Lab 4.1: Sử dụng giải thuật di truyền để tạo mật khẩu

ThS. Lê Thị Thùy Trang

2025-1-12

Trong bài thực hành này, sinh viên sẽ tìm hiểu và phân tích mã nguồn mô phỏng thuật toán di truyền ("genetic algorithm") để giải quyết bài toán tìm mật khẩu.

1. Giải thuật di truyền

Giải thuật di truyền (GA) là thuật toán tối ưu hóa dựa trên các nguyên tắc của chọn lọc tự nhiên và di truyền học. Đây là một phương pháp tìm kiếm heuristic được sử dụng để tìm các giải pháp gần đúng cho các bài toán tối ưu hóa và tìm kiếm. GA lấy cảm hứng từ quá trình tiến hóa, trong đó các sinh vật có đặc điểm ưu việt có nhiều khả năng sống sót và sinh sản hơn, truyền lại các đặc điểm đó cho thế hệ sau.

Trong một thuật toán di truyền, một quần thể các giải pháp tiềm năng cho một vấn đề được tạo ra và đánh giá độ thích nghi, đây là thước đo mức độ đáp ứng các tiêu chí của một giải pháp thành công. Các cá thể có độ thích nghi cao nhất được chọn để sinh sản, và các toán tử di truyền như lai ghép (crossover) và đột biến (mutation) được áp dụng để tạo ra các con cái mới. Các con cái mới sau đó được đánh giá độ thích nghi, và quá trình này được lặp lại cho đến khi tìm được một giải pháp thỏa mãn hoặc đạt đến điều kiện kết thúc.

GA có thể được áp dụng cho một loạt các bài toán tối ưu hóa, bao gồm cả những bài toán khó hoặc không thể giải quyết bằng các phương pháp toán học hoặc phân tích truyền thống. Chúng đã được sử dụng trong nhiều ứng dụng như thiết kế kỹ thuật, lập lịch trình và tối ưu hóa danh mục đầu tư tài chính. Mặc dù GA không đảm bảo tìm ra giải pháp tối ưu, nhưng chúng thường có thể tìm ra các giải pháp tốt trong một khoảng thời gian hợp lý, khiến chúng trở thành một công cụ giá trị để giải quyết các bài toán tối ưu hóa phức tạp.

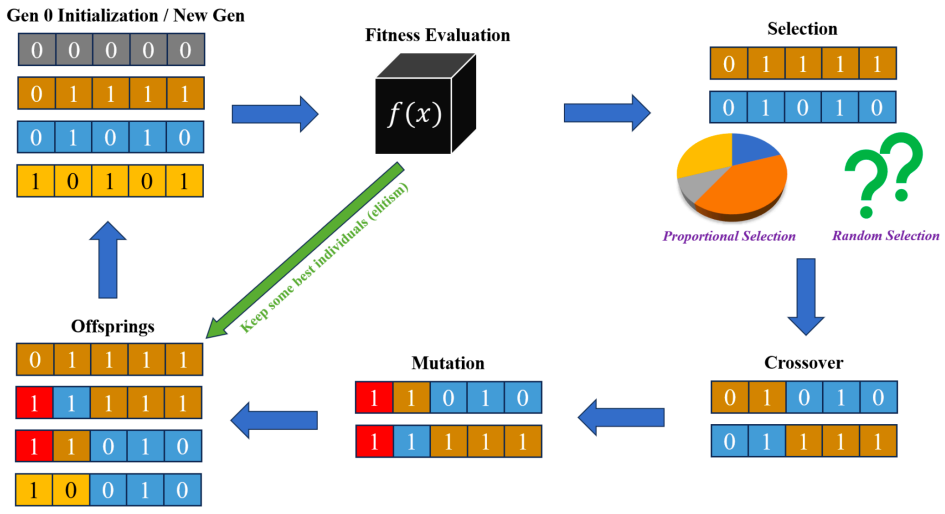

Figure 1: Pipeline cơ bản của giải thuật di truyền

Giải thuật di truyền trải qua một loạt các bước mô phỏng các quá trình tiến hóa tự nhiên để tìm ra các giải pháp tối ưu. Các bước này cho phép quần thể tiến hóa qua nhiều thế hệ, cải thiện chất lượng các giải pháp.

- Bước 1: Khởi tạo: Đầu tiên, chúng ta tạo ra một quần thể ban đầu gồm các cá thể ngẫu nhiên. Bước này tạo ra một tập hợp đa dạng các giải pháp tiềm năng để bắt đầu thuật toán.
- Bước 2: Đánh giá: Tiếp theo, chúng ta cần tính toán mức độ phù hợp của từng cá thể trong quần thể. Ở đây, chúng ta sử dụng hàm độ phù hợp để đánh giá mức độ tốt của từng giải pháp.
- Bước 3: Lựa chọn: Sử dụng các tiêu chí lựa chọn, chúng ta chọn các cá thể để sinh sản dựa trên mức độ phù hợp của chúng. Bước này xác định những cá thể nào sẽ là cha mẹ.
- Bước 4: Lai ghép: Tiếp theo là lai ghép. Bằng cách kết hợp vật liệu di truyền của các bậc cha mẹ được chọn, chúng ta áp dụng các kỹ thuật lai ghép để tạo ra các giải pháp hoặc thế hệ con mới.
- Bước 5: Đột biến: Để duy trì sự đa dạng trong quần thể của chúng ta, chúng ta cần đưa các đột biến ngẫu nhiên vào thế hệ con.
- Bước 6: Thay thế: Tiếp theo, chúng ta thay thế một số hoặc toàn bộ quần thể cũ bằng thế hệ con mới, bằng cách xác định những cá thể nào sẽ chuyển sang thế hệ tiếp theo.
- Bước 7: Lặp lại: Các bước trước 2-6 được lặp lại trong một số thế hệ nhất định hoặc cho đến khi đáp ứng được điều kiện kết thúc. Vòng lặp này cho phép quần thể tiến hóa theo thời gian, hy vọng sẽ đưa đến một giải pháp tốt.

2. Mô tả bài toán

Cho một tập các ký tự hợp lệ "`abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}`"

Với kích thước quần thể là 100, hãy xây dựng chương trình sử dụng thuật toán di truyền để tạo ra chuỗi mật khẩu: "hoilamgi".

Chương trình cần tìm ra chuỗi ký tự giống với mật khẩu và trả về số thế hệ cần thiết để đạt được kết quả.

3. Triển khai thuật toán GA bằng python

Chương trình thực hiện các yêu cầu của bài toán, theo các bước sau

3.1 Import the necessary libraries

```
import random
import numpy as np
```

3.2 Khởi tạo các tham số

Ta định nghĩa các hằng số sau

```
POPULATION_SIZE = 100  # Kích thước quần thể
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
TARGET = "hoilamgi"  # Chuỗi mục tiêu
```

Trong đó:

- `POPULATION_SIZE`: Số lượng cá thể trong quần thể ban đầu.
- `GENES`: Tập hợp tất cả các ký tự hợp lệ có thể xuất hiện trong chuỗi.
- `TARGET`: Chuỗi mục tiêu mà thuật toán phải tìm ra.

3.3 Khởi tạo cá thể trong quần thể

Lớp `Individual` đại diện cho mỗi cá thể trong quần thể

Phương thức khởi tạo

```
class Individual:
  def __init__(self, chromosome):
    self.chromosome = chromosome
    self.fitness = self.cal_fitness()
```

Trong đó:

- `chromosome`: chuỗi ký tự đại diện cho cá thể
- `fitness`: độ thích nghi của cá thể, được tính bằng `cal_fitness`.

Tiếp theo, để tạo một gen ngẫu nhiên bằng cách chọn một ký tự trong tập hợp `GENES`, ta sử dụng phương thức `mutated_genes`

```
  @classmethod
  def mutated_genes(cls):
    '''
    Create random genes for mutation
    '''
    return random.choice(GENES)
```

Sau đó, tạo một chuỗi ký tự ngẫu nhiên ban đầu, có độ dài bằng độ dài của chuỗi mục tiêu `TARGET`, sử dụng phương thức `create_gnome`

```
  @classmethod
  def create_gnome(cls):
    '''
    Create chromosome or string of genes
    '''
    return [cls.mutated_genes() for _ in range(len(TARGET))]
```

Để tạo ra một cá thể con mới, ta thực hiện lai ghép với một cá thể khác, bằng phương thức `mate`

Xác suất lai ghép:

- 45% lấy gen từ cha mẹ 1
- 45% lấy gen từ cha mẹ 2
- 10% thực hiện đột biến tạo ra gen ngẫu nhiên

```
def mate(self, par2):
  '''
  Perform mating and produce new offspring
  '''
  child_chromosome = []
  for gp1, gp2 in zip(self.chromosome, par2.chromosome):
    prob = random.random()
    if prob < 0.45:
```

```
      child_chromosome.append(gp1)
    elif prob < 0.90:
      child_chromosome.append(gp2)
    else:
      child_chromosome.append(self.mutated_genes())
  return Individual(child_chromosome)
```

Tính độ thích nghi của cá thể dựa trên số lượng ký tự khác biệt so với chuỗi mục tiêu, độ thích nghi càng nhỏ, cá thể càng tốt

```
  def cal_fitness(self):
    '''
    Calculate fitness score
    '''
    return sum(1 for gs, gt in zip(self.chromosome, TARGET) if gs != gt)
```

3.4 Main function

Sau khi tạo ra một quần thể gồm các cá thể có nhiễm sắc thể ngẫu nhiên, ta lặp lại các bước sau cho đến khi tìm được cá thể có fitness bằng 0:

- Sắp xếp quần thể theo fitness.
- Kiểm tra nếu cá thể tốt nhất có fitness bằng 0, dừng thuật toán.
- Tạo thế hệ mới bằng cách:
  - Giữ lại 10% cá thể tốt nhất (elitism).
  - Lai ghép các cá thể từ 50% tốt nhất để tạo ra 90% cá thể còn lại.
- In kết quả của mỗi thế hệ.

```
def main():
    global POPULATION_SIZE

    # Initialize population
    population = [Individual(Individual.create_gnome()) for _ in range(POPULATION_SIZE)]

    generation = 1
    found = False

    while not found:
        # Sort population by fitness
        population = sorted(population, key=lambda x: x.fitness)

        # Check if the best individual has fitness 0
        if population[0].fitness == 0:
            found = True
            break

        # Generate new generation
        new_generation = []

        # Elitism: Keep top 10% of the population
        s = int((10 * POPULATION_SIZE) / 100)
        new_generation.extend(population[:s])
```

```
        # From 50% of the fittest population, mate to produce offspring
        s = int((90 * POPULATION_SIZE) / 100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
            child = parent1.mate(parent2)
            new_generation.append(child)

        population = new_generation

        # Print progress
        print(f"Generation: {generation}\tString: {''.join(population[0].chromosome)}\
        \tFitness: {population[0].fitness}")
        generation += 1

    # Print final result
    print(f"Generation: {generation}\tString: {''.join(population[0].chromosome)}\
    \tFitness: {population[0].fitness}")

if __name__ == '__main__':
    main()
```

4. Thực hành

Sinh viên thử mở rộng chương trình bằng cách:

Q1: Thay đổi kích thước quần thể `POPULATION_SIZE` và quan sát sự khác biệt về tốc độ hội tụ.

Q2: Thay đổi xác suất lai ghép và đột biến trong phương thức mate để tìm ra thông số tối ưu.

Q3: Thay đổi cách tính độ thích nghi trong phương thức `cal_fitness`: Thay vì dựa vào số ký tự khác biệt, hãy thử tính độ thích nghi dựa vào tổng khoảng cách ASCII giữa các ký tự.