Weekly research report

School of Information and Communications and Technology
Hanoi University of Science and Technology

Name: Nguyễn Hoàng Vũ
Student Number:
Research Centre: Optimization & Genetic Algorithm
Primary Supervisor:

I Bài toán tối ưu là gì?

- Chúng ta đã được tiếp cận với bài toán tối ưu từ rất lâu, qua các dạng bài như: "Tìm giá trị cực tiểu của hàm số $f(x) = x^2 + 1$".
- Tổng quát hơn, chúng ta có thể phát biểu bài toán như sau:

$$\begin{array}{ll} \min f(x) & \text{(P)} \\ \text{với điều kiện } x \in X & \end{array}$$

trong đó:
+ $X \subset \mathbb{R}^n$ được gọi là tập chấp nhận được hoặc tập ràng buộc
+ mỗi $x \in X$ được gọi là 1 phương án chấp nhận được (feasible solution)
+ $f(x)$ xác định trên X được gọi là hàm mục tiêu

1 Các khái niệm cơ bản

Nghiệm tối ưu địa phương

- $x^* \in X$ được gọi là **nghiệm tối ưu địa phương** (local optimization solution) của bài toán (P) nếu $\exists$ lân cận U của x^* sao cho: $f(x^*) \leq f(x) \forall x \in X \cap U$.
- Nếu dấu bằng chỉ xảy ra khi $x = x^*$ thì gọi x^* là **nghiệm tối địa phương chặt** (strictly local optimization solution).

Nghiệm tối ưu toàn cục

- $x^* \in X$ được gọi là **nghiệm tối ưu toàn cục** (global optimization solution) của bài toán (P) nếu: $f(x^*) \leq f(x) \forall x \in X$.
- Nếu dấu bằng chỉ xảy ra khi $x = x^*$ thì gọi x^* là **nghiệm tối ưu toàn cục chặt** (strictly global optimization solution). Dễ thấy được nếu nghiệm tối ưu toàn cục chặt tồn tại thì nó tồn tại duy nhất.

Tập nghiệm tối ưu toàn cục

- Ta kí hiệu $\text{Argmin}(P) = \underset{x \in D}{\text{Argmin}}\, f(x) = \text{Argmin}\{f(x), x \in X\}$ là tập nghiệm tối ưu toàn cục của bài toán (P). Nếu tập nghiệm này chỉ có một phần tử x^*, viết $x^* = \text{argmin}(P) = \underset{x \in D}{\text{argmin}}\, f(x) = \text{argmin}\{f(x), x \in X\}$. Với bài toán tìm max, tương tự ta kí hiệu Argmax và argmax.

2 Phân loại các bài toán tối ưu

- Tùy thuộc vào hàm mục tiêu, hàm ràng buộc ta có thể phân loại bài toán tối ưu thành các dạng như sau:
+ Quy hoạch tuyến tính: Nếu hàm mục tiêu là tuyến tính và không gian lời giải là một tập lồi.
+ Quy hoạch phi tuyến: Nếu hàm mục tiêu hoặc hàm ràng buộc không phải tuyến tính.
+ Quy hoạch nguyên: Nếu không gian lời giải là các giá trị nguyên rời rạc.
+ Quy hoạch đa mục tiêu: Nếu có nhiều hơn một hàm một tiêu cần tối ưu.
+ Quy hoạch ngẫu nhiên: Khi hàm mục tiêu hoặc hàm ràng buộc chứa các hệ số không rõ ràng, mà giá trị được tuân theo một phân phối nào đó.
+ Quy hoạch tham số: Khi hàm mục tiêu hoặc hàm ràng buộc chứa các tham số.

3 Các phương pháp giải bài toán tối ưu

Phương pháp giải chính xác

- Là các phương án đảm bảo tìm ra nghiệm tối ưu toàn cục chính xác.
- Đặc điểm: đưa ra lời giải đúng 100%, tốn tài nguyên tính toán, nhất là các bài toán lớn. Thường chỉ khả thi với các bài toán nhỏ hoặc có cấu trúc đặc biệt.
- Vd: phương pháp đơn hình (simplex), phương pháp nhánh cận (branch and bound), phương pháp quy hoạch động (dynamic programming), ...

Phương pháp giải gần đúng

- Là các phương pháp không đảm bảo tìm đúng nghiêm tối ưu, nhưng có thể tìm ra nghiệm gần tối ưu trong thời gian ngắn hơn và với chi phí tính toán thấp hơn.
- Đặc điểm: có thể rơi vào bẫy cục bộ, không đảm bảo tối ưu toàn cục, tốc độ nhanh hơn, dùng được cho các bài toán lớn/ phức tạp.
- Vd: thuật toán di truyền (genetic algorithm), thuật toán mô phỏng tôi luyện (simulated annealing), tham lam (greedy), ...

4 Một số ví dụ về bài toán tối ưu

4.1 Bài toán cái túi (Knapsack problem)

- Cho một túi có kích thước W và n đồ vật, mỗi vật có khối lượng w_i và giá trị v_i. hãy tính giá trị lớn nhất có thể mang, không thể mang nặng quá W.
- Gọi:

- n: số lượng vật phẩm,
- v_i: giá trị của vật phẩm thứ i,
- w_i: khối lượng của vật phẩm thứ i,
- W: sức chứa tối đa của túi,
- $x_i \in \{0, 1\}$: biến nhị phân, 1 nếu chọn vật i, 0 nếu không chọn.

- Hàm mục tiêu: $\max \sum_{i=1}^{n} v_i x_i$
- Ràng buộc: $\sum_{i=1}^{n} w_i x_i \leq W$

4.2 Bài toán người du lịch (Travelling Salesman Problem)

- Một người ghé thăm mỗi thành phố một lần, rồi quay trở lại thành phố xuất phát. Mỗi cặp thành phố có một chi phí đi lại. Mục tiêu là tìm lộ trình có chi phí di chuyển nhỏ nhất.
- Gọi:

- n: số lượng thành phố
- c_{ij}: chi phí đi từ thành phố i đến thành phố j
- x_{ij}: 1 nếu đi từ i đến j, 0 nếu không

- Hàm mục tiêu: $\max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ - Ràng buộc: mỗi thành phố được đến đúng 1 lần

II Tính toán tiến hóa, thuật toán tiến hóa là gì?

1 Giới thiệu về tính toán tiến hóa

- Là một tập hợp các thuật toán **Metaheuristic** lấy cảm hứng chọn lọc tự nhiên và di truyền học trong sinh học. Có hai toán tử cần lưu ý là toán tử lai ghép (crossover operator) và toán tử đột biến (mutation operator).

2 Giới thiệu về thuật toán tiến hóa - Evolution Algorithm

- Là một phương pháp thuộc thuật toán metaheuristic, tính toán gần đúng giá trị tối ưu.
- Ý tưởng: giống như trong tự nhiên, các cá thể cạnh tranh với nhau để sinh tồn. Qua các thế hệ cha mẹ, tạo ra thế hệ con thông qua các phép toán lai ghép và đột biến để đa dạng hóa quần thể, rồi tiến hành chọn lọc để sinh ra các thế hệ kế tiếp.
- Mã giả:

```
1 begin
2 Khởi tạo quần thể gồm các cá thể
3 Đánh giá các cá thể
4 while chưa đạt điều kiện dừng do
5     Chọn ra các cá thể cha mẹ để sinh thế hệ kế tiếp
6     Tạo ra các cá thể con bằng các toán tử biến đổi
7     Đánh giá các cá thể con
8     Chọn ra các cá thể trong quần thể mới
9 end
```

Trong đó:

- Khởi tạo các quần thể là xác định các cá thể tiềm năng trong không gian lời giải.
- Chọn ra các cá thể cha mẹ là chọn ra hai hay nhiều (tùy thuật toán) cá thể tiềm năng trong quần thể.
- Tạo ra các cá thể con là đi thực hiện các phép lai tạo trên cá thể cha mẹ, và toán tử đột biến trên chính cá thể con tạo ra.
- Đánh giá các cá thể con là đi xác định độ thích nghi

- Quá trình tạo ra các cá thể con sẽ làm cho số lượng quần thể gia tăng, vì vậy cần phải lựa chọn lại, việc lựa chọn dựa trên giá trị thích nghi của cá thể.

- Điều kiện dừng ở đây có thể là một giá trị tối ưu nào đó, hoặc thời gian thực thi tối đa của thuật toán, số lần đánh giá hàm thích nghi, độ đa dạng của quần thể ở một ngưỡng nào đó.

- Để có thể giải quyết được bài toán tiến hóa, ta phải mô hình hóa bài toán. Khi đó, lời giải đã được mã hóa được gọi là kiểu gen, còn lời giải suy luận từ lời giải mã hóa được gọi là kiểu hình.
- Tùy theo cách mô hình hóa bài toán, thuật toán tiến hóa được phân thành nhiều nhánh khác nhau:
  + Thuật toán di truyền (Genetic Algorithm - GA) các cá thể là chuỗi kí tự
  + Chiến lược tiến hóa (Evolution Stratery - ES) các cá thể là vector số thực
  + Lập trình tiến hóa (Evolutionary Programming - EP)
  ....

III Thuật toán di truyền (Genetic Algorithm - GA)

1 Giới thiệu về thuật toán GA

- GA làm việc với một quần thể các lời giải (được gọi là cá thể) và tiến hóa chúng qua nhiều thế hệ bằng các phép tính toán. - Các thao tác chính trong thuật toán GA:

- Chọn lọc: giữ lại các cá thể tốt.

- Lai ghép (crossover): kết hợp hai cá thể cha mẹ tạo ra cá thể con.

- Đột biến (mutation): thường sẽ ảnh hưởng xấu lên cá thể, tuy nhiên cần để có thể tránh đưa bài toán vào bẫy cục bộ.

2 Mã giả của thuật toán

begin
| Khởi tạo quần thể có N kiểu gen
| Tính toán độ thích nghi (fitness) cho từng cá thể
| **While** (chưa đạt điều kiện dừng) **do**:
| | Chọn các cá thể cha mẹ từ quần thể
| | **While** (chưa chọn được N cá thể con) **do**:
| | | Tiến hành lai ghép với xác suất p_c
| | | Tiến hành đột biến với xác suất p_m
| | **end while**
| | chọn lọc sinh tồn
| **end while**
end

3 Một số định nghĩa cần biết

3.1 Các phương pháp chọn lọc cha mẹ

a) Chọn lọc ngẫu nhiên
- Lấy ngẫu nhiên một số lượng cá thể trong tập quần thể để bổ sung vào tập cha mẹ
b) Chọn lọc theo giá trị thích nghi
Bánh xe Roulete

- Từ tập các giá trị thích nghi $(f_1, f_2, ..., f_n)$ chuyển sang tập giá trị xác suất $(p_1, p_2, ..., p_n)$ sau đó tiến hành lựa chọn dựa trên tập xác suất.

Chọn lọc giao đấu

- Chọn ngẫu nhiên k cá thể từ quần thể, dựa vào độ thích nghi, chọn ra cá thể tốt nhất. Lặp lại cho tới khi đủ số lượng cha mẹ.

Chọn lọc thứ hạng

- Sắp xếp lại giá trị thích nghi, với mỗi cá thể có một hạng 1, 2, ..., n chuyển sang chọn trên tập xác suất.

3.2 Độ thích nghi

a) Độ thích nghi tiêu chuẩn

- Với mỗi cá thể i có một giá trị thích nghi f_i. Độ thích nghi tiêu chuẩn là $F_i = \frac{f_i}{\sum f_i}$.

b) Độ thích nghi xếp hạng

- Với mỗi cá thể i sẽ có giá trị thích nghi f_i hạng j. Độ thích nghi xếp hạng: $F_i = p \times (1-p)^{j-1}$, với $p \in [0, 1]$ được gọi là độ hút.

c) Độ thích nghi có ràng buộc

- Áp dụng khi bài toán có ràng buộc, nếu cá thể vi phạm sẽ bị phạt.
- Công thức: $\text{fitness}_i = f(x_i) - penalty(x_i)$

3.3 Các kiểu lai ghép

- Lai ghép là quá trình kết hợp thông tin di truyền của hai cá thể cha và mẹ để tạo ra một hoặc nhiều cá thể con, nhằm duy trì độ đa dạng của quần thể và khai thác các điểm tốt từ cha mẹ.

a) Lai ghép với mã hóa nhị phân

Lai ghép một điểm cắt - one point crossover

- Chọn một điểm để cắt ngang kiểu gen của cha, mẹ ra làm 2. Sau đó lấy 1 phần của cha ghép với 1 phần của mẹ.
- Ví dụ: đầu vào là

b	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
p1	1	0	0	0	1	0	1	0
p2	0	1	1	1	0	1	0	0

đầu ra

b	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
p1	1	0	0	0	0	1	0	0
p2	0	1	1	1	1	0	1	0

Lai ghép nhiều điểm cắt - multi point crossover

- Có nhiều điểm cắt, các cá thể cha mẹ sẽ hoán đổi gen một cách xen kẽ.

Lai ghép đồng nhất - uniform crossover

- Hiểu đơn giản là từng phần tử trong kiểu gen sẽ có xác suất p trao đổi cho nhau.

b) Lai ghép với mã hóa hoán vị

Lai ghép theo thứ tự - order crossover

ví dụ: có đầu vào

idx	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
p1	2	1	5	3	7	6	4	8
p2	4	2	3	7	1	5	8	6

B0: khởi tạo c1, c2:

idx	1	2	3	4	5	6	7	8
c1								
c2								

B1: chọn ngẫu nhiên một đoạn idx, sau đó sao chép giá trị của p1 vào c1, của p2 vào c2

idx	1	2	3	4	5	6	7	8
p1	2	1	5	3	7	6	4	8
c1			5	3	7	6		
p2	4	2	3	7	1	5	8	6
c2			3	7	1	5		

B2: Xác định vị trí cắt thứ 2 của p1, p2 (trong ví dụ này là 7), sau đó đổi đoạn dưới lên đầu

idx	1	2	3	4	5	6	7	8
p1	4	8	2	1	5	3	7	6
p2	8	6	4	2	3	7	1	5

B3: Loại bỏ các phần tử trong p2 đã xuất hiện ở c1, và trong p1 đã xuất hiện ở c2

idx	1	2	3	4	5	6	7	8
p1	4	8	2					6
p2	8		4	2			1	

B4: Ghép chuỗi còn lại của p2 và p1 vào vị trí 7 của lần lượt c1 và c2

idx	1	2	3	4	5	6	7	8
c1	2	6	5	3	7	6	4	8
c2	2	1	3	7	1	5	8	4

Lai ghép ánh xạ từng phần - partially mapped crossover
ví dụ: có đầu vào

idx	1	2	3	4	5	6	7	8
p1	2	1	5	3	7	6	4	8
p2	4	2	3	7	1	5	8	6

B0: khởi tạo c1, c2:

idx	1	2	3	4	5	6	7	8
c1								
c2								

B1: chọn ngẫu nhiên một đoạn idx, sau đó sao chép giá trị của p1 vào c1, của p2 vào c2

idx	1	2	3	4	5	6	7	8
p1	2	1	5	3	7	6	4	8
c1			5	3	7	6		
p2	4	2	3	7	1	5	8	6
c2			3	7	1	5		

B2: Điền các gen còn thiếu từ trái qua phải, lấy của p2 điền cho c1 và p1 điền cho c2. Nếu như gen này đã tồn tại, thực hiện phép ánh xạ.

ánh xạ: $5 \iff 3;3 \iff 7;7 \iff 1;6 \iff 5$

idx	1	2	3	4	5	6	7	8
p1	2	1	5	3	7	6	4	8
c1	4	2	5	3	7	6	8	**1**
p2	4	2	3	7	1	5	8	6
c2	2	**6**	3	7	1	5	4	8

Ở tại phần tử số 8 của c1 vì 6 đã có nên lúc đấy sẽ tiến hành ánh xạ.
6 ánh xạ tới 5, tuy nhiên 5 đã có trong c1 nên tiếp tục ánh xạ.
5 ánh xạ tới 3, tuy nhiên 3 cũng đã có trong c1 nên tiếp tục ánh xạ tiếp.
3 ánh xạ tới 7, tuy nhiên cũng đã có 7 trong c1 nên tiếp tục ánh xạ.
7 ánh xạ tới 1, lần này c1 chưa có nên ta thấy $c1[8] = 1$.
Tương tự đối với trường hợp của c2.

c) Lai ghép với kiểu mã hóa số thực

Lai ghép số thực đơn giản - simple real number crossover

ví dụ: có đầu vào

idx	1	2	3	4	5	6	7	8
p1	0.1	0.4	0.1	0.8	0.2	0.3	0.7	0.5
p2	0.2	0.3	0.4	0.6	0.5	0.7	0.8	0.2

B0: Chọn ngẫu nhiên một vị trí lai ghép (ở đây ta lấy k = 5), chọn ngẫu nhiên $\alpha \in [0;1]$ (ở đây tôi chọn 0.7) và khởi tạo c1, c2.
B1: Lấy các phần tử từ $1 \to k$ sao chép p1 vào c1, p2 vào c2.

idx	1	2	3	4	5	6	7	8
c1	0.1	0.4	0.1	0.8	0.2			
c2	0.2	0.3	0.4	0.6	0.5			

B2: Các phần tử còn lại được tính từ công thức:

$$c1 = \alpha p1[i] + (1-\alpha)p2[i]; c2 = \alpha p2[i] + (1-\alpha)p1[i].$$

idx	1	2	3	4	5	6	7	8
c1	0.1	0.4	0.1	0.8	0.2	0.42	0.73	0.41
c2	0.2	0.3	0.4	0.6	0.5	0.58	0.85	0.29

Lai ghép mô phỏng nhị phân - simulated binary crossover

B0: lấy ngẫu nhiên $r \in [0;1]$. Tính β

$$\beta = (2r)^{\frac{1}{\eta_c + 1}}, r \leq 0.5$$

$$\beta = (\frac{1}{2(1-r)})^{\frac{1}{\eta_c + 1}}, r > 0.5$$

với η_c là **chỉ số phân tán** của lai ghép. Giá trị càng lớn thì càng gần cha mẹ, ít khác biệt. Giá trị càng nhỏ thì càng ở xa hơn, tăng mức khám phá.
B1: Khi đó các giá trị con được tính bằng công thức:

$$c1 = 0.5 \times ((1+\beta) \times p1 + (1-\beta) \times p2)$$
$$c2 = 0.5 \times ((1+\beta) \times p2 + (1-\beta) \times p1)$$

3.4 Các kiểu đột biến

- Đột biến tuy sẽ ảnh hưởng có thể xấu đến cá thể con. Nhưng đây chính là cách giúp cho ta có thể tăng sự đa dạng của quần thể và tránh rơi vào cái bẫy cục bộ.

a) Đột biến đảo bit - bit flip mutation
- Với mỗi phần tử trong gen, lấy một giá trị $r_i \in [0;1]$. Ta có một giá trị xác suất đột biến p_m. Nếu $r_i \leq p_m$ thì đảo bit đó.
ví dụ: $p_m = 0.4$

r	**0.4**	**0.2**	**0.1**	**0.9**	**0.6**	**0.5**	**0.3**	**0.2**
b	1	0	0	0	1	0	1	0

kết quả:

b	0	1	1	0	1	0	0	1

b) Đột biến trao đổi chéo - swap mutation
- Chọn 2 vị trí khác nhau bất kì. Đổi giá trị phần tử ở 2 vị trí đấy.
c) Đột biến đảo đoạn - inversion mutation
- Chọn ra một đoạn $b_i \to b_j$ sau đó đảo ngược lại đoạn đó.
d) Đột biến đa thức - polynomial mutation
- Thường áp dụng cho mã hóa số thực
Đầu vào: x
B1: Lấy ngẫu nhiên $\mu \in [0, 1]$
B2: Tại mỗi vị trí i tính các giá trị δ^L và δ^R

$$\delta^L = (2\mu)^{\frac{1}{1+\eta_m}} - 1 \quad \text{với } \mu \leq 0.5$$

$$\delta^R = 1 - (2 - 2\mu)^{\frac{1}{1+\eta_m}} \quad \text{với } \mu > 0.5$$

B3: Cập nhật giá trị theo công thức

$$c_i = x_i + \delta^L \times x_i \quad \text{với } \mu \leq 0.5$$

$$c_i = x_i + \delta^R \times (1 - x_i) \quad \text{với } \mu > 0.5$$

3.5 Chọn lọc sinh tồn

a) Chọn lọc xén - truncation selection
- Trong bước chọn lọc, các cá thể sẽ được sắp xếp giảm dần theo độ thích nghi. Mọt ngưỡng Trunc đưa ra để quyết định bao nhiêu phần tử được chọn.
- Ví dụ: Trunc = 0.5 => chọn một nửa số cá thể; có 3 cá thể, Trunc = 0.35, có 3 x 0.35 = 1.05 => chọn 2 cá thể; ....
b) Chọn lọc theo vòng quay Roulete - roulete wheel selection
- Đây là kiểu chọn lọc theo xác suất theo giá trị thích nghi, giá trị thích nghi càng cao thì xác suất lựa chọn càng cao.
- Đầu vào: quần thể N cá thể cho thế hệ tiếp theo.
B1: Với cá thể thứ i có giá trị thích nghi f_i và độ thích nghi tiêu chuẩn F_i với
$F_i \in [0;1], \sum F_i = 1$.
B2: Xếp các cá thể lên một đoạn dài

F	F_1	F_2	...	F_i	...	F_n
S	S_1	S_2	...	S_i	...	S_n

với S bên dưới là cộng dồn độ thích nghi tiêu chuẩn
B3: Chọn 1 số ngẫu nhiên $r \in [0;1]$. Tìm vị trí i thỏa mãn: $S_{i-1} \leq r \leq S_i$. Cá thể i là cá thể được chọn. Lặp lại đến khi nào đạt đủ cá thể.
c) Chọn lọc cục bộ
- Chọn ra một cá thể trong quần thể, sau đó chọn các cá thể nằm lân cận. Với mỗi bài toán, cần định nghĩa thế nào là lân cận?

d) Chọn lọc giao đấu
- Chọn ra k cá thể ngẫu nhiên trong tập quần thể, cá thể tốt nhất được lựa chọn, lặp lại cho tới khi đủ số lượng.

3.6 Điều kiện dừng

Điều kiện	**Mô tả**
Số thế hệ tối đa	Dừng sau khi chạy đến số thế hệ $G_{\max}$
Số lần đánh giá hàm thích nghi tối đa	Dừng sau khi số lần gọi hàm fitness vượt quá giới hạn
Fitness không cải thiện	Dừng nếu fitness tốt nhất không thay đổi trong k thế hệ liên tiếp
Đạt được ngưỡng fitness mong muốn	Nếu một cá thể đạt hoặc vượt ngưỡng fitness định trước
Mức độ đa dạng quần thể thấp	Dừng nếu cá thể trong quần thể trở nên đồng nhất (gần như giống nhau)
Hết thời gian cho phép	Dừng khi vượt quá thời gian tính toán cho phép

4 Một số câu hỏi

4.1 Phương pháp này thoát ra khỏi bẫy cục bộ bằng cách nào?

- Để có thể thoát ra khỏi bẫy cục bộ, ta cần đa dạng hóa quần thể. Điều này giúp thuật toán khám phá nhiều vùng khác nhau trong không gian tìm kiếm, tránh bị mắc kẹt tại một vùng cục bộ.
- Để đa dạng hóa quần thể ta có thể sử dụng lai ghép. Giúp kết hợp các điểm tốt từ các cùng khác nhau trong không gian nghiệm.
- Ngoài ra, phải kể đến cơ chế đột biến. Đột biến có thể đưa cá thể đến vùng chưa được khám phá, là chìa khóa để thoát khỏi cục bộ.
- Vì việc đột biến có thể khiến cá thể xấu đi, do đó thuật toán không chọn toàn bộ các cá thể tốt mà sẽ dùng chọn lọc theo xác suất. Điều này giúp giữ lại một số cá thể kém, duy trì đa dạng $\rightarrow$ tránh hội tụ sớm vào nghiệm địa phương.

4.2 Phương pháp này có gì khác so với các phương pháp bạn đã biết?

Đối với các phương pháp giải chính xác
- Về tính tối ưu, GA sẽ không đảm bảo đưa ra chính xác nghiệm tối ưu toàn cục mà chỉ là gần đúng. Phương pháp giải chính xác thì đưa ra kết quả chính xác.
- Về tốc độ thì GA sẽ cho kết quả nhanh hơn trong những bài toán phức tạp lớn.
- Về khả năng mở rộng GA cũng cho khả năng mở rộng tốt hơn.

5 Cài đặt tổng quát thuật toán

```
import numpy as np
import random
import copy
import matplotlib.pyplot as plt

################################
```

```
class Problem:
    def __init__(self):
        pass

################################

def decode(chromosome, problem: Problem): #decoding for gene encoding
    pass
def get_fitness(x):
    pass

################################

class Individual:
    def __init__(self):
        self.chromosome = None
        self.fitness = None

    def genIndi(self, problem: Problem):
        self.chromosome

    def cal_fitness(self, problem):
        self.fitness = get_fitness(self.chromosome)

    def clone(self):
        return copy.deepcopy(self)

    def __repr__(self):
        return f"chromosome={self.chromosome}, fitness={self.fitness}"

################################

def crossover(parent1, parent2, problem: Problem, eta = 2.0):
    off1 = Individual()
    off2 = Individual()
    r = np.random.rand()
    if (r <= 0.5):
        beta = (2 * r) ** (1.0/(eta + 1))
    else:
        beta = (1.0/(2*(1 - r))) ** (1.0/(eta + 1))

    p1 = parent1.chromosome
    p2 = parent2.chromosome

    c1 = 0.5 * ((1 + beta) * p1 + (1 - beta) * p2)
    c2 = 0.5 * ((1 + beta) * p2 + (1 - beta) * p1)

    c1 = np.clip(c1, 0.0, 1.0)
    c2 = np.clip(c2, 0.0, 1.0)

    off1.chromosome = c1
    off2.chromosome = c2
```

```
    return off1.clone(), off2.clone()


################################

def mutation(ind, eta = 2.0):
    chr = ind.chromosome
    for i in range(chr.size):
        mu = np.random.rand()
        if (mu <= 0.5):
            delta = (2 * mu) ** (1.0/(1 + eta)) - 1
            chr[i] = chr[i] + delta * chr[i]
        else:
            delta = 1 - (2 - 2 * mu) ** (1.0/(1 + eta))
            chr[i] = chr[i] + delta * (1 - chr[i])

    chr = np.clip(chr, 0.0, 1.0)
    ind.chromosome = chr
    return ind.clone()

################################

class Population:
    def __init__(self, pop_size, problem: Problem):
        self.pop_size = pop_size
        self.list_ind = []
        self.problem = problem

    def genPop(self):
        for i in range(self.pop_size):
            ind = Individual()
            ind.genIndi(self.problem)
            ind.cal_fitness(self.problem)
            self.list_ind.append(ind)

    def __repr__(self):
        pass

################################

def selection(list, k = 2):
    tour1 = random.sample(list, k)
    tour2 = random.sample(list, k)
    x = max(tour1, key=lambda ind: ind.fitness)
    y = max(tour2, key=lambda ind: ind.fitness)
    return x.clone(), y.clone()

################################

def survival_selection(list, pop_size):
    list = sorted(list, key=lambda ind: ind.fitness, reverse=True)
    list = list[0: pop_size]
    return list
```

```
################################

def GA(problem, pop_size, max_gen, pc, pm):
    pop = Population(pop_size, problem)
    pop.genPop()
    history = []
    for i in range(max_gen):
        child = []
        while(len(child) < pop_size):
            p1, p2 = selection(pop.list_ind)
            if (np.random.rand() <= pc):
                c1, c2 = crossover(p1, p2, problem)
                c1.cal_fitness(problem)
                c2.cal_fitness(problem)
                child.append(c1)
                child.append(c2)
            if (np.random.rand() <= pm):
                p1 = mutation(p1)
                p2 = mutation(p2)
                p1.cal_fitness(problem)
                p2.cal_fitness(problem)
                child.append(p1)
                child.append(p2)
        pop.list_ind = survival_selection(pop.list_ind + child, pop_size)
        history.append(pop.list_ind[0].fitness)
    solution = pop.list_ind[0]
    return history, solution

################################

# setup
problem = Problem()

pop_size = 200
max_gen = 2000
Pc = 0.9
Pm = 0.2

# start
fitness_history, solution = GA(problem, pop_size, max_gen, Pc, Pm)

#show
for i in range(len(fitness_history)):
    print(f"Generation {i}, bestfitness = {fitness_history[i]:.2f}")

#show
np.set_printoptions(precision=2, suppress=True)
print("solution:")
print(decode(solution.chromosome, problem))
print(f"{solution.fitness:.2f}")

generations = list(range(len(fitness_history)))
```

```
plt.figure(figsize=(10, 5))
plt.plot(generations, fitness_history, marker='o', linestyle='-', color='b', label='
    Best Fitness')

plt.xlabel("Generation")
plt.ylabel("Best Fitness")
plt.title("Fitness Progress Over Generations")
plt.legend()
plt.grid(True)
plt.show()
```