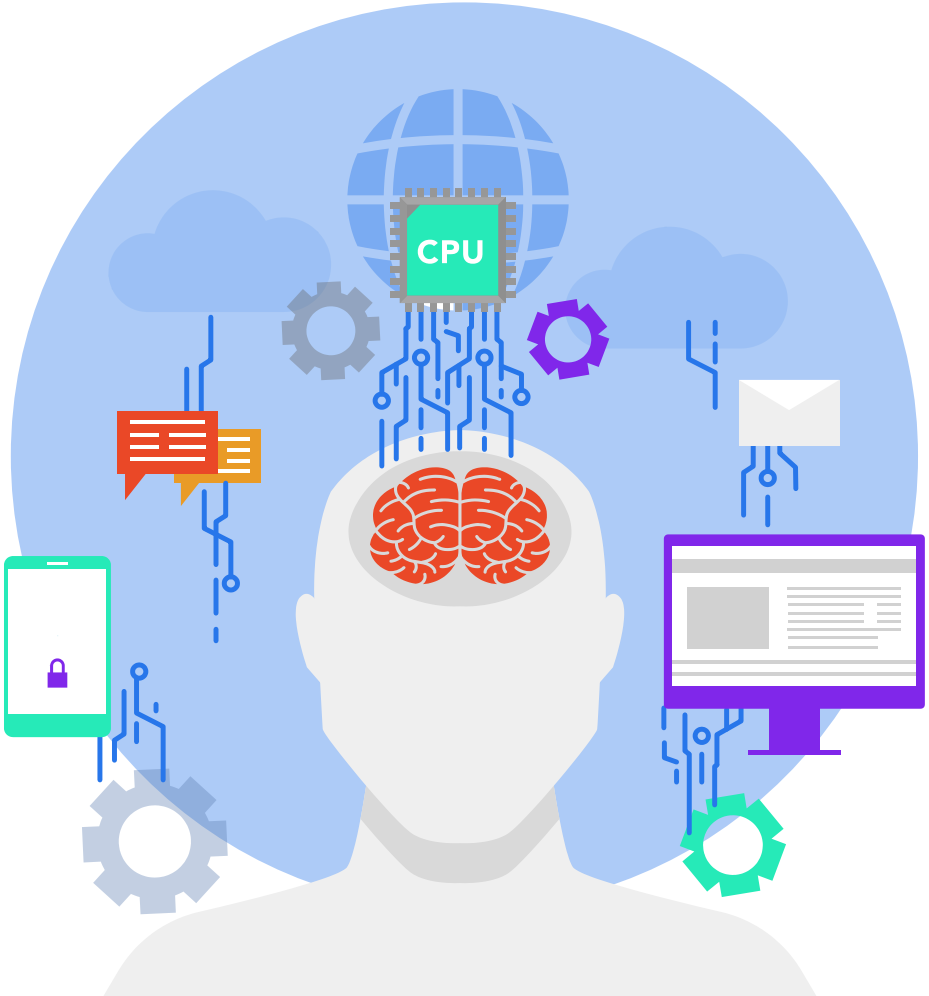
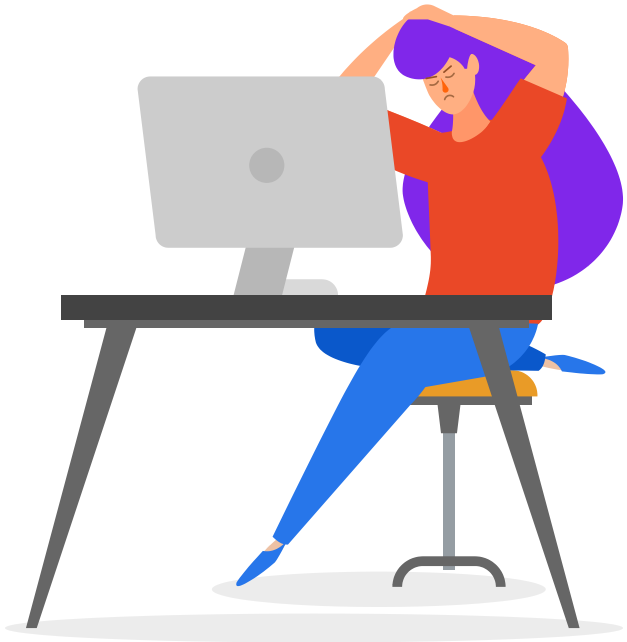


Chương 6: Lập trình tiến hoá và giải thuật di truyền

6.1 Lập trình tiến hoá

Giới thiệu

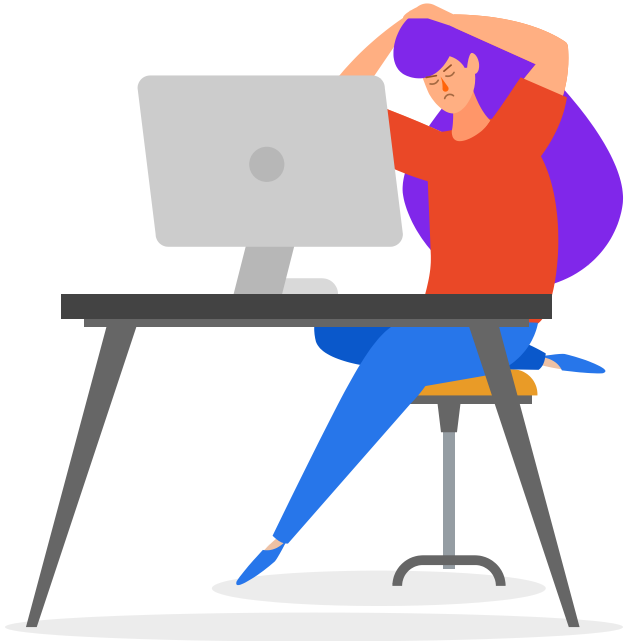
Lập trình tiến hóa (Evolutionary Programming - EP) là một phương pháp tính toán dựa trên các nguyên tắc của tiến hóa sinh học, được sử dụng để giải quyết các bài toán tối ưu hóa phức tạp. EP mô phỏng các cơ chế như chọn lọc tự nhiên, đột biến và sinh sản để tìm kiếm lời giải tối ưu hoặc gần tối ưu trong không gian tìm kiếm lớn.



6.1 Lập trình tiến hoá

Khái niệm cơ bản

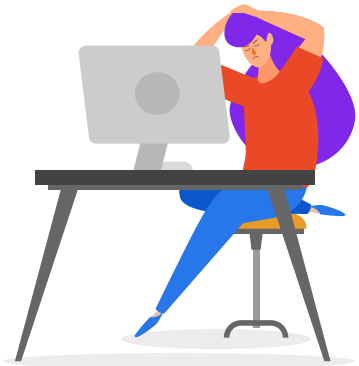
- EP duy trì một quần thể các cá thể, mỗi cá thể đại diện cho một lời giải tiềm năng của bài toán.
- Các cá thể tiến hóa qua nhiều thế hệ thông qua các toán tử như đột biến và chọn lọc.
- Không giống giải thuật di truyền (GA), EP không sử dụng toán tử lai ghép, mà tập trung vào đột biến để tạo ra sự đa dạng.



6.1 Lập trình tiến hoá

Quy trình chi tiết

1. **Khởi tạo**: Tạo ngẫu nhiên một quần thể ban đầu gồm các cá thể (thường được mã hóa dưới dạng số thực hoặc vectơ).
2. **Đánh giá độ thích nghi**: Tính toán giá trị hàm thích nghi (fitness function) để đánh giá chất lượng của mỗi cá thể.
3. **Đột biến**: Áp dụng các phép đột biến ngẫu nhiên để tạo ra các cá thể mới, ví dụ: thêm nhiễu Gaussian vào các tham số của cá thể.
4. **Chọn lọc**: Lựa chọn các cá thể có độ thích nghi cao để tạo thành thế hệ tiếp theo (thường sử dụng chọn lọc giải đấu - tournament selection).
5. **Lặp lại**: Tiếp tục quá trình cho đến khi đạt tiêu chí dừng, chẳng hạn như số thế hệ tối đa hoặc lời giải đạt ngưỡng mong muốn.



6.1 Lập trình tiến hoá

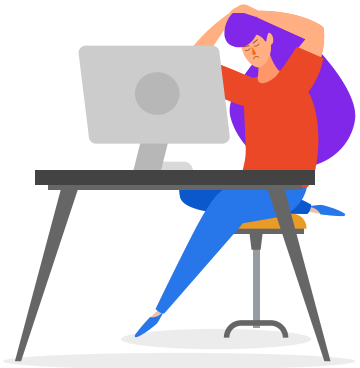
Ví dụ

Giả sử bài toán tối ưu hóa hàm số $f(x) = -x^2 + 2x + 50$ trong khoảng $x \in [0, 10]$.
Mục tiêu là tìm giá trị x sao cho f(x) đạt cực đại.

- Khởi tạo quần thể: 10 cá thể với giá trị x_i ngẫu nhiên trong [0, 10].
- Hàm thích nghi: $f(x) = -x^2 + 2x + 50$.
- Đột biến: Thêm nhiễu ngẫu nhiên $\delta \sim \mathcal{N}(0.01)$ vào (x):

$$x_{new} = x + \delta, \delta \sim \mathcal{N}(0.01)$$

- Chọn lọc: Chọn 5 cá thể gốc + 5 đột biến tốt nhất dựa trên giá trị f(x) → 10 quần thể mới
- Sau nhiều thế hệ, quần thể sẽ hội tụ quanh giá trị $x \approx 5$, nơi f(x) đạt cực đại.



6.1 Lập trình tiến hoá

```python
import numpy as np

def fitness(x):
    return -x**2 + 10*x + 50
# Khởi tạo quần thể
population = np.random.uniform(0, 10, size=10)

for generation in range(50):
    scores = fitness(population)

    # Chọn lọc 5 cá thể tốt nhất
    top_idx = np.argsort(scores)[-5:]
    top_individuals = population[top_idx]

    # Sinh thế hệ mới qua đột biến
    new_individuals = top_individuals +
np.random.normal(0, 0.1, size=5)
    new_individuals = np.clip(new_individuals, 0, 10)

    # Tạo quần thể mới
    population = np.concatenate([top_individuals,
new_individuals])
```

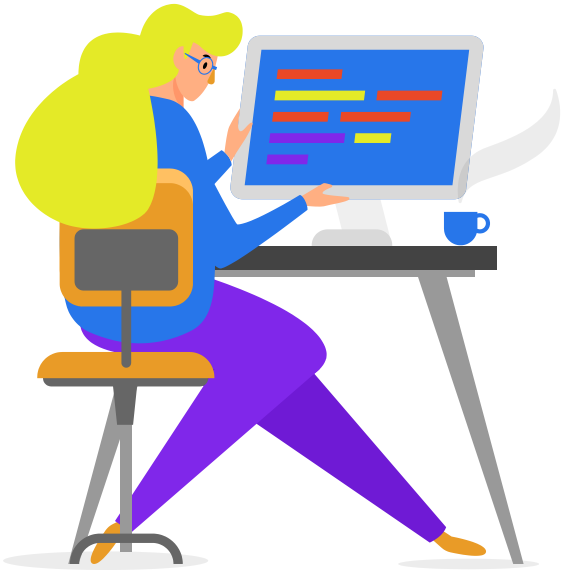
```python
# Kết quả
best_x =
population[np.argmax(fitness(population))]
print("Giá trị x tốt nhất:", best_x)
print("f(x) =", fitness(best_x))
```

6.2 Lịch sử phát triển

Lập trình tiến hóa và các giải thuật liên quan đã trải qua nhiều giai đoạn phát triển, từ ý tưởng ban đầu đến các ứng dụng hiện đại:

- **Thập niên 1960**: Lawrence J. Fogel giới thiệu lập trình tiến hóa tại Mỹ, tập trung vào việc mô phỏng tiến hóa để giải quyết các bài toán trí tuệ nhân tạo, đặc biệt là trong nhận dạng mẫu và dự đoán.
- **Thập niên 1970**: John Holland phát triển giải thuật di truyền (GA), đặt nền móng cho các phương pháp tính toán tiến hóa. Ông nhấn mạnh vào toán tử lai ghép để tạo ra các cá thể mới.
- **Thập niên 1980**: Các biến thể của EP được phát triển, với sự tập trung vào tối ưu hóa các bài toán liên tục. Các phương pháp như chiến lược tiến hóa (Evolution Strategies - ES) cũng ra đời.
- **Thập niên 1990**: James Kennedy và Russell Eberhart giới thiệu giải thuật tối ưu hóa bày đàn (PSO) vào năm 1995, lấy cảm hứng từ hành vi tập thể của các loài động vật.
- **Thế kỷ 21**: Các giải thuật tiến hóa được tích hợp vào học máy, tối ưu hóa công nghiệp, tài chính, y học và các lĩnh vực khác. Các thư viện như DEAP (Python) và ECJ (Java) hỗ trợ triển khai các giải thuật này.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá



01 Quần thể (Population)

02 Hàm thích nghi (Fitness Function)

03 Đột biến (Mutation)

03 Chọn lọc (Selection)

04 Tiêu chí dừng

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Quần thể

Định nghĩa:

Trong các thuật toán tiến hoá như Giải thuật di truyền (GA) hay PSO, quần thể là một tập hợp gồm nhiều cá thể (individuals), trong đó mỗi cá thể đại diện cho một lời giải tiềm năng của bài toán đang xét.

- Mỗi cá thể được **mã hóa** dưới một dạng cụ thể (gọi là **genotype**), như:
  - **Chuỗi nhị phân**: ví dụ 101011 đại diện cho một tham số nhị phân.
  - **Vector số thực**: ví dụ $[3.2, 1.5, 0.7]$ là lời giải trong không gian liên tục.
  - **Chuỗi ký tự**: dùng cho bài toán lập lịch, xếp lớp, v.v.

Vai trò của quần thể:

- Tạo sự **đa dạng trong tìm kiếm** → giúp tránh kẹt ở cực trị cục bộ.
- Mỗi thế hệ (generation) là một quần thể mới được sinh ra từ thế hệ trước qua chọn lọc, lai ghép, đột biến.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Quần thể

Bài toán tối ưu hàm một biến:

Tối ưu hàm:

$$f(x) = -x^2 + 2x + 50 \text{ với } x \in [0,10]$$

- Mã hoá cá thể bằng **số thực**: mỗi cá thể là một giá trị x trong đoạn [0, 10].

```
# Quần thể gồm 5 cá thể (số thực)
population = [2.5, 7.8, 4.1, 6.0, 1.2]
```

→ Mỗi giá trị trong quần thể là **một lời giải tiềm năng**.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Quần thể

Ví dụ minh hoạ:

Bài toán tối ưu tổ hợp:

Bài toán: tìm chuỗi nhị phân 6 bit có nhiều số 1 nhất.

- Mã hoá cá thể: chuỗi nhị phân dài 6 ký tự.

```
population = ['101011', '111000', '000000', '110110', '011001']
```

→ Hàm fitness sẽ đếm số lượng số 1 trong chuỗi.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Thích nghi

Định nghĩa:

Hàm thích nghi (Fitness Function) là hàm dùng để đánh giá chất lượng của mỗi cá thể trong quần thể, tức là xác định "mức độ tốt" của lời giải mà cá thể đó đại diện.

- Giá trị hàm thích nghi càng cao (đối với bài toán tối đa hóa) → cá thể càng có khả năng được chọn để sinh ra thế hệ tiếp theo.
- Hàm này hướng dẫn quá trình tiến hóa, giúp quần thể dần hội tụ đến lời giải tối ưu.

Đặc điểm:

- Được thiết kế phù hợp với mục tiêu tối ưu của bài toán.
- Có thể là: Hàm mục tiêu gốc (trong bài toán tối ưu hóa) hoặc Hàm đánh giá riêng (trong các bài toán tổ hợp, phân loại, lập lịch...).
- Có thể cần chuẩn hóa hoặc chuyển đổi nếu bài toán là tối thiểu hóa (ví dụ: dùng $\frac{1}{1+f(x)}$ để biến thành tối đa hóa).

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Thích nghi

Ví dụ 1: Bài toán tối ưu hóa hàm số
Tối ưu hàm:

$$f(x) = -x^2 + 10x + 50, x \in [0,10]$$

Ta chọn **hàm thích nghi chính là hàm mục tiêu**:

$$fitness(x) = f(x) = -x^2 + 10x + 50$$

```
def fitness(x):
    return -x**2 + 10*x + 50

# Đánh giá một quần thể gồm 3 cá thể:
population = [2.0, 5.0, 9.0]
fitness_values = [fitness(x) for x in population]
# Kết quả: [66.0, 75.0, 59.0]
```

→ Cá thể có giá trị x=5.0 được xem là "thích nghi nhất".

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Thích nghi

Ví dụ 2: Bài toán đếm số 1 trong chuỗi nhị phân

Tối đa hóa số lượng bit 1 trong chuỗi dài 6 ký tự.

•Cá thể: '101011'

•Hàm thích nghi:

$$fitness(bitstring) = số\ lượng\ số\ 1\ trong\ chuỗi$$

```
def fitness(bitstring):
    return bitstring.count('1')

# Đánh giá quần thể:
population = ['101011', '111000', '000000']
fitness_values = [fitness(x) for x in population]
# Kết quả: [4, 3, 0]
```

→ Cá thể '101011' có fitness cao nhất → được ưu tiên chọn.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Đột biến

Định nghĩa:

Đột biến là **quá trình thay đổi ngẫu nhiên một phần (hoặc toàn bộ) gen của cá thể**, nhằm **duy trì sự đa dạng di truyền** trong quần thể và giúp thuật toán thoát khỏi các điểm cực trị cục bộ.

- Đột biến thường được thực hiện sau khi chọn lọc và lai ghép.
- Mức độ đột biến thường **nhỏ** để tránh phá hỏng cấu trúc tốt đã có của lời giải.

Vai trò của đột biến:

•Giúp quần thể **khám phá không gian lời giải** rộng hơn.

•Tránh hiện tượng **hội tụ sớm** (premature convergence).

•Tạo cơ hội để tìm ra **các lời giải tốt hơn** trong các vùng chưa được khám phá.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Đột biến

Ví dụ 1: Đột biến trên cá thể số thực

Giả sử cá thể được mã hóa bằng số thực:

$$x = 4.7$$

Thêm nhiễu Gaussian:

$$\delta \sim \mathcal{N}(0.01) \Rightarrow x' = 4.7 + \delta$$

```
import numpy as np

x = 4.7
delta = np.random.normal(0, 0.1)
x_mutated = np.clip(x + delta, 0, 10)  # đảm bảo vẫn
trong [0,10]
```

→ Cá thể mới sẽ hơi khác xxx, giúp khám phá vùng lân cận.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Đột biến

Ví dụ 2: Đột biến chuỗi nhị phân

Cá thể: '101011'

→ Chọn ngẫu nhiên một vị trí và lật bit:

```
import random

def mutate(bitstring):
    index = random.randint(0, len(bitstring) - 1)
    mutated_bit = '0' if bitstring[index] == '1' else '1'
    return bitstring[:index] + mutated_bit +
bitstring[index+1:]

mutate('101011')  # có thể thành '100011', '101111', v.v.
```

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Chọn lọc

Định nghĩa:

Chọn lọc là bước trong giải thuật di truyền dùng để **lựa chọn các cá thể "tốt" hơn** (có fitness cao hơn) từ quần thể hiện tại để **tạo ra thế hệ kế tiếp** thông qua **lai ghép (crossover)** và **đột biến (mutation)**.

- **Mục tiêu**: Duy trì các cá thể tốt và loại bỏ dần các cá thể kém để tăng chất lượng quần thể theo thời gian.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Chọn lọc

Phương pháp	Mô tả ngắn gọn	Ưu điểm	Nhược điểm
Roulette Wheel (Tỷ lệ xác suất)	Xác suất chọn tỷ lệ với fitness. Giống như quay vòng quay may mắn.	Đơn giản, dễ hiểu	Có thể thiên lệch nếu một cá thể có fitness quá cao
Tournament Selection (Giải đấu)	Chọn ngẫu nhiên một nhóm nhỏ cá thể, chọn cá thể tốt nhất trong nhóm đó.	Cân bằng giữa khai thác và khám phá	Phụ thuộc kích thước nhóm
Rank Selection (Xếp hạng)	Xếp hạng cá thể theo fitness, chọn theo vị trí	Tránh việc cá thể quá mạnh chi phối	Mất thời gian sắp xếp

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Chọn lọc

Ví dụ 1: Roulette Wheel Selection

Giả sử có 4 cá thể với fitness như sau:

Cá thể	Fitness
A	10
B	20
C	40
D	30

Tổng fitness = 100

→ Xác suất chọn từng cá thể:

- A: 10%
- B: 20%
- C: 40%
- D: 30%

Cá thể C dễ được chọn hơn, nhưng A vẫn **có khả năng nhỏ được chọn**, giúp duy trì sự đa dạng.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Chọn lọc

Ví dụ 2: Tournament Selection (kích thước 3)

Quần thể có 6 cá thể: $x1, x2, ..., x6$ với fitness khác nhau.

- Bước 1: Chọn ngẫu nhiên 3 cá thể: $x2, x4, x5$
- Bước 2: So sánh fitness: Giả sử $x4$ là tốt nhất → chọn $x4$ vào thế hệ mới.

Lặp lại để chọn thêm cá thể khác.

Ưu điểm: **Không cần tính xác suất hay tổng fitness**.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Chọn lọc

Ví dụ 3: Rank Selection
Giả sử có 4 cá thể với fitness như sau:

Hạng	Cá thể	Xác suất chọn
1 (tốt nhất)	x3	30%
2	x1	25%
3	x4	20%
4	x2	15%
5 (tệ nhất)	x5	10%

Dù fitness gốc chênh lệch nhiều, nhưng xác suất được **phân bổ công bằng hơn**, giúp tránh bị **chi phối quá mức** bởi cá thể mạnh.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Đột biến

Định nghĩa:

Tiêu chí dừng là điều kiện để kết thúc quá trình tiến hóa trong các thuật toán như Giải thuật Di truyền (GA) hay Lập trình tiến hóa (EP). Việc đặt tiêu chí dừng hợp lý giúp:

- Tránh lãng phí tài nguyên tính toán.
- Đảm bảo chất lượng lời giải đủ tốt mà không chạy vô hạn.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Tiêu chí dừng

Các tiêu chí dừng phổ biến:

1. **Số thế hệ tối đa (Max Generations):**
    1. Dừng khi thuật toán đã lặp qua một số thế hệ xác định trước, ví dụ: 100 hoặc 1000 thế hệ.
    2. Phổ biến và dễ kiểm soát.
    3. Không đảm bảo tìm được lời giải tốt nhất.
2. **Ngưỡng hàm thích nghi (Fitness Threshold):**
    1. Dừng khi giá trị hàm thích nghi của cá thể tốt nhất đạt hoặc vượt ngưỡng mong muốn.
    2. Phù hợp với bài toán có mục tiêu rõ ràng.
    3. Cần biết trước giá trị tối ưu kỳ vọng.
3. **Không cải thiện sau N thế hệ (No Improvement):**
    1. Dừng nếu sau N thế hệ liên tiếp mà không có cải thiện đáng kể về fitness.
    2. Giúp tránh chạy quá lâu nếu đã đạt cực trị cục bộ.
    3. Có thể dừng sớm nếu ngẫu nhiên bị "kém may".
4. **Hội tụ quần thể (Population Convergence):**
    1. Dừng khi hầu hết các cá thể trong quần thể đều giống nhau (ít đa dạng → đã hội tụ).
    2. Dựa vào đặc trưng tiến hóa tự nhiên.
    3. Có thể rơi vào cực trị cục bộ nếu không có đột biến đủ mạnh.

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

Đột biến

- Quần thể: 10 cá thể (số thực)
- Đột biến: Thêm nhiễu Gaussian
- Chọn lọc: Tournament size = 3

Áp dụng tiêu chí dừng:

6.3 Các nguyên tắc cơ bản của lập trình tiến hoá

```
best_fitness = 0
no_improve_count = 0
for generation in range(100):
    # Giả sử best_fitness được cập nhật mỗi vòng
    if current_best_fitness > best_fitness + 1e-3:
        best_fitness = current_best_fitness
        no_improve_count = 0
    else:
        no_improve_count += 1

    if best_fitness >= 74.9:
        print("Dừng vì đạt ngưỡng fitness")
        break
    if no_improve_count >= 10:
        print("Dừng vì không cải thiện")
        break
```

→ Tại thế hệ thứ 34, fitness đạt 75.1 → thuật toán dừng sớm.

6.4 Giải thuật di truyền (GA)

Giải thuật di truyền (Genetic Algorithm - GA) là một nhánh của lập trình tiến hóa, sử dụng các toán tử lai ghép, đột biến và chọn lọc để tìm kiếm lời giải tối ưu.

Quy trình chi tiết:

01 **Khởi tạo:** Tạo quần thể ban đầu với các cá thể được mã hóa (ví dụ: chuỗi nhị phân, số thực).

---

02 **Đánh giá độ thích nghi:** Tính toán giá trị hàm thích nghi cho mỗi cá thể

---

03 **Chọn lọc –** Chọn các cá thể tốt nhất

- **Roulette Wheel Selection**: Cá thể có độ thích nghi cao hơn có xác suất được chọn cao hơn.
- **Tournament Selection**: So sánh một nhóm nhỏ cá thể và chọn cá thể tốt nhất.

04 **Lai ghép (Crossover)**: Kết hợp hai cá thể cha mẹ để tạo ra cá thể con, ví dụ:

- Lai ghép một điểm: Chọn một điểm ngẫu nhiên và hoán đổi phần sau của hai chuỗi.
- Lai ghép hai điểm: Chọn hai điểm và hoán đổi đoạn giữa.

05 **Đột biến**: Thay đổi ngẫu nhiên một phần của cá thể, ví dụ: đảo bit trong chuỗi nhị phân.

---

06 **Thay thế**: Cập nhật quần thể bằng các cá thể mới và lặp lại.

6.4 Giải thuật di truyền (GA)

Ví dụ minh hoạ

Bài toán ba lô (Knapsack Problem): Chọn một tập hợp các vật phẩm sao cho tổng giá trị lớn nhất trong khi tổng trọng lượng không vượt quá giới hạn.

- Mã hóa: Mỗi cá thể là một chuỗi nhị phân, trong đó (1) biểu thị chọn vật phẩm, (0) biểu thị không chọn.
- Hàm thích nghi: Tổng giá trị của các vật phẩm được chọn, với điều kiện tổng trọng lượng không vượt quá giới hạn.
- Lai ghép: Sử dụng lai ghép một điểm để tạo cá thể con.
- Đột biến: Đảo ngẫu nhiên một bit trong chuỗi.

Ưu điểm:

- Khả năng tìm kiếm toàn cục trong không gian giải pháp lớn.
- Linh hoạt, áp dụng được cho nhiều bài toán khác nhau.

Nhược điểm:

- Cần điều chỉnh nhiều tham số (tỷ lệ lai ghép, tỷ lệ đột biến, kích thước quần thể).
- Có thể hội tụ sớm vào cực trị cục bộ.

6.5 Giải thuật bầy đàn (PSO)

Định nghĩa:

Giải thuật PSO (Particle Swarm Optimization) là một phương pháp tối ưu hóa dựa trên hành vi tập thể của đàn chim, cá hoặc côn trùng, trong đó các cá thể (gọi là hạt – particle) cùng nhau tìm kiếm lời giải tối ưu trong không gian.

Nguyên lý hoạt động:

- Mỗi hạt i đại diện cho một lời giải trong không gian tìm kiếm.
- Mỗi hạt có:
  - **Vị trí hiện tại:** $x_i(t) \in R^d$
  - **Vận tốc hiện tại:** $v_i(t) \in R^d$
  - **Vị trí tốt nhất từng đạt được:** $pbest_i$
  - **Vị trí tốt nhất của toàn bộ đàn:** $gbest_i$

6.5 Giải thuật bầy đàn (PSO)

Cập nhật vận tốc và vị trí:

1. Cập nhật vận tốc:

$$v_i(t+1) = w.v_i(t) + c_1.r_1.\left(pbest_i - x_i(t)\right) + c_2.r_2.\left(gbest_i - x_i(t)\right)$$

Trong đó:

- w: hệ số quán tính (thường trong khoảng [0.4, 0.9]
- c_1, c_2: hệ số học tập (thường đặt là 2.0)
- r_1, r_2: số ngẫu nhiên trong [0, 1]

2. Cập nhật vị trí:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

6.5 Giải thuật bầy đàn (PSO)

Quy trình thuật toán PSO

1. Khởi tạo ngẫu nhiên $\boldsymbol{x_i(0)}$ và $\boldsymbol{v_i(0)}$ cho tất cả các hạt.
2. Đánh giá hàm thích nghi (fitness function) tại mỗi $\boldsymbol{x_i}$.
3. Cập nhật $\boldsymbol{pbest_i}$ nếu cá thể đạt điểm tốt hơn trước.
4. Cập nhật $\boldsymbol{gbest_i}$ nếu có hạt đạt fitness tốt nhất toàn đàn.
5. Cập nhật $\boldsymbol{v_i, x_i}$ theo công thức trên.
6. Lặp lại các bước trên cho đến khi:

- Đạt số vòng lặp tối đa, hoặc
- Đạt được sai số chấp nhận được (convergence).

6.4 Giải thuật di truyền (GA)

Ví dụ minh hoạ

$$f(x, y) = -(x^2 + y^2)$$

với ràng buộc:

$$x, y \in [-5,5]$$

Quy trình:

- Khởi tạo: 20 hạt với vị trí và vận tốc ngẫu nhiên.
- Hàm thích nghi: $f(x, y) = -(x^2 + y^2)$ Mục tiêu: tìm (x,y) sao cho hàm đạt cực đại (tức tối thiểu hóa $x^2 + y^2$)

6.4 Giải thuật di truyền (GA)

Ví dụ minh hoạ

Ưu điểm:

- Cấu trúc đơn giản, dễ lập trình và triển khai.
- Hiệu quả trong các bài toán **tối ưu liên tục** và nhiều chiều.
- Không cần tính đạo hàm của hàm mục tiêu.

Nhược điểm:

- Dễ bị **kẹt ở cực trị cục bộ** nếu không có đột biến hoặc điều chỉnh tham số hợp lý.
- Phụ thuộc khá nhiều vào các tham số: w, c_1, c_2

6.6 Ứng dụng giải bài toán tối ưu

Lập trình tiến hóa (Evolutionary Programming) và các giải thuật như Giải thuật di truyền (GA), Giải thuật bầy đàn (PSO) là những công cụ mạnh mẽ để tối ưu hóa các bài toán phức tạp, phi tuyến, không khả vi, nơi các phương pháp truyền thống như gradient descent khó áp dụng.

1. Kỹ thuật (Engineering)

Tối ưu thiết kế kỹ thuật:

- Tối ưu hình dạng cánh máy bay, ô tô để giảm lực cản khí động học nhưng vẫn đảm bảo độ bền.
- Ví dụ: Dùng GA để tìm hình dạng cánh với lực cản nhỏ nhất bằng cách mã hóa các tham số thiết kế thành chuỗi nhị phân.

Tối ưu phân bổ tài nguyên & lập lịch:

- Trong nhà máy sản xuất, cần phân bổ máy móc, nguyên vật liệu, công nhân theo thứ tự và thời gian hợp lý.
- Ví dụ: PSO được dùng để lập lịch máy CNC nhằm giảm thời gian sản xuất tổng thể (makespan).

6.6 Ứng dụng giải bài toán tối ưu

2. Học máy & Trí tuệ nhân tạo (AI & Machine Learning)

Tối ưu siêu tham số (Hyperparameter tuning):

- Tìm giá trị tối ưu cho các tham số như learning rate, số hidden layers trong neural network.
- **Ví dụ:** GA hoặc PSO dùng để chọn kết hợp tốt nhất giữa batch size, số neuron, activation function trong mạng nơ-ron.

Chọn đặc trưng (Feature Selection):

- Chọn ra tập đặc trưng nhỏ nhưng hiệu quả nhất để tăng độ chính xác của mô hình và giảm overfitting.
- **Ví dụ:** GA mã hóa mỗi đặc trưng bằng bit 0/1 (chọn hoặc bỏ), sau đó tiến hóa để tìm tập tối ưu.

6.6 Ứng dụng giải bài toán tối ưu

3. Tài chính – Kinh tế (Finance & Economics)

Tối ưu danh mục đầu tư (Portfolio Optimization):

- Cân bằng giữa lợi nhuận kỳ vọng và rủi ro, trong khi tuân thủ các ràng buộc (như tỷ trọng vốn).
- **Ví dụ:** PSO dùng để chọn tỷ lệ đầu tư vào các cổ phiếu để tối đa hóa Sharpe ratio.

Dự đoán xu hướng thị trường:

- Dùng GA để điều chỉnh tham số cho mô hình ARIMA hoặc MLP nhằm dự báo giá cổ phiếu chính xác hơn.

6.6 Ứng dụng giải bài toán tối ưu

4. Y học & Sinh học (Medical Applications)

Lập kế hoạch điều trị phóng xạ (Radiation Therapy Planning):

- Tối ưu hóa liều lượng và vị trí chiếu xạ sao cho tiêu diệt tế bào ung thư hiệu quả mà không làm tổn hại mô khỏe.
- **Ví dụ:** GA xác định góc chiếu và cường độ tia X để tối thiểu hóa thiệt hại lên mô lành.

Phân tích dữ liệu gen (Genomic Data Analysis):

- Dùng PSO để phát hiện ra các chuỗi gen có liên quan đến một bệnh cụ thể trong kho dữ liệu gen khổng lồ.

6.6 Ứng dụng giải bài toán tối ưu

5. Trò chơi và Hệ thống đa tác nhân (Games & Multi-Agent Systems)

Huấn luyện tác nhân AI trong game:

- GA và PSO giúp các bot học cách chơi tốt hơn bằng cách tối ưu chiến lược (reward).
- **Ví dụ:** PSO huấn luyện bot điều khiển xe trong game racing để tối thiểu thời gian hoàn thành vòng đua.

Chiến lược trong hệ thống đa tác nhân:

- Trong các hệ thống có nhiều AI tương tác (robotics, traffic simulation), giải thuật tiến hóa giúp các agent học cách phối hợp tối ưu.

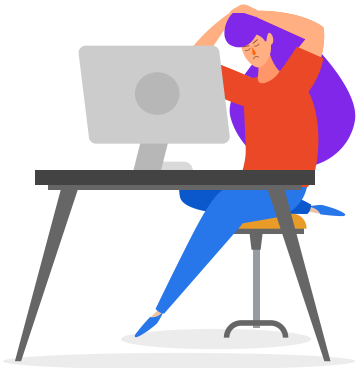
6.7 Bài tập thực hành

Bài tập 1: Giải thuật di truyền cho bài toán tối ưu hóa hàm số

Sử dụng giải thuật di truyền để tìm giá trị cực đại của hàm ($f(x) = -x^2 + 10x + 50$) trong khoảng ($x \in [0, 10]$).

Yêu cầu:

- Kích thước quần thể: 20 cá thể.
- Số thế hệ: 50.
- Tỷ lệ lai ghép: 0.8.
- Tỷ lệ đột biến: 0.1.
- Mã hóa: Sử dụng số thực cho (x).
- Chọn lọc: Sử dụng chọn lọc giải đấu (tournament selection).
- Lai ghép: Sử dụng lai ghép trung bình (average crossover).
- Đột biến: Thêm nhiễu Gaussian (N(0, 0.5)).



```python
import numpy as np

# Hàm thích nghi
def fitness(x):
    return -x**2 + 10*x + 50

# Khởi tạo quần thể
def initialize_population(pop_size, x_min, x_max):
    return np.random.uniform(x_min, x_max, pop_size)

# Chọn lọc giải đấu
def tournament_selection(population, fitness_values,
tournament_size=3):
    selected = []
    for _ in range(len(population)):
        indices = np.random.choice(len(population),
tournament_size)
        tournament_fitness = [fitness_values[i] for i in indices]
        winner_idx = indices[np.argmax(tournament_fitness)]
        selected.append(population[winner_idx])
    return np.array(selected)

# Lai ghép trung bình
def crossover(parent1, parent2, crossover_rate=0.8):
    if np.random.rand() < crossover_rate:
        return (parent1 + parent2) / 2
    return parent1

# Đột biến Gaussian
def mutation(individual, mutation_rate=0.1, sigma=0.5, x_min=0,
x_max=10):
    if np.random.rand() < mutation_rate:
        individual += np.random.normal(0, sigma)
        individual = np.clip(individual, x_min, x_max)
    return individual
```

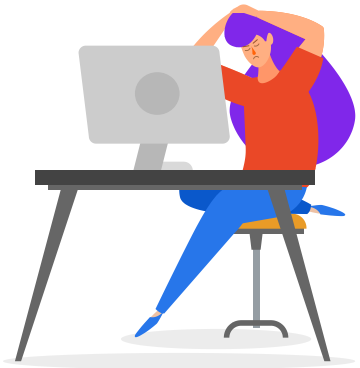
```python
# Giải thuật di truyền
def genetic_algorithm(pop_size=20, generations=50, x_min=0, x_max=10):
    population = initialize_population(pop_size, x_min, x_max)
    for gen in range(generations):
        fitness_values = np.array([fitness(x) for x in population])
        new_population = []

        # Chọn lọc
        selected = tournament_selection(population, fitness_values)

        # Lai ghép
        for i in range(0, pop_size, 2):
            parent1 = selected[i]
            parent2 = selected[min(i+1, pop_size-1)]
            child1 = crossover(parent1, parent2)
            child2 = crossover(parent1, parent2)
            new_population.append(child1)
            new_population.append(child2)

        # Đột biến
        new_population = [mutation(x) for x in new_population]
        population = np.array(new_population)

        # In kết quả tốt nhất mỗi thế hệ
        best_fitness = np.max(fitness_values)
        best_x = population[np.argmax(fitness_values)]
        print(f"Thế hệ {gen+1}: x = {best_x:.4f}, f(x) =
{best_fitness:.4f}")

    return population[np.argmax([fitness(x) for x in population])]

# Chạy chương trình
best_solution = genetic_algorithm()
print(f"Giải pháp tốt nhất: x = {best_solution:.4f}, f(x) =
{fitness(best_solution):.4f}")
```

6.7 Bài tập thực hành

Bài tập 2: Giải thuật bày đàn cho bài toán tối ưu hóa hàm 2 biến

Sử dụng PSO để tìm giá trị cực đại của hàm ($f(x,y) = -(x^2 + y^2)$) trong khoảng ($x, y \in = [-5, 5]$).

- **Yêu cầu**:
  - Số hạt: 30.
  - Số lần lặp: 100.
  - Tham số: ($w = 0.7, c_1 = 2, c_2 = 2$).
  - Khởi tạo vị trí và vận tốc ngẫu nhiên trong khoảng [-5, 5].



```python
import numpy as np

# Hàm thích nghi
def fitness(x, y):
    return -(x**2 + y**2)

# Khởi tạo quần thể hạt
def initialize_particles(num_particles, x_min, x_max):
    positions = np.random.uniform(x_min, x_max, (num_particles, 2))
    velocities = np.random.uniform(-1, 1, (num_particles, 2))
    pbest_positions = positions.copy()
    pbest_values = np.array([fitness(x, y) for x, y in positions])
    gbest_idx = np.argmax(pbest_values)
    gbest_position = pbest_positions[gbest_idx].copy()
    return positions, velocities, pbest_positions, pbest_values, gbest_position

# Giải thuật PSO
def particle_swarm_optimization(num_particles=30, iterations=100, x_min=-5, x_max=5, w=0.7, c1=2, c2=2):
    positions, velocities, pbest_positions, pbest_values, gbest_position = initialize_particles(num_particles, x_min, x_max)

    for it in range(iterations):
        for i in range(num_particles):
            # Đánh giá độ thích nghi
            current_value = fitness(positions[i, 0], positions[i, 1])
            if current_value > pbest_values[i]:
                pbest_values[i] = current_value
                pbest_positions[i] = positions[i].copy()

            # Cập nhật gbest
            if current_value > fitness(gbest_position[0], gbest_position[1]):
                gbest_position = positions[i].copy()

        # Cập nhật vận tốc và vị trí
        for i in range(num_particles):
            r1, r2 = np.random.rand(2)
            velocities[i] = (w * velocities[i] +
                             c1 * r1 * (pbest_positions[i] - positions[i]) +
                             c2 * r2 * (gbest_position - positions[i]))
            positions[i] += velocities[i]
            positions[i] = np.clip(positions[i], x_min, x_max)

        # In kết quả tốt nhất mỗi lần lặp
        best_value = fitness(gbest_position[0], gbest_position[1])
        print(f"Lần lặp {it+1}: (x, y) = ({gbest_position[0]:.4f}, {gbest_position[1]:.4f}), f(x, y) = {best_value:.4f}")

    return gbest_position

# Chạy chương trình
best_solution = particle_swarm_optimization()
print(f"Giải pháp tốt nhất: (x, y) = ({best_solution[0]:.4f}, {best_solution[1]:.4f}), f(x, y) = {fitness(best_solution[0], best_solution[1]):.4f}")
```