
3	01000	8	0	1000
4	01110	14	132	868