THUẬT GIẢI Ơ-RÍT-TÍC

1. Ơ-rít-tíc

Ơ-rít-tíc – heuristic[1]*,* có nguồn gốc trong tiếng Hy Lạp, có nghĩa khám phá, dùng mô tả một quy tắc hợp lý hoặc một phương pháp giải quyết vấn đề dựa vào kinh nghiệm cá nhân, chẳng hạn như quá trình loại suy hay phương pháp thử-sai. Một cách bình dân, có thể hiểu heuristic như "*mẹo*" để giải quyết một vấn đề được cho. Trong khoa học máy tính, ta có thể hiểu thuật ngữ heuristic là *mẹo giải* bên cạnh thuật ngữ algorithm – *thuật giải* mà ta đã sử dụng trong các chương trước.

Nghe bất ngờ nhưng thực ra heuristics có thể biết đến như "*quy tắc ngón cái* – rule of thumb", là các nguyên tắc không hướng đến sự chính xác hoặc đáng tin cậy trong mọi tình huống, nhưng lại có ứng dụng rộng rãi, như ngón cái tham gia hầu hết mọi hoạt động của bàn tay. Nguyên tắc ngón cái đề cập đến một quy trình dễ học và dễ áp dụng, dựa trên kinh nghiệm thực tế hơn là lý thuyết.

Trong khoa học máy tính, heuristics thường được sử dụng để giải các bài toán chưa có thuật giải hay có thuật giải nhưng độ phức tạp thời gian quá lớn khiến thuật giải đó không thể áp dụng trong thực tế. Các thuật giải tìm kiếm BFS, DFS, trong lý thuyết đồ thị, chính là các heuristics mà "mẹo" ở đây là thác triển cây tìm kiếm theo chiến lược chiều rộng hay chiều sâu trước, sử dụng phương pháp *thử sai và quay lui*. Hay *phương pháp đạo hàm* – gradient, được dùng phổ biến để giải bài toán tối ưu hàm số thực, chính là heuristic với "mẹo" là *leo đồi* theo hướng dốc nhất, hướng của đạo hàm. Heuristic "leo đồi" này sẽ được sử dụng lại trong bài này để phát triển một mô hình được gọi là *mạng nơ-ron – neural network*. Bên cạnh mạng nơ-ron, *thuật giải di truyền – genetic algorithm*, một phương pháp rất mạnh có thể áp dụng để giải các vấn đề tối ưu trong thực tiễn, phương pháp tìm kiếm dựa trên heuristic di truyền và sinh tồn.

2. Mạng nơ-ron

2.1. Hàm NN

Mạng nơ-ron có thể xem như một ánh xạ NN: $R^n \to R^m$ và được biểu diễn bằng một hay nhiều ma-trận các giá trị thực, như ma-trận $W \in R^{(n+1)\times m}$ trong phương pháp hồi quy logistic.

Không mất tính tổng quát và cũng là để đảm bảo tính hiệu quả của các chương trình máy tính, mạng nơ-ron được xem là ánh xạ xác định trên một miền con D, $NN_{A,B}$: $D \to R$ với $D \subset R^n$, $R \subset R^m$, được biểu diễn bởi 2 ma-trận $A \in R^{(n+1)\times K}$, $B \in R^{(K+n+1)\times m}$, và được xác định bởi:

[1] Trong tài liệu này, chúng tôi sẽ sử dụng từ gốc heuristic thay cho phiên âm ơ-rít-tic

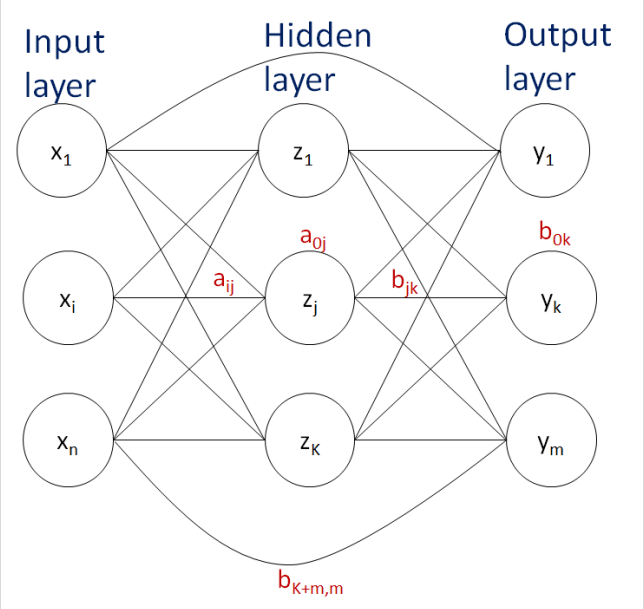
$NN_{A,B}(x) = NN_{A,B}(x_1, \dots, x_n) = (y_1, \dots, y_m)$, trong đó

$y_k = S(V_k), V_k = \sum_{j=1}^{K} b_{jk} z_j + \sum_{j=K+1}^{K+n} b_{jk} x_{j-K} + b_{0k}, \forall k = 1,2, \dots, m,$

$z_j = S(U_j), U_j = \sum_{i=1}^{n} a_{ij} x_i + a_{0j}, \forall j = 1,2, \dots, n,$

với S là *hàm kích hoạt* có dạng hình chữ S, gọi là hàm sigmoid.

Hình sau hình ảnh hóa ánh xạ $NN_{A,B}$ bằng một kiến trúc mạng nơ-ron 2 tầng (*layer*), là một đồ thị có trọng với các nút, được gọi là các *nơ-ron*, và các *cung*, là các dây thần kinh nối 2 nơ-ron. Mỗi nơ-ron với nhãn là các tham số và có 1 trọng lượng, hoặc nhận vào hoặc do tính được. Mỗi cung có một trọng số biểu diễn quan hệ giữa 2 nơ-ron liên quan. Mạng gồm 3 lớp nơ-ron và 2 tầng trọng số. Lớp thứ nhất, gọi là lớp nhập, và lớp thứ 3, lớp xuất, được xem như các nơ-ron ngoại biên, nhận tín hiệu từ môi trường vào lớp nhập, và phản ứng lại qua lớp xuất. Lớp thứ 2, gọi là lớp ẩn là các nơ-ron trung gian để điều chỉnh tín hiệu từ lớp nhập chuyển tới lớp xuất. Mạng nơ-ron nhân tao giả sử các nơ-ron trong cùng lớp không liên kết nhau. Số nơ-ron lớp nhập (**n**) và lớp xuất (**m**) được xác định theo bài toán. Số nơ-ron (các) lớp ẩn do người thiết kế ánh xạ NN xác định tùy thuộc độ phức tạp của bài toán.



Mạng nơ-ron 2-lớp

$NN_{A,B}(X)$:

#input x = $(1, x_1, \dots, x_n)$

#output y = $(y_1, \dots, y_m)$

(1) $\forall j = 0, 1, \dots, K$:

$$u_j = \begin{cases} 1, & j = 0 \\ \sum_{i=0}^{n} x_i a_{ij}, & j = 1, \dots, n \end{cases}$$

$z_j = S(u_j)$

(2) $\forall k = 1, \dots, m$:

$v_k = \sum_{j=0}^{K} z_k b_{jk} + \sum_{j=K+1}^{K+n} x_{j-K} b_{jk}$

$y_k = S(v_k)$

(3) return y = $(y_1, \dots, y_m)$

Thuật giải tính NN(x)

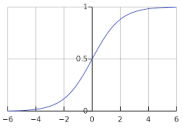
2.2. Hàm kích hoạt

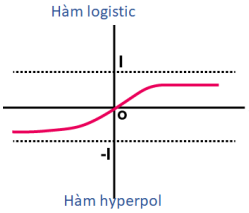
Hàm kích hoạt, được áp dụng cho các nơ-ron ẩn và nơ-ron xuất, **S(u)**, là một hàm có dạng chữ S, gọi là hàm *sigmoid*, thỏa các điều kiện:

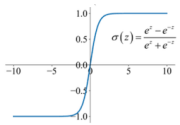
- S(u) bị chặn: |S(u)| < U, U > 0, ∀u.
- S(u) đơn điệu tăng: S(u) < S(u'), ∀u < u'.
- S(u) liên tục và trơn: ∃S'(u) và S'(u), S'(u') khác nhau rõ rệt nếu u khác u'.

Tùy thuộc miền giá trị đầu ra của $NN_{A,B}$ mà có thể sử dụng các hàm sigmoid sau:

- Hàm logistic
  $S(u) = g(u) = \frac{1}{1+\frac{1}{e^u}} \in [0,1]$.
  Hàm này có đạo hàm g'(u)=g(u)(1-g(u))
- Hàm hyperpol
  $S(u) \equiv h(u) = \frac{1-e^{-u}}{1+e^{-u}} = 2g(u) - 1 \in [-1,1]$.
- Hàm tang-hyperpol
  $S(u) \equiv tan\, h(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} \in [-1,1]$.
  (hàm tanh(u) tiến tới giới hạn nhanh hơn hàm h(u)).


Hàm logistic


Hàm hyperpol


Hàm tan hyperpol

2.3. Máy học

Như đã biết, ánh xạ $NN_{A,B}$ 2-lớp được xác định hoàn toàn khi xác định được các ma-trận trọng số $A \in R^{(n+1)\times(K+1)}$ và $B \in R^{(K+n+1)\times m}$. Trong phân tích dữ liệu, hai ma-trận này được xây dựng từ tập dữ liệu thu thập được, gọi là *học từ dữ liệu* hay *học giám sát*. Một cách hình thức, học giám sát được mô tả hình thức như sau.

Bài toán học học. Cho $\Omega = \{(x^i, t^i) = (1, x_1^i, \dots, x_n^i, t^1, \dots, t_m^i) \in D_1 \times D_2 \subset R^{1+n+m}\}_{i=1}^N$, xây dựng ánh xạ $NN_{A,B}: D_1 \to D_2$ sao cho $y^i = NN_{A,B}(x^i) \to \|t^i - y^i\| < \varepsilon, \forall i = 1, \dots, N$, và $\varepsilon > 0$.

2.3.1. Hàm lỗi – **error function**

Gọi $E_i(A,B) = \frac{1}{2}\sum_{k=1}^m (t_k^i - y_k^i)^2$ là sai số của mẫu thứ i, và

$E(A,B) = \frac{1}{NK}\sum_{i=1}^N E_i(A,B)$ là hàm lỗi của ánh xạ $NN_{A,B}$.

Bài toán học được đưa về giải bài toán tối ưu sau:

$\underset{A,B}{argmin}\, E(A,B)$.

2.3.2. Phương pháp đạo hàm – **gradient**

Sử dụng heuristic "leo đồi" theo hướng "dốc" nhất để giải gần đúng phương trình đạo hàm hàm lỗi bằng 0 để tìm các ma-trận trọng số. Phương pháp này được gọi là *phương pháp giảm đạo hàm – gradient descent*, viết tắt ***GD***. ***GD*** sử dụng *quy tắc chuỗi đạo hàm* để tính giá trị đạo hàm của hàm lỗi tại mỗi bước lặp của nó.

Quy tắc chuỗi đạo hàm. Để tính đạo hàm hàm hợp f(x), ta tính các thành phần của chuỗi $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial x_2} \dots \frac{\partial x_p}{\partial x}$.

Đặt $\Delta(x) = \frac{\partial f}{\partial x}$, phương pháp giảm đạo hàm được mô tả gồm 3 bước như trong phương pháp hồi quy logistic.

Bước 0. x = x(0) được khởi tạo ngẫu nhiên.

Bước 1. Tính Δ(x).

Bước 2. Cập nhật x $\leftarrow$ x - $\alpha\Delta$(x). $\alpha \in (0, 1]$ được gọi là hệ số học

(quay lại Bước 1)

Bây giờ, ta sẽ áp dụng quy tắc chuỗi đạo hàm để tính đạo hàm hàm lỗi theo các trọng số a_{ij} và b_{jk}. Để đơn giản hóa công thức, ta quy ước:

- bỏ qua các chỉ số. Chẳng hạn, thay vì cụ thể giá trị nút xuất thứ k, y_k, ta viết đơn giản là y; hay trọng số b_{jk}, ta viết b_j.
- hàm sigmoid là hàm logistic g(x), khi đó g'(x) = g(x)(1 – g(x)).

2.3.3. Trọng số tầng xuất – **B = [b_{jk}]**

Đạo hàm hàm lỗi theo trọng số lớp ẩn (b) là chuỗi:

$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial b_j}$.

Tính chi tiết các thành phần.

- $\frac{\partial E}{\partial y}$

Nếu nút xuất đang xét có giá trị tính được là y và giá trị đúng của nó là t, sai số bình phương sẽ là

$E(y) = \frac{1}{2}(y-t)^2$, suy ra

$\frac{\partial E}{\partial y} = y - t$ (**1**).

- $\frac{\partial y}{\partial v}$

Do $y = g(v) = \frac{1}{1+e^{-v}}$, suy ra

$\frac{\partial y}{\partial v} = g(v)\big(1 - g(v)\big) = y(1-y)$ (**2**)

- $\frac{\partial v}{\partial b_j}$

Từ $v = b_0 + \sum_{j=1}^{K} b_j z_j + \sum_{j=K+1}^{j+n} b_j x_{j-K}$, suy ra

$$\frac{\partial v}{\partial b_j} = \begin{cases} 1, & j = 0 \\ y_j, & 0 < j \le K \\ x_{j-K}, & j > K \end{cases} \quad (\mathbf{3})$$

Từ (1), (2), (3), đặt

$p = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v} = (y-t)y(1-y)$, (**4**)

thì đạo hàm riêng tương ứng với trọng số b_j là

$$\Delta(b_j) = \frac{\partial E}{\partial b_j} = \begin{cases} p, & j = 0 \\ pz_j, & 0 < j \le K \\ px_{j-K}, & j > K \end{cases} \quad (\mathbf{*})$$

2.3.4. Trọng số tầng ẩn – $\mathbf{A = [a_{ij}]}$

Từ công thức chuỗi đạo hàm

$\frac{\partial E}{\partial a} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial u}\frac{\partial u}{\partial a}$, các thành phần được tính chi tiết như sau.

- $\frac{\partial E}{\partial z}$

$\frac{\partial E}{\partial z} = \sum_{k=1}^{K} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial z}$ (**5**)(lưu ý là đã lược bỏ chỉ số j).

Các thành phần trong tổng (trong dấu Σ) sẽ được tính như sau:

(.) Hai thừa số đầu của tích được tính tương tự như đã tính trong phần trên. Riêng thành phần cuối, thêm chỉ số k cho p được đặt ở (**4**), p_k để xác định p này thuộc nút xuất nào, (**5**) được viết lại:

$\frac{\partial E}{\partial z} = \sum_{k=1}^{m} p_k \frac{\partial v_k}{\partial z}$ (**6**)

Với $\frac{\partial v_k}{\partial z} = b_k$, (6) được viết lại thành

$\frac{\partial E}{\partial z} = \sum_{k=1}^{m} p_k b_k$ (**7**)

Tương tự cách tính cho trong số b_{jk}, hai thừa số sau được tính:

- $\frac{\partial z}{\partial u}$

$\frac{\partial z}{\partial u} = z(1-z)$ (**8**)

- $\frac{\partial u}{\partial a}$

$\frac{\partial u}{\partial a_i} = \begin{cases} 1, & i = 0 \\ x_i, & 0 < i \le n \end{cases}$ (**9**).

Đặt

$q = \frac{\partial E}{\partial z}\frac{\partial z}{\partial u}$, (**10**)

Từ (7)-(10), đạo hàm riêng tương ứng với trọng số a_i là:

$\Delta(a_i) = \frac{\partial E}{\partial a_i} = \begin{cases} q, & i = 0 \\ qx_i, & 0 < i \le n \end{cases}$ (**)

Từ (*) và (**), công thức cập nhật trọng số áp dụng cho thuật giải **DG**, gọi là quy tắc học là

Quy tắc học. $w = w - \alpha\Delta(w)$, với $w = a_{ij}\ hay\ w = b_{jk}$ và

$$\Delta(b_{jk}) = \frac{\partial E}{\partial b_{jk}} = \begin{cases} p, & j = 0 \\ pz_j, & 0 < j \le K \\ px_{j-K}, & j > K \end{cases}, \quad k = 0, 1, \ldots, m, m+1, \ldots, m+n$$

$$\Delta(a_{ij}) = \frac{\partial E}{\partial a_{ij}} = \begin{cases} q, & i = 0 \\ qx_i, & 0 < i \le n \end{cases}, \quad j = 0, 1, \ldots, K.$$

2.3.4. Mã giả thủ tục tính ngược đạo hàm từ kết quả NN – **back**

```
back #tính đạo hàm hàm lỗi

FOR j in range(K+1):

   q[j] = 0 #khởi tạo ∂E/∂u = 0

FOR k in range(m+1):
```

```
   p[k] = (y[k]-t[k])*y[k]*(1-y[k])
   db[0][k] = db[0][k] + p[k] #Δ(bjk)
   For j in range(K+1):
      db[j][k] = db[j][k] + p[k]*z[j]
      q[j] = q[j] + p[k]*z[k]
   FOR i in range(K+1,K+n):
      db[j][k] = db[j][k] + p[k]*x[j-K]
FOR j in range(K+1):
   q[j] = q[j]*z[j]*(1 – z[j])
   da[0][j] = da[0][j] + q[j] #Δ(aij)
   FOR i in range(1,n+1):
      da[i][j] = da[i][j] + q[j]*x[i]
```

Bài tập. Cho mạng nơ-ron 3 lớp cài đặt hàm AND, OR, XOR.

a) Chỉ định các ma-trận A và B.

b) Viết chương trình cho mạng học các hàm này.

3. Thuật giải di truyền

3.1. Mê-ta ơ-rít-tic

Rất khó để dịch thuật ngữ *meta-heuristic*, nhưng các thuật ngữ có gắn tiếp đầu ngữ "*meta*" để diễn tả meta-heuristic là một phương pháp mạnh, rất mạnh trong khoa học máy tính nói chung và trong trí tuệ nhân tạo nói riêng. Tuy nhiên, cần lưu ý (**i**) một phương pháp mạnh, tổng quát thường không hiệu quả nếu chỉ được áp dụng một cách máy móc, và (**ii**) như hầu hết các kỹ thuật của trí tuệ nhân tạo, bài toán có thể được giải với giải pháp hay lời giả có thể chấp nhận được nhưng không thể hoặc rất khó giải thích tại sao lại (làm) như thế.

Thuật giải di truyền – Genetic Algorithm, **GA**, được xem là một meta-heuristics và là thuật giải "rất" mạnh trong khoa học máy tính. Thực vậy, bằng cách mô phỏng sự "*tiến hóa tự nhiên*" với heuristic "tham lam - greedy" theo nghĩa sinh tồn, **GA** có thể giải các bài toán tối ưu nếu chúng có thể được biểu diễn di truyền. Mặt khác, hầu hết các vấn đề trong thực tế đều có thể được xem như tìm kiếm giải pháp tối ưu và một lời giải tốt là lời giải có thể được sử dụng trong đời sống.

3.2. Các thành phần của thuật giải di truyền

GA mượn các thuật ngữ của di truyền Mendel và các khái niệm tiến hóa của Darwin để phát triển một lược đồ - scheme tổng quát cho GA.

3.2.1. Biểu diễn di truyên – **data structure**

. Cá thể - individual. Mỗi giải pháp khả thi được xem như một cá thể trong tập các cá thể "sống".

. Nhiễm sắc thể - chromosome. Là một biểu diễn bên trong máy tính của một cá thể. Thuật ngữ nhiễm sắc thể và cá thể có thể được dùng chung cho giải pháp khả thi.

. Gien/gen – gene. Là thành phần cơ bản, không thể chia cắt, cấu thành nhiễm săc thể.

. Quần thể - population. Tập các cá thể "sống". Thường kích thước quần thể nhân tạo là một hằng số.

. Độ thích nghi – fitness. Là một giá trị, thường là giá trị của hàm tối ưu, biểu diễn "sức sống" của cá thể trong quần thể có số cá thể là hằng số.

Tùy thuộc bài toán mà các cá thể được biểu diễn khác nhau. Trong đa số trường hợp, nhiễm sắc thể được biểu diễn bằng chuỗi nhị phân. Cách biểu diễn này giúp cho việc cài đặt các phép toán di truyền dễ dàng hơn.

3.2.2. Các phép toán di tuyền – **genetical operation**

. Khởi tạo quần thể - initiation. Là thủ tục phát sinh tập các lời giải khả thi mô phỏng quần thể gồm các cá thể sống. Kích thước quần thể thường là hằng số.

. Phép chọn – selection. Là phép toán tựa ngẫu nhiên mô phỏng tiến trình *chọn – selection*, để tuyển chọn một số cá thể để tạo ra thế hệ mới. Mặc dù có tính ngẫu nhiên nhưng việc chọn cũng dựa trên độ thích nghi của cá thể.

. Phép lai – crossover. Là thủ tục mô phỏng phép sinh sản hữu tính để tạo ra các cá thể mới từ 2 cá thể được chọn bởi phép chọn - selection.

. Phép đột biến – mutation. Là thủ tục mô phỏng phép sinh sản vô tính hoặc phép đột biến tự nhiên để tạo ra cá thể mới từ các thể được chọn.

. Phép chọn lọc – excommunication. Là thủ tục mô phỏng quá trình tiến hóa bằng cách loại bỏ các cá thể "yếu", là những cá thể có độ thích nghi với môi trường kém nhất trong quần thể.

Tùy thuộc biểu diễn di truyền mà các phép toán di truyền có thể được cài đặt.

3.2.3. Lược đồ - **scheme**

Quy trình chung sử dụng thuật giải di truyền

I. Biểu diễn di truyền và xác định kích thước quần thể.

II. Cài đặt các phép toán di truyền.

III. Thuật giải di truyền.

#N – kích thước quần thể. r1 – tỷ lệ sinh sản. r2 – tỷ lệ đột biến.

(**1**) Ω ← initiation(N) #tạo quần thể ban đầu có n cá thể sống.

(**2**) FOREVER

(**2.1**) Ω1 ← {f, m ← selection(Ω,r1); children ← crossover(f,m)}

(**2.2**) Ω2 ← {ind ← selection(Ω, r2); new_ind ← mutation(ind)}

(**2.3**) Ω ← excommunication(Ω ∪ Ω1 ∪ Ω2, n).

(**2.4**) IF breakpoint THEN return max_fitness(Ω) ELSE goto (2).

Lưu ý, di truyền hay tiến hóa không có điểm dừng. Tuy nhiên, vì là chương trình máy tính mô phỏng tiến hóa tự nhiên nên chương trình phải có điểm dừng - breakpoint. Thông thường, điều kiện dừng được thiết lập khi đạt số lần lặp, gọi là số thế hệ max_epoch xác định trước.

4. Lập trình tiến hóa

Ta sẽ áp dụng lược đồ học **GA** để giải một số bài toán minh họa, qua đó học cách sử dụng thuật giải di truyền giải các bài toán thực tế. Kỹ thuật dùng GA giải quyết vấn đề, ta gọi là *lập trình tiến hóa*.

4.1. Quy hoạch nguyên và thuật giải di truyền

Quy hoạch nguyên – integer programming là lớp các bài toán có thể được biểu diễn dưới dạng tối ưu hóa một hàm tuyến tính thỏa các ràng buộc tuyến tính trên các biến kiểu nguyên. Chẳng hạn các bài toán lập lịch sản xuất, lập thời khóa biểu, … là những bài toán quy hoạch nguyên.

Quy hoạch nguyên tổng quát là bài toán *NP-khó – NP-hard*, tuy nhiên, trong nhiều trường hợp cụ thể, một số bài toán quy hoạch nguyên có thể giải được trong thời gian hợp lý dùng thuật giải di truyền với cấu trúc di truyền là chuỗi bit. Ta xem xét giải bài toán đơn giản sau.

Bài toán lập kế hoạch sản xuất. Để mở rộng sản xuất, công ty cần trang bị thêm một số máy dập và máy tiện. Mỗi máy dập dự kiến sinh lãi 70USD/ngày và mỗi máy tiện cỡ 60USD/ngày. Ngân sách chỉ có thể giới hạn tới 30 ngàn đô-la Mỹ và diện tích có thể sử dụng trong xưởng có $12m^2$; và máy dập giá 6 ngàn đô-la Mỹ, chiếm $2m^2$ diện tích, máy tiện chiếm $3m^2$, giá 5 ngàn đô-la Mỹ. Câu hỏi đặt ra là mua mỗi loại bao nhiêu máy?

Có nhiều yếu tố khiến ta có thể nghĩ đến việc sử dụng GA để giải những bài toán thực tế kiểu này, nhưng yếu tố quan trọng nhất chính là nó *không yêu cầu cấp bách*, nghĩa là yếu tố thời gian không quan trọng bằng yêu tố chi phí. Nói cách khác, có thể giải nhiều giờ nhưng tiết kiệm được nhiều tiền hơn là giải trong vài phút nhưng lại tốn tiền. Bài toán này đơn giản và có nhiều cách khác có thể giải. Ở đây ta chỉ dùng nó để minh họa các kỹ thuật lập trình tiến hóa cho đơn giản.

Trước tiên, ta biểu diễn bài toán lập kế hoạch chi tiêu trên dưới dạng bài toán tối ưu với các biến nguyên, vì không thể mua nửa hay 1/3 cái máy được. Ta sẽ cực đại hóa lợi nhuận thay vì cực tiểu hóa chi phí vì lợi nhuận mỗi ngày mỗi sinh trong khi chi phí chỉ tiết kiệm được đúng 1 lần. Bài toán quy hoạch nguyên tương ứng như sau:

Gọi x_1, x_2 lần lượt là số máy dập và máy tiện. Thì . Lợi nhuận mỗi ngày: $70x_1 + 60x_2$. . Chi phí phải trả: $6x_1 + 5x_2$. . Diện tích bị chiếm: $2x_1 + 3x_2$.	$\max_{(x_1,x_2)\in N^2} f(x_1,x_2) = 70x_1 + 60x_2$ s.t. (subject to) $g_1(x_1,x_2) = 6x_1 + 5x_2 \le 30\ (ngàn\ usd)$, $g_2(x_1,x_2) = 2x_1 + 3x_2 \le 12\ (m^2)$ $x_1, x_2 \ge 0$.

- **Biểu diễn di truyền và độ thích nghi**

. Với ràng buộc ngân sách như mô tả, có thể sử dụng 3 bit để biểu diễn từng biến x_1 và x_2. Như vậy, một cá thể (x_1, x_2) được biểu diễn bằng chuỗi nhị phân 6 bit, 3 bit đầu biểu diễn x_1 và 3 bit sau cho x_2. Chẳng hạn s = 010001 tương ứng với (x_1, x_2) = (2, 1). Bằng cách biểu diễn như vậy, ràng buộc x_1, $x_2 \ge 0$ luôn thỏa.

. Ta cần tím giá trị lớn nhất nên độ thích nghi chính nên độ thích nghi fitness(s) = fitness($s_1s_2s_3s_4s_5s_6$) = f(x_1, x_2) = $70x_1 + 60x_2$ với x_1=int($s_1s_2s_3$), x_2=int($s_4s_5s_6$).

Đây chỉ là một cách mã hóa lời giả bằng chuỗi nhị phân. Có thể có nhiều cách khác tùy thuộc "kỹ thuật lập trình tiến hóa" của bạn.

- **Các phép toán di truyền**

Sau đây là mô tả 1 cách cài đặt các phép toán di truyền.

. **initiation(N)**. Sinh ra tập n chuỗi nhị phân, ở đây n = 6 bit.

. **selection()**. Gọi interval là tổng các độ thích nghi của tất cả các cá thể trong quần thể hiện hành, theo tính chất bài toán đang xét, interval là số nguyên không âm. Chia interval thành N đoạn $[a_i, b_i)$ với $a_1 = 0$ và b_N = interval + 1. Hàm selection sinh ngẫu nhiên số nguyên i lớn hơn bằng 0 và nhỏ hơn bằng interval. Cá thể thứ i trong quần thể được chọn.

. **crossover(father, mother)**. Nhận đầu vào là 2 chuỗi n = len(father) bit, father và mother. Hàm crossover trước hết, sinh ra một số nguyên ngẫu nhiên $0 \le p \le n$, gọi là điểm lai; và tao ra (các) cá thể con từ điểm lai p, chẳng hạn, nếu $0 < p < n$, tạo ra child = father[1..p]||mother[p+1..n], ngược lại, có thể xem như cặp này "vô sinh".

. **mutation(individual)**. Nhận đầu vào là chuỗi n = len(individual) bit. Hàm mutation phát sinh một số nguyên $0 \le p \le n$, và nếu $0 < p < n$ thì thay bit thứ p của individual bằng bit bù của nó, $individual_p$ = 2*$individual_i$ – 1. Ngược lại, cá thể này không bị đột biến.

. **excommunication()**. Sắp xếp theo thứ tự giảm dần tất cả các cá thể đang tồn tại, bao gồm cả các cá thể thế hệ trước và các thế hệ mới sinh ra hay bị đột biến và chỉ giữ lại N cá thể đầu tiên.

4.2. Xấp xỉ và thuật giải di truyền

Cho trước độ chính xác $\varepsilon > 0$ cần đạt, có thể chọn số bit thích hợp để có thể biểu diễn được lời giải với độ chính xác ε. Chẳng hạn, xét lại bài toán tối ưu trong bài trước, tìm nghiệm xấp xỉ của hệ

$\begin{cases} F(x,y) = 15.5x^{-0.5} - 8 + 1.3y^{-0.2} - 0.06yx^{-1.08} \\ G(x,y) = 9y^{-0.4} - 5 + 0.5x^{-0.08} - 0.26x^{-1.2} \end{cases}$, $(x,y) \in \left[-\sqrt{7}, \sqrt{5}\right]^2$, với độ chính xác 1/1000 (ε = 0.001).

Với $\varepsilon = \frac{1}{1000}$, bằng cách chia đoạn $\left[-\sqrt{7}, \sqrt{5}\right]$ thành $N = \frac{\sqrt{5}+\sqrt{7}}{\varepsilon}$ đoạn con $[a_i, b_i], a_1 = -\sqrt{7}, b_{N+1} = \sqrt{5}$ liên tiếp, mỗi đoạn có chiều dài $b_i - a_i = \varepsilon$, bài toán tìm nghiệm xấp xỉ trên được đưa về tìm nghiệm nguyên.

Bây giờ, đặt

$$f(x,y) = |F(x,y) + G(x,y)| \ge 0,$$

bài toán tìm nghiệm xấp xỉ được đua về bài toán quy hoạch nguyên

$\min_{(i,j)} f(x_i, y_j)$

s.t. $x_i, y_j \in [-\sqrt{7}, \sqrt{5}]$,

và có thể sử dụng lại các phép toán di truyền trong bài toán trước với biểu diễn di truyền là chuỗi 20 bit, 10 bit đầu cho x và 10 bit sau cho y, với độ thích nghi fitness(s) = fitness($s_1..s_{10}s_{11}..s_{20}$) = 1/f(x,y), với x = int($s_1..s_{10}$), y = int($s_{11}..s_{20}$).

Cách giải vậy chưa thực sự thuyết phục ta vì rõ ràng ta có thể "vét cạn" trên tập hữu hạn {1,…, N} × {1,…, N} mà là N^2 khả năng là ta có thuật giải tất định.

Cách giải khác là mã hóa di truyền bằng chính 2 số thực máy tính (float) x và y và giải bài toán tối ưu:

$$\min_{(x,y)\in R^2} f(x,y) = |F(x,y) + G(x,y)|$$

s.t.

$$-\sqrt{7} \le x \le \sqrt{5}$$
$$-\sqrt{7} \le y \le \sqrt{5}.$$

Với biểu diễn một các thể là một véc-tơ thực 2 thành phần (x, y) ∈ R^2, cần điều chỉnh lại cách tính độ thích nghi cũng như một số phép toán di truyền, cụ thể phép lai và đột biến, tùy thuộc cách định nghĩa độ thích nghi. Chẳng hạn, với định nghĩa

$$fitness(s) = fitness\big((x,y)\big) = \begin{cases} \frac{1}{f(x,y)}, & -\sqrt{7} \le x, y \le \sqrt{5} \\ -\infty, & ngược\ lại \end{cases}.$$

Khi ấy phép lai và phép đột biến có thể sinh ra thế hệ mới là một điểm giữa 2 điểm của thế hệ trước đối với phép lai, và là một điểm ngẫu nhiên với đột biến. Chẳng hạn

. **crossover**((x_f, y_f), (x_m, y_m)) trả về các điểm có dạng (x_c, y_c) = (θx_f + (1-θ)x_m, θy_f + (1-θ)y_m), với θ ∈ [0, 1] được sinh ngẫu nhiên.

. **mutation**(x_ind, y_ind). Sinh một bit b ngẫu nhiên, và 1 giá trị thực v ngẫu nhiên trong miền xác định. Nếu b == 0 thì trả về (v, y_ind), ngược lại (b == 1), trả về (x_ind, v).

4.3. Về tính đúng đắn của thuật giải di truyền

Trước khi tiếp tục với các bài toán khác, ta phát biểu và chứng minh tính đúng đắn của thuật giải di truyền giải bài toán tối ưu số (nguyên hay thực) ta vừa khảo sát trên.

Về sự hội tụ của thuật giải di truyền. Thuật giải di truyền hội tụ về điểm tối ưu toàn cục của bài toán tối ưu được cho.

Thực vậy, không mất tính tổng quát, giả sử bài toán tối ưu là không ràng buộc và

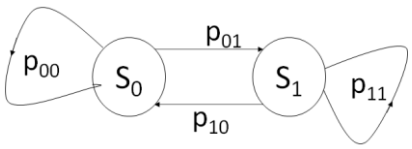
$\max_x f(x)$.

Bằng cách xây dựng một xích Markov 2 trạng thái S_0 và S_1, với ký hiệu x(t) là lời giải tốt nhất của quần thể $\Omega(t)$ ở thế hệ thứ t. Nghĩa là $f(x(t)) = \max\{f(x): x \in \Omega(t)\}$ and $x(t) \in \Omega(t)$.

. S0 = "không đổi" theo nghĩa lời giải tốt nhất ở bước chuyển thứ t cũng là lời giải tốt nhất ở bước chuyên thứ t-1 ngay trước đó, x(t) = x(t-1).

. S1 = "có đổi" theo nghĩa $x(t) \neq x(t-1)$ and $f(x(t)) \geq f(x(t-1))$.

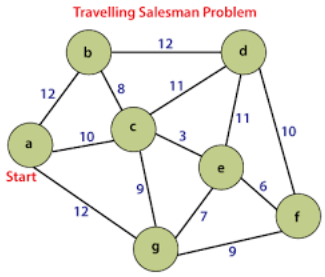
Thì, ma-trận xác xuất chuyển là ma-trận có các phần tử lớn hơn 0, hay là ma-trận chính quy. Nên xích Markov sẽ hội tụ về trạng thái hấp thu.



4.4. Tìm kiếm tối ưu và thuật giải di truyền

Bên cạnh các bài toán tối ưu số (nguyên hay thực), thuật giải di truyền có thể được áp dụng giải các bài toán tối ưu thực tế khác. Với các bài toán tìm kiếm tối ưu như vậy, để sử dụng GA, nhiệm vụ của người thiết kế là phải biểu diễn được (không gian) lời giải, xây dựng được hàm finess tính độ thích nghi, và cài đặt được các phép toán di truyền dựa trên biểu diễn di truyền và hàm fitness đã xác định. Chẳng hạn, ta sẽ áp dụng thuật giải di truyền để giải *bài toán người bán hàng thông minh – travelling salesman problem* (**TSP**).

TSP. Một người bán hàng cần đi qua n thành phố, mỗi thành phố đi qua đúng một lần, với giả thiết luôn có đường đi trực tiếp giữa 2 thành phố và khoảng cách hay thời gian di chuyển giữa 2 thành phố đó được cho. Hay thiết kế một lộ trình ngắn nhất cho người bán hàng.



Ta thiết kế thuật giải di truyền giải bài toán người du lịch thông minh – **GA_TSP** như sau.

- **Biểu diễn di truyền và độ thích nghi**

Giả sử đồ thị biểu diễn n thành phố là đồ thị G(V, E).

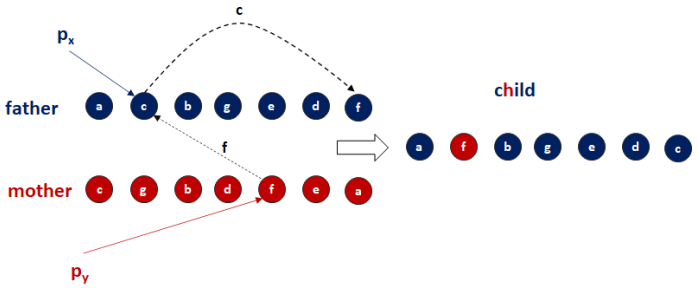
. Gọi P = $x_1x_2..x_n$ là lộ trình đi qua n thành phố thỏa $x_i \in V$, $\forall i = 1, 2,.., n$ và $x_ix_{i+1} \in E$, $x_i \neq x_j$ và $d(x_i, x_{i+1}) > 0$, $\forall i = 1,.., n-1$. Thì P là một lời giải khả thi. Chẳng hạn, trong đồ thị trên, P = abcdefg là một lộ trình.

. Đặt $fitness(P) = fitness(x_1x_2..x_n) = nD - \sum_{i=1}^{n-1} d(x_i, x_{i+1})$, với $D = max\{d(u,v): (u,v) \in E\}$, là độ thích nghi của cá thể P. Ví dụ fitness(abcdefg) = 10×6-12+8+11+11+6+9 = 3.

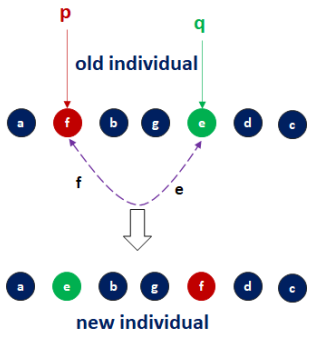
Tiếp theo, ta có thể cài đặt phép lai và phép đột biến, các phép khác có thể sử dụng lại các hàm đã định nghĩa trên. Các cài đặt sau chỉ là một trong nhiều cách, bạn có thể phát triển các cài đặt khác cho 2 phép di truyền chính của GA cho TSP.

- **Phép lai và phép đột biến**

. **crossover**($x_1..x_n$, $y_1..y_n$). Nhận vào lộ trình "mẹ" $x_1..x_n$ và lộ trình "cha" $y_1..y_n$. Trước hết, crossover phát sinh 2 số nguyên $p_x, p_y \in \{1, 2,.., n\}$, và sinh ra (các) lộ trình "con" bằng cách hoán đổi các gien tại các vị trí này.



. **mutation**($u_1..u_n$). Nhận đầu vào là cá thể chịu đột biến $u_1..u_n$, mutation sinh một cặp vị trí ngẫu nhiên $(p, q) \in \{1, 2,.., n\}^2$, và hoán vị 2 gien ở 2 vị trí này.



Về sự hội tụ của thuật giải GA-TSP. Ta thiết kế xích Markov có n! trạng thái $S_i(t)$ lộ trình tốt nhất của quần thể ở thế hệ thứ t và cũng là là hoán vị thứ i, i =1, 2,.., n!. Thì xác suất từ trạng thái i sang trạng thái j là $p_{ij} = 1/n!$ với i, j = 1, 2,.., n!. Mặc dù p_{ij} rất nhỏ nhưng $p_{ij} \neq 0$, nên ma-trận xác suất chuyển $P = [p_{ij}] \in (0, 1]^{n! \times n!}$ là ma-trận dương nên xích được xây dựng như trên là xích chính quy và tiến trình sẽ hội tụ về lộ trình tốt nhất.

5. Hồi quy di truyền

Trong phân tích dữ liệu, ta đã được học cách *khớp dữ liệu – data fitting* bằng kỹ thuật trong thống kê gọi là *hồi quy – regression*. Theo đó, một *hàm hồi quy* f được thiết lập sao cho f "lượn khớp" nhất với các điểm dữ liệu quan sát được.

Hàm f có thể có dạng tuyến tính, gọi là *hồi quy tuyến tính – linear regression*, mà *biến phụ thuộc* y là tổ hợp tuyến tính của các *biến độc lập* x_i, i = 1,.., n, $y = f_\theta(x_1,.., x_n) = \theta_0 + \theta_1 x_1 +..+ \theta_n x_n$; hay dạng *tựa tuyến tính – quasi-linear regression*, mà là tổ hợp tuyến tính của một số hàm cơ sở $f_i(x_1,..,x_n)$, i=1,..,p, xác định trước, $y = f_\theta(x_1,..,x_n) = \theta_1 f_1(x_1,..,x_n) +..+ \theta_p f_p(x_1,..,x_n)$. Và hồi quy – regression là sử dụng kỹ thuật bình phương tối tiểu thiết lập và giải bài toán tối ưu lồi

$\min_\theta E(\theta)$

để cực tiểu hàm lỗi,

$\theta^* = \min_\theta E(\theta)$.

Trong phần này, ta sử dụng kỹ thuật lập trình tiến hóa để tìm hàm hồi quy có thể có dạng không tuyến tính – non-linear, ta sẽ gọi là *hồi quy di truyền – Genetic Regression* (**GR**) để giải bài toán data fitting trên tập dữ liệu cho trước.

Thực ra, đây cũng chỉ là ứng dụng của thuật giải di truyền - GA nhưng chúng tôi trình bày như một mục riêng biệt vì, như sẽ thấy, đây là một kỹ thuật tổng quát của lập trình tiến hóa mà có thể được sử dụng giải nhiều bài toán tối ưu. Ta vẫn theo lược đồ GA cơ bản gồm

- Biểu diễn di truyền, nhưng ở đây, các chất liệu di truyền sẽ gồm những gien tích cực – active gen và gien thụ động – passive gen.
- Các phép lai (crossover) và đột biến (mutation) vẫn theo nguyên tắc "hoán đổi" hay "thay thế", nhưng do có những gien tích cực có thể gây "đột biến" khác hẳn đột biến "thay gien".

5.1. Bài toán xấp xỉ hàm số

Trong phần đầu, ta đã sử dụng mạng nơ-ron $N_{A,B}$ để xấp xỉ một hàm "lượn" qua tập các điểm quan sát được $\Omega = \{(x^{(i)},y^{(i)}): x^{(i)} \in Domain \subset R^n, y^{(i)} \in Range \subset R, i = 1,2,.., N\}$. Ta sẽ giải lại bài toán này bằng kỹ thuật hồi quy di truyền.

Trong data fiting, để "khớp" (với) dữ liệu, ta giả sử hàm f cần tìm có dạng là tổ hợp tuyến tính của một số hàm cơ sở $f_1,.., f_p$, $f(x) = \theta_1 f_1(x) + .. + \theta_p f_p(x)$ và ta dùng phương pháp *bình phương tối tiểu – OLS* (Ordinal Least Square) để tìm véc-tơ tham số $\theta = (\theta_1, .., \theta_p)$. Cũng thế, trong hồi quy di truyền, ta cũng sẽ sử dụng tập hàm cơ sở $\{f_1, .., f_p\}$ cho trước. Khác biệt giữa OLS và GR là GR sẽ tìm dạng hàm f chứ không chỉ tìm các tham số như trong OLS. Bài toán xấp xỉ hàm liên tục được mô tả hình thức như sau:

Cho tập dữ liệu $\Omega = \{(x^{(i)}, y^{(i)}) \in Domain \times Range \subset R^n \times R, i=1,2,.., N\}$, tìm hàm f: Domain $\rightarrow$ Range "khớp" nhất với các dữ liệu trong Ω.

5.2. Biểu diễn di truyền và độ thích nghi

Trước hết, ta xem hàm cần tìm như một *biểu thức* toán học, mà là sự kết hợp giữa các toán tử và toán hạng theo một *văn phạm biểu thức – expression grammar*. Như vậy, chất liệu di truyền – gen, sẽ gồm các toán tử và toán hạng trong tập $\mathcal{T} \cup \mathcal{O}$, với $\mathcal{T}$ là tập các *ký hiệu kết thúc – terminal* và $\mathcal{O}$ là tập các *ký hiệu không kết thúc – non-terminal*, gồm các toán tử hay các hàm cơ sở xác định trước. Để xấp xỉ một hàm thực n biến, tập ký hiệu kết thúc $\mathcal{T}$ sẽ chứa các ký hiệu biến đầu vào, cùng với các ký hiệu số vô tỉ $e \approx 2.71828$ và $\pi \approx 3.1416$. Ngoài 4 phép số học thông thường, tập ký hiệu kết thúc có thể có thêm các hàm cơ sở khác tùy thuộc người thiết kế. Chẳng hạn, dưới đây là một thiết kế thuật giải di truyền cho quy hoạch di truyền.

- **Chất liệu di truyền**

$\mathcal{O}$ = {+, -, *, /} ∪ {exp(.,.), log(.,.), sin(.), cos(.), neg(.)}

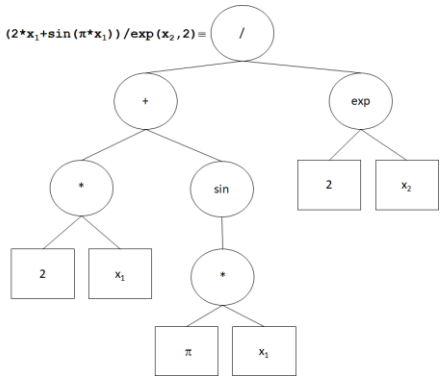
$\mathcal{T}$ = {x_1,.., x_n} ∪ {e, π}

f(x_1,.., x_n) là biểu thức gồm các toán tử thuộc $\mathcal{O}$ và toán hạng thuộc $\mathcal{T}$ hoặc các chuỗi biểu diễn số trong tập số thực R. Chẳng hạn f(x_1,x_2)=(2*x_1+sin(π*x_1))/exp(x_2,2), là một *biểu thức cá thể*, một nhiễm sắc thể.

Lưu ý. Để minh họa trực quan, chúng tôi sẽ sử dụng cấu trúc cây để biểu diễn các biểu thức, gọi là cây biểu thức hay đơn giản là biểu thức. Chẳng hạn cây biểu thức cho

f(x_1, x_2)=(2*x_1+sin(π*x_1))/exp(x_2,2)

như hình vẽ bên.



- **Độ thích nghi**

Gọi **eval**(f) là hàm lượng giá biểu thức f, thì độ thích nghi của biểu thức cá thể f được xác định bởi

$$fitness(f) = \sum_{(x,y)\in\Omega} |eval(f(x)) - y| .$$

Với định nghĩa này, cá thể thích nghi nhất là cá thể "hoàn hảo" nhất trong quần thể hiện tại, nghĩa là ít lỗi nhất. Như vậy, thao tác sắp xếp trong hàm chọn lọc, excommunication(.), sẽ sắp xếp độ thích nghi theo thứ tự tăng dần.

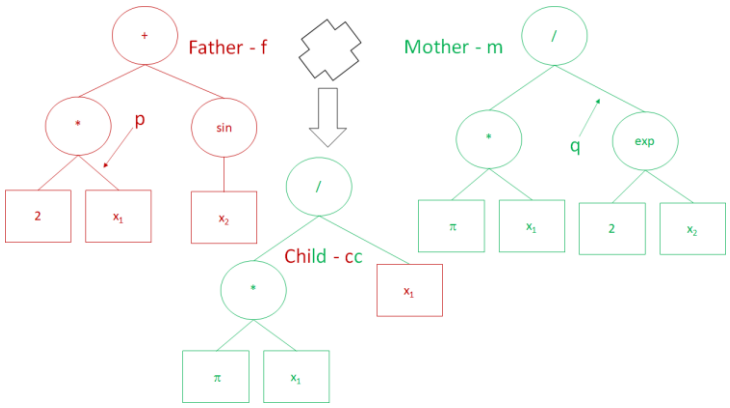
5.3. Phép lai và phép đột biến

Cùng với mô tả các bước thực hiện phép toán di truyền, ta cũng sẽ minh họa các phép toán di truyền này bằng hình vẽ qua các ví dụ cụ thể.

crossover(f,m)

. chọn ngẫu nhiên các điểm lai p, q trên f và m, p ∈ {1,.., len(f)} và q ∈ {1,.., len(m)}.

. hoán chuyển các cây con để sinh ra (các) cây biểu thức c cho thế hệ mới.



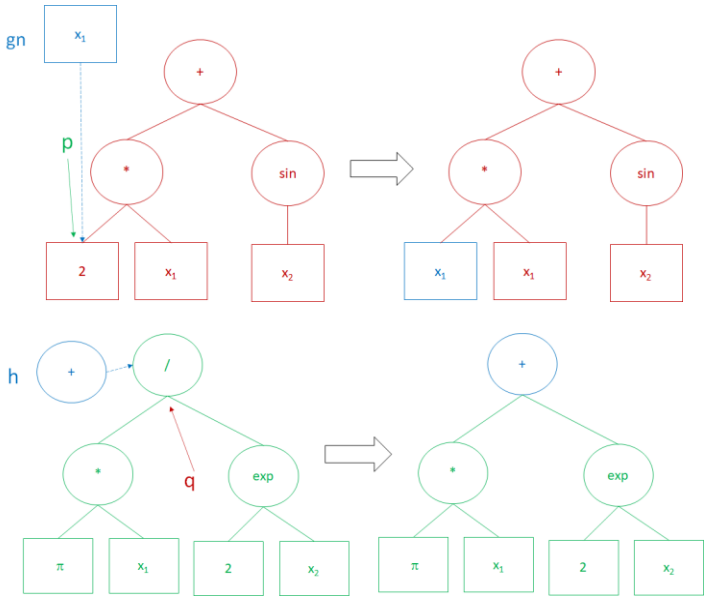
Về phép đột biến, có 2 hình thức đột biến. Hình thức thứ nhất, đột biến thay thế, thực hiện, đột biến bằng thay đổi chất liệu di truyền này bằng chất liệu khác đã có sẵn, ta vẫn ký hiệu hàm **mutation**(.). Hình thức thứ 2 là đột biến làm biến đổi cấu trúc hình dạng cá thể bằng cách rút gọn biểu thức, ta ký hiệu hàm **reduce**(.).

mutation(f)

. chọn ngẫu nhiên vị trí p, p ∈ {1,.., len(f)} mà có thể bị đột biến gen.

. nếu gen tại vị trí p là toán hạng, gen(p) ∈ $\mathcal{T}$, thì chọn ngẫu nhiên một toán hạng khác, gn ∈ $\mathcal{T}$, và thay thế cho gen ở vị trí p, f[p] = gn.

. nếu gen(p) là toán tử, g = gen(p) ∈ $\mathcal{O}$, thì chọn ngẫu nhiên hàm h ∈ $\mathcal{O}$ có cùng số tham số như hàm h và thay thế gen ở vị trí p, f[p] = h.



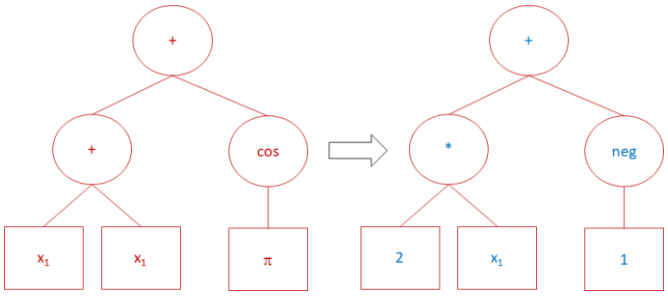
reduce(f) rút gọn các biểu thức con mà không sử dụng giá trị của các biến. Chẳng hạn

. $x_1 + x_1$ rút gọn thành 2*x

. cos(π) chuyển thành -1 = neg(1)

..

Chính hàm này phát sinh những giá trị mới như hình bên, sau khi ruduce, các nhân tố mới xuất hiện là 2 và 1.



5.5. Xấp xỉ hàm thực liên tục

Trước hết, ta hiểu khái niệm trù mật như sau

Tập A trù mật trong B nếu mỗi phần tử x ∈ B và với ε > 0, ∃y ∈ A: d(x,y) < ε, trong đó d(x,y) ký hiệu khoảng cách giữa 2 phần tử x và y.

Như vậy, với các số vô tỷ e và π, và với các phép toán số học thông thường, hồi quy di truyền – GR, có thể sinh ra tập trù mật trong tập số thực R. Hơn nữa, hàm thực f liên tục trên tập D, chính là một biểu thức được tạo từ các số thực, các biến độc lập và các phép toán hay các hàm sơ cấp.

Với ε > 0, do khả năng phát sinh tập trù mật với R của GP, sẽ tồn tại hàm g được tạo từ các thành phần trong $\mathcal{T}$ và $\mathcal{O}$ và các hệ số sinh từ GR bằng các phép số học xấp xỉ hàm f, sao cho |g(x) – f(x)| < ε. Như vậy, có thể thấy:

GF có thể xấp xỉ mọi hàm thực liên tục.

5.6. Một số lưu ý về kỹ thuật hồi quy di truyền

5.6.1. Sử dụng biểu thức dạng hậu tố - **postfix**

Trước hết, nên sử dụng văn phạm *biểu thức Balan ngược* hay *biểu thức hậu tố - postfix*, nghĩa là toán tử được đặt sau toán hạng thay vì đặt giữa các toán hạng như biểu thức *trung tố - infix*, quen thuộc, hay cấu trúc dữ liệu cây ta minh họa trong các ví dụ trên. Chẳng hạn với biểu biểu thức

f_{infix} ≡ 2*x+y*(2-z) sẽ có dạng $f_{postfix}$ ≡ 2.x.*.y.2.z.-.*.+.

Thuật giải **infix2posfix**(.) sau chuyển một biểu thức dạng trung tố đơn giản (chỉ 4 phép toán số học 2 ngôi, không có dấu ngoặc).

Thuật giải infix2postfix sử dụng một ngăn xếp – stack để xử lý thứ tự ưu tiên của phép toán (nhân/chia trước, cộng/trừ sau). Hàm infix2postfix nhận vào một mảng là một biểu thức infix và trả về một mảng là biểu thức postfix tương đương.

Ý tưởng cơ bản của thuật giải là xét từng ký hiệu symbol trong biểu thức infix, symbol = infix[i], i = 0,.., len(infix) – 1. Nếu symbol là toán hạng thì chuyển thẳng symbol ra chuỗi postfix, mà khởi tạo đầu bằng rỗng, postfix = []. Nếu symbol là toán tử, symbol in [‘+’, ‘-‘, ‘*’, ‘/’], thì nếu stack khác rỗng và độ ưu tiên tính toán của toán tử symbol nhỏ hơn độ ưu tiên toán tử trên đỉnh skack thì đổ hết các toán tử đang có trong stack mà độ ưu tiên nhỏ hơn toán tử symbol ra chuỗi postfix; rồi thì đặt toán tử symbol vào đỉnh stack. Ngược lại, stack rỗng hay độ ưu tiên lớn hơn đưu tiên trên đỉnh stack, thì đặt toán tử symbol vào đỉnh stack.

```
def infix2postfix(infix):
    def op(symbol):
        if symbol in ['+', '-', '*', '/']:
            return True
        else:
            return False
    def prior(operator):
        if operator in ['+', '-']:
            return 1
        else:
            return 2
    postfix = []
    stack = []
    n = len(infix)
    for i in range(n):
        symbol = infix[i]
        if not(op(symbol)):
            postfix.append(symbol)
        else:
            if stack == []:
```

```
            stack = [symbol] + stack
        else:
            if prior(symbol) > prior(stack[0]):
                stack = [symbol] + stack
            else:
                while (stack != []) and (prior(stack[0]) >= prior(symbol)):
                    o = stack[0]
                    postfix.append(o)
                    stack.remove(o)
                stack = [symbol] + stack

    while stack != []:
        o = stack[0]
        postfix.append(o)
        stack.remove(o)
    return postfix
#main
inf = [2, '*',3, '+','x','/',5]
post = infix2postfix(inf)
print(post)
```

Bài tập. Khi hiểu rõ cơ chế hoạt động của hàm infix2postfix, bạn có thể mở rộng để nó có thể chuyển biểu thức tổng quát gồm các phép toán hai 2 ngôi, và có dấu ngoặc, chỉ bằng cách mở rộng tập **O** và quy định độ ưu tiên cho các phép toán mới.

Với cách biểu diễn bằng biểu thức postfix, thuật giải lượng giá một biểu thức sẽ dễ dàng hơn như ta sẽ khảo sát sau.

Biểu thức tiền tố - prefix, với toán tử trước toán hạng cũng có thể được xử lý hiệu quả theo cùng cách.

5.6.2. Xử lý ngoại lệ - **exception**

Trong quá trình tiến hóa, GP cần lượng giá các biểu thức cá thể để tính độ thích nghi của nó. Để đảm bảo quá trình không bị hệ điều hành ngắt

(interrupt) vụn do các phép tính có thể dẫn đến trường hợp không xác định, chẳng hạn phép chia cho 0, log của số âm, căn bậc chẵn của số âm, … mà ta gọi chung là những trường hợp ngoại lệ.

Như vậy, khi thực hiện các phép toán có thể gây ra ngoại lệ, cần kiểm tra điều kiện xem các toán hạng có làm cho ngoại lệ xảy ra hay không. Nếu ngoại lệ xảy ra thì ta xem độ thích nghi của cá thể đó là vô cùng lớn mà không cần tính tiếp.

5.6.3. Lượng giá một biểu thức hậu tố - **eval**

Thuật giải **eval**(postfix) sau nhận vào một biểu thức postfix đơn giản với các phép toán số học 2 ngôi quen thuộc, lượng giá và trả về giá trị của postfix. Nếu ngoại lệ xảy ra thì trả về giá trị vô cùng lớn được thiết lập trước. Thuật giả eval cũng sử dụng một ngăn xếp stack để lưu các giá trị trung gian.

Thuật giải eval(postfix) quét mọi ký hiệu symbol của biểu thức postfix, symbol = postfix[i], i = 0,.., len(postfix) – 1. Nếu symbol là toán hạng thì chuyển sang giá trị số và đặt vào stack, khởi tạo stack = []. Nếu ký hiệu là phép toán (2 ngôi) thì lấy các toán hạng tương ứng trong stack ra và thực hiện phép toán symbol trên các toán hạng này, kết quả đặt ngược lại vào stack. Khi chuỗi postfix đã được quét hết, trả kết quả đang ở đỉnh stack về.

```
def eval(postfix):
    infinity = 100000.0
    def op(symbol):
        if symbol in ['+', '-', '*', '/']:
            return True
        else:
            return False
    stack = []
    n = len(postfix)
    for i in range(n):
        symbol = postfix[i]
        if not op(symbol):
            operand = float(symbol)
            stack.append(operand)
        else:
            oper2 = stack[len(stack)-1]
```

```
        oper1 = stack[len(stack)-2]
        stack.remove(oper2)
        stack.remove(oper1)
        if symbol == '+':
            r = oper1 + oper2
        elif symbol == '-':
            r = oper1 - oper2
        elif symbol == '*':
            r = oper1 * oper2
        elif symbol == '/':
            if oper2 == 0:
                return infinity
            else:
                r = oper1 / oper2
        else:
            print("000")
        stack.append(r)
    return stack[len(stack)-1]
#main
post = [2, 3, 4, '/', '+', 5, '+']
print(post)
print(eval(post))
```

Bài tập. Bạn có thể mở rộng thuật giải eval để xử lý biểu thức tổng quát gồm các toán tử với số ngôi khác nhau.

5.6.4. Quá khớp – **over fitting**

Mặc dù GR có thể xấp xỉ mọi hàm liên tục nhưng trong thực tế, khi sử dụng GR để "khớp" dữ liệu, các biểu thức kết quả có khi dài và rất phức tạp để có thể "lượn" qua mọi điểm dữ liệu, gọi là *over fitting*. Vấn đề của các bài toán khớp dữ liệu không phải là tìm hàm quá khớp với dữ liệu đã biết mà ta cần tìm một hàm có khả năng dự báo tốt những điểm dữ liệu không thuộc tập dữ liệu đã biết. Vì thế, bằng cách chấp nhật hàm do GR sinh ra không "khớp" hoàn hảo tập dữ liệu, ta nên khống chế chiều cao

cây biểu thức, hay số toán tử tối đa có thể có trong một biểu thức cá thể.

Tóm tắt

Bằng cách xem nhiệm vụ giải quyết các vấn đề thực tế bằng máy tính là nhiệm vụ tìm kiếm tối ưu, ta có thể đưa về bài toán khớp dữ liệu – data fitting và sử dung các kỹ thuật tìm kiếm heuristics.

. Mạng nơ-ron là phương pháp mô hình hóa bằng cách tìm các ma-trận trọng số theo heuristic "leo đồi" theo hướng dốc nhất. Kỹ thuật giảm đạo hàm – gradient descent (GD hay Δ) được áp dụng để cập nhật các ma-trận trọng số.

. Thuật giải di truyền là heuristic mô phỏng tiến hóa tự nhiên giải bài toán tối ưu mà trên nguyên tắc, có thể tìm được lời giải tối ưu toán cục.

. Kỹ thuật lập trình tiến hóa là áp dụng thuật giải di truyền bằng cách biểu diễn di truyền được cấu trúc lời giải, định nghĩa được môi trường qua hàm thích nghi fitness, và cài đặt được các phép toán di truyền.

. Hồi quy di truyên – Genetic Regression là một kỹ thuật mạnh của lập trình tiến tiến hóa, theo đó cấu trúc di truyền có thể thay đổi cả hình thức lẫn nhội dung (chất liệu) di truyền.