Lab 4: Thực hành giải thuật di truyền

ThS. Lê Thị Thùy Trang

2025-1-12

Mục tiêu của bài thực hành này là thiết kế và triển khai một giải thuật di truyền (Genetic Algorithm - GA) cho một tác tử có thể sơn phòng một cách hiệu quả.

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.

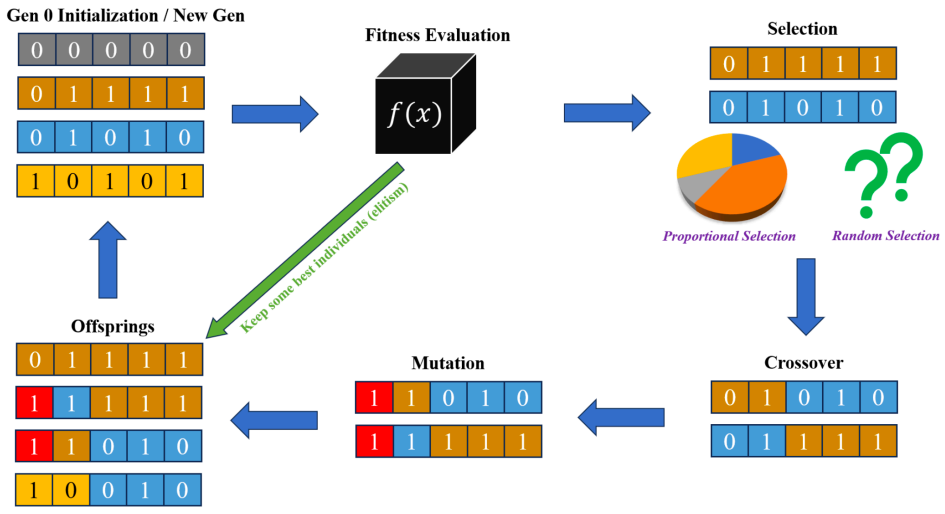

Figure 1: Pipeline cơ bản của giải thuật di truyền

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

Hãy tưởng tượng, ta có một căn phòng chữ nhật được biểu diễn dưới dạng một ma trận `2D`, có kích thước `20*40` và một tác tử robot sơn phòng. Ban dầu, tất cả các ô trong phòng đều có giá trị `0` tương ứng với chưa được sơn. Mỗi khi robot sơn được một ô trên sàn phòng, giá trị này chuyển thành `1`.

Robot này bắt đầu ở một vị trí ngẫu nhiên trong phòng và có thể di chuyển theo 4 hướng: lên, xuống, trái, phải và sơn liên tục. Robot có thể thực hiện các hành động sau:

- Tiến lên: di chuyển theo hướng hiện tại;
- Quay trái: Thay đổi hướng sang trái;
- Quay phải: Thay đổi hướng sang phải;
- Quay ngẫu nhiên: Chọn một hướng ngẫu nhiên.
- Robot có thể di chuyển qua các cạnh của phòng. Tức là, nếu robot đi ra khỏi cạnh phải, nó sẽ xuất hiện lại ở cạnh trái.

Giả định rằng ta có lượng sơn vừa đủ để phủ kín sàn phòng. Tức là việc đi qua một vị trí nhiều lần hoặc robot đứng yên tại chỗ là lãng phí và có thể khiến sàn của căn phòng không được sơn kín.

Giải thuật di truyền được triển khai bằng `Python` và sử dụng thư viện `NumPy` để thực hiện các tính toán trên mảng. Thuật toán di truyền được kiểm tra trên một tập hợp các phòng được định nghĩa trước, hướng đến mục tiêu cuối cùng là tìm ra một nhiễm sắc thể tối ưu giúp robot sơn phòng một cách hiệu quả nhất, tức là sơn được nhiều khu vực nhất với lượng sơn cho trước.

3. Triển khai giải thuật di truyền bằng Python

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

3.1 Import the necessary libraries

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

3.2 Helper functions

3.2.1 Initialization

Trong GA, quần thể là một tập hợp các cá thể (nhiễm sắc thể), mỗi cá thể đại diện cho một giải pháp tiềm năng cho bài toán.

Hàm `create_chromosome()` tạo ra một nhiễm sắc thể ngẫu nhiên, là một mảng gồm `54` phần tử, mỗi phần tử có giá trị từ 0 đến 3 (đại diện cho các hành động của robot). Cụ thể:

- `0`: Tiến lên
- `1`: Quay trái
- `2`: Quay phải
- `3`: Quay ngẫu nhiên

Mỗi phần tử trong nhiễm sắc thể tương ứng với một trạng thái cụ thể của robot, được xác định bởi:

- `c`: trạng thái của ô hiện tại (0: chưa sơn, 1: đã sơn)
- `f`: trạng thái của ô phía trước robot
- `l`: trạng thái của ô bên trái robot
- `r`: trạng thái của ô bên phải robot

Chỉ số trạng thái được tính bằng công thức:

$$\text{state} = (27c + 9f + 3l + r) \bmod 54$$

```
#Function to generate a random chromosome
def create_chromosome():
  #Create an array of 54 elements, values from 0 to 3
  return np.random.randint(0, 4, size=54)
```

Hàm `create_population(pop_size)` tạo ra một quần thể gồm `pop_size` nhiễm sắc thể ngẫu nhiên.

```
#Function to generate a population
def create_population(pop_size):
  return np.array([create_chromosome() for _ in range(pop_size)])
```

3.2.2 Fitness Evaluation

Hàm `evaluate_population(population, room)` tính toán độ thích nghi (fitness) của từng cá thể trong quần thể.

Độ thích nghi là một giá trị đánh giá mức độ tốt của một giải pháp. Trong bài toán này, độ thích nghi được tính bằng hiệu suất sơn phòng của robot (tỷ lệ diện tích được sơn so với diện tích có thể sơn).

Hàm này sẽ sử dụng hàm `painter_play()` để mô phỏng hành vi của robot dựa trên nhiễm sắc thể và trả về giá trị độ thích nghi tương ứng.

```
# Function to evaluate population fitness
def evaluate_population(population, room):
    fitness = []
    for chromosome in population:
        efficiency, _ = painter_play(chromosome, room)
        fitness.append(efficiency)
    return np.array(fitness)
```

3.2.3 Selection

Hàm `select_parents(population, fitness, num_parents)`: chọn ra các cá thể tốt nhất (cha mẹ) từ quần thể hiện tại để tạo ra thế hệ tiếp theo.

Trong GA, chọn lọc là quá trình chọn ra các cá thể có độ thích nghi cao hơn để tham gia vào quá trình sinh sản (lai ghép và đột biến).

Hàm này sử dụng phương pháp chọn lọc tỷ lệ (roulette wheel selection), trong đó xác suất một cá thể được chọn tỷ lệ thuận với độ thích nghi của nó.

```
# Function to select parents
def select_parents(population, fitness, num_parents):
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        idx = np.random.choice(np.arange(len(population)),\
        p = fitness/fitness.sum())
        parents[i, :] = population[idx, :]
    return parents
```

3.2.4 Crossover

Hàm `crossover(parents, offspring_size)`: tạo ra các cá thể con từ các cá thể cha mẹ bằng cách lai ghép.

Hàm này sử dụng phương pháp lai ghép một điểm (single-point crossover): chọn một điểm ngẫu nhiên trên nhiễm sắc thể, sau đó kết hợp phần đầu của cha mẹ 1 và phần cuối của cha mẹ 2 để tạo ra cá thể con.

```
# Function to cross
def crossover(parents, offspring_size):
    offspring = np.empty(offspring_size)
    crossover_point = np.random.randint(1, offspring_size[1])
    for k in range(offspring_size[0]):
        parent1_idx = k % parents.shape[0]
        parent2_idx = (k + 1) % parents.shape[0]
        offspring[k, :crossover_point] = parents[parent1_idx,:crossover_point]
        offspring[k, crossover_point:] = parents[parent2_idx,crossover_point:]
    return offspring
```

3.2.5 Mutation

Hàm `mutate(offspring, mutation_rate)`: duyệt qua từng gen của từng cá thể con với xác suất `mutation_rate`, thay đổi giá trị của gen đó thành một giá trị ngẫu nhiên từ 0 đến 3.

```
# Function to mutate
def mutate(offspring, mutation_rate):
```

```
    for idx in range(offspring.shape[0]):
        for gene in range(offspring.shape[1]):
            if np.random.rand() < mutation_rate:
                offspring[idx, gene] = np.random.randint(0, 4)
    return offspring
```

3.3 Hàm mô phỏng robot sơn phòng

Hàm `painter_play(chromosome, room)` đóng vai trò mô phỏng hành vi của robot sơn phòng dựa trên một nhiễm sắc thể (chromosome) cụ thể. Dưới đây là giải thích chi tiết về mục đích và cách hoạt động của hàm này:

3.3.1 Mục đích của hàm `painter_play`

Mô phỏng hành vi của robot Hàm này mô phỏng cách robot di chuyển và sơn phòng dựa trên các hành động được mã hóa trong nhiễm sắc thể.

Nhiễm sắc thể đóng vai trò như một bộ quy tắc (rule set) quyết định hành động của robot trong từng tình huống cụ thể.

Đánh giá hiệu suất của nhiễm sắc thể Hàm này trả về hiệu suất sơn phòng (efficiency), là tỷ lệ diện tích được sơn so với tổng diện tích có thể sơn.

Hiệu suất này được sử dụng để đánh giá độ thích nghi (fitness) của nhiễm sắc thể trong quá trình tiến hóa.

Tạo dữ liệu đầu ra để trực quan hóa Hàm này cũng trả về một ma trận painted, biểu diễn các khu vực đã được sơn trong phòng. Ma trận này có thể được sử dụng để hiển thị kết quả sơn phòng dưới dạng hình ảnh

3.3.2 Cách hàm `painter_play` hoạt động

Khởi tạo:

- Vị trí ban đầu của robot: Robot được đặt ở một vị trí ngẫu nhiên trong phòng `(x, y)`.
- Hướng ban đầu của robot: Robot được định hướng ngẫu nhiên (`0`: lên, `1`: phải, `2`: xuống, `3`: trái).
- Ma trận painted: Được khởi tạo để theo dõi các khu vực đã được sơn.

Mô phỏng từng bước di chuyển: Robot thực hiện tối đa `max_steps` bước di chuyển (để tránh vòng lặp vô hạn).

Trong mỗi bước:

- Kiểm tra ô hiện tại: Nếu ô hiện tại chưa được sơn `(room[x, y] == 0)`, robot sẽ sơn ô đó `(painted[x, y] = 1)`.
- Xác định trạng thái hiện tại:
  - `c`: Trạng thái của ô hiện tại (`0`: chưa sơn, `1`: đã sơn).
  - `f`: Trạng thái của ô phía trước robot.
  - `l`: Trạng thái của ô bên trái robot.
  - `r`: Trạng thái của ô bên phải robot.
- Tính chỉ số trạng thái (state): Chỉ số trạng thái được tính dựa trên các giá trị `c`, `f`, `l`, `r` và được giới hạn trong phạm vi `0-53`.

- Lấy hành động từ nhiễm sắc thể: Hành động được xác định bởi giá trị tại vị trí `state` trong nhiễm sắc thể.
- Thực hiện hành động:
  - `0`: Tiến lên.
  - `1`: Quay trái.
  - `2`: Quay phải.
  - `3`: Quay ngẫu nhiên.
- Di chuyển robot: Robot di chuyển theo hướng hiện tại.

Kết thúc mô phỏng: Sau khi hoàn thành tất cả các bước, hàm tính toán hiệu suất sơn phòng:

$$\text{efficiency} = \frac{\text{Tổng số ô đã sơn}}{\text{Tổng số ô có thể sơn}}$$

Hàm này trả về giá trị `efficiency` và ma trận `painted`.

3.3.3 Vai trò của hàm `painter_play`

Đánh giá độ thích nghi: Hàm này được sử dụng trong hàm `evaluate_population()` để tính toán độ thích nghi của từng nhiễm sắc thể trong quần thể.

Độ thích nghi (fitness) chính là hiệu suất sơn phòng (efficiency) được trả về bởi `painter_play`.

Tối ưu hóa nhiễm sắc thể: Thông qua quá trình tiến hóa (chọn lọc, lai ghép, đột biến), giải thuật di truyền dần dần tìm ra nhiễm sắc thể giúp robot sơn phòng hiệu quả nhất.

Hàm `painter_play` là công cụ để đánh giá chất lượng của từng nhiễm sắc thể.

Trực quan hóa kết quả: Sau khi tìm được nhiễm sắc thể tốt nhất, hàm `painter_play` được sử dụng để mô phỏng lại hành vi của robot và hiển thị kết quả sơn phòng dưới dạng hình ảnh.

```
# Function to simulate room painting robot
def painter_play(chromosome, room):
    rows, cols = room.shape
    x, y = np.random.randint(0, rows), np.random.randint(0, cols)
    direction = np.random.randint(0, 4)  # 0: up, 1: right, 2: down, 3: left
    painted = np.zeros_like(room)
    steps = 0
    max_steps = 1000  # Limit the number of steps to avoid infinite loops

    while steps < max_steps:
        if room[x, y] == 0:
            painted[x, y] = 1
        # Determine current state
        c = int(painted[x, y])  # c must be 0 or 1
        f = int(room[(x - 1) % rows, y] if direction == 0 else \
                room[x, (y + 1) % cols] if direction == 1 else \
                room[(x + 1) % rows, y] if direction == 2 else \
                room[x, (y - 1) % cols])
        f = 1 if f == 1 else 0  # Make sure f is only 0 or 1
        l = int(room[x, (y - 1) % cols] if direction == 0 else \
                room[(x - 1) % rows, y] if direction == 1 else \
```

```
                room[x, (y + 1) % cols] if direction == 2 else \
                room[(x + 1) % rows, y])
        l = 1 if l == 1 else 0  # Make sure l is only 0 or 1
        r = int(room[x, (y + 1) % cols] if direction == 0 else \
                room[(x + 1) % rows, y] if direction == 1 else \
                room[x, (y - 1) % cols] if direction == 2 else \
                room[(x - 1) % rows, y])
        r = 1 if r == 1 else 0  # Make sure r is only 0 or 1
        # Calculate status index and limit in range 0-53
        state = int((c * 27 + f * 9 + l * 3 + r) % 54)
        # Get action from chromosome
        action = chromosome[state]
        # Take action
        if action == 1:  # Turn left
            direction = (direction - 1) % 4
        elif action == 2:  # Turn right
            direction = (direction + 1) % 4
        elif action == 3:  # Random spin
            direction = np.random.randint(0, 4)
        # Move
        if direction == 0:
            x = (x - 1) % rows
        elif direction == 1:
            y = (y + 1) % cols
        elif direction == 2:
            x = (x + 1) % rows
        elif direction == 3:
            y = (y - 1) % cols
        steps += 1

    efficiency = np.sum(painted) / np.sum(room == 0)
    return efficiency, painted
```

3.4 Hàm thực hiện toàn bộ quy trình GA qua nhiều thế hệ

Hàm `genetic_algorithm()`: thực hiện các bước của GA (khởi tạo, đánh giá, chọn lọc, lai ghép, đột biến, thay thế) trong một vòng lặp qua `num_generations` thế hệ.

Sau khi tạo ra các cá thể con thông qua lai ghép và đột biến, quần thể mới được hình thành bằng cách thay thế một phần quần thể cũ bằng các cá thể con.

Sau mỗi thế hệ, độ thích nghi tốt nhất được lưu lại để theo dõi sự tiến hóa của quần thể.

```
#function to run genetic algorithm
def genetic_algorithm(room, pop_size=50, num_generations=200, mutation_rate=0.002):
    population = create_population(pop_size)
    best_fitness = []
    for generation in range(num_generations):
        fitness = evaluate_population(population, room)
        best_fitness.append(np.max(fitness))
        parents = select_parents(population, fitness, pop_size//2)
        offspring = crossover(parents, (pop_size - parents.shape[0], population.shape[1]))
        offspring = mutate(offspring, mutation_rate)
        # Create a new population by combining parent and offspring individuals.
```

```
        population[:parents.shape[0], :] = parents
        population[parents.shape[0]:, :] = offspring
    return population, best_fitness
```

3.5 Thực thi thuật toán và hiển thị kết quả

```
# Create a room
room = np.zeros((20, 40))

# Run genetic algorithm
population, best_fitness = genetic_algorithm(room)

# Graphing fitness over generations
plt.plot(best_fitness)
plt.xlabel('Generation')
plt.ylabel('Best Fitness')
plt.title('Best Fitness vs Generation')
plt.show()

# Best chromosome test
best_chromosome = population[np.argmax(evaluate_population(population, room))]
efficiency, painted = painter_play(best_chromosome, room)
print(f"Best chromosome efficiency: {efficiency}")

# Print out the room after painting
plt.imshow(painted, cmap='gray')
plt.title('Painted Room')
plt.show()
```

Sau khi thực thi thuật toán, ta thu được kết quả như hai hình dưới đây

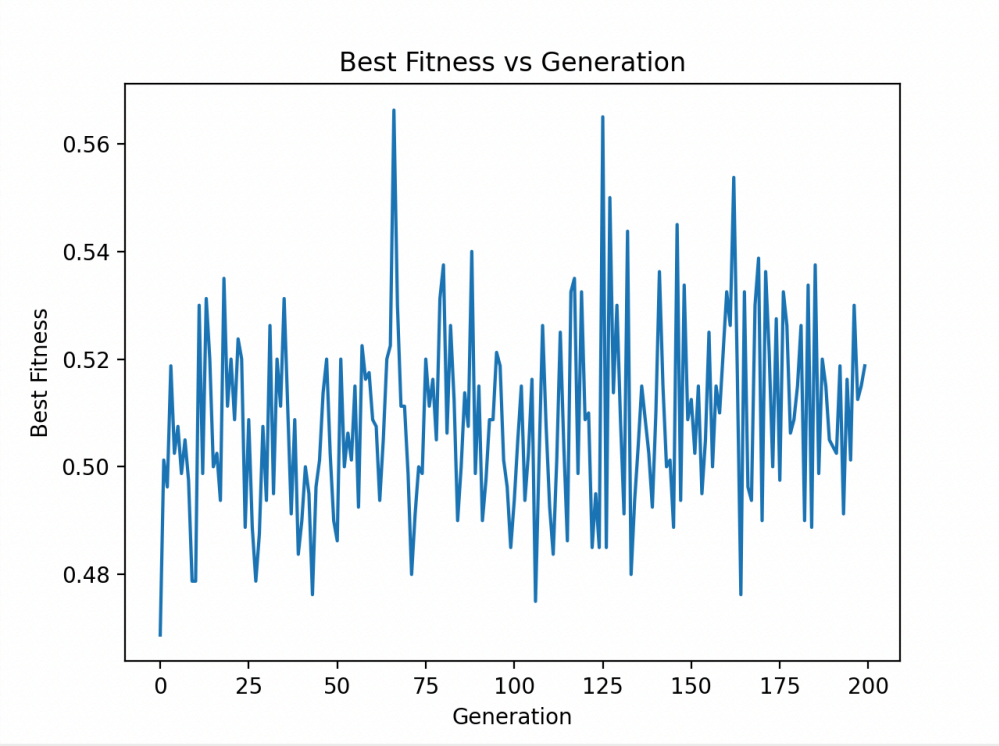

Figure 2: Sự thay đổi của độ thích nghi tốt nhất qua các thế hệ

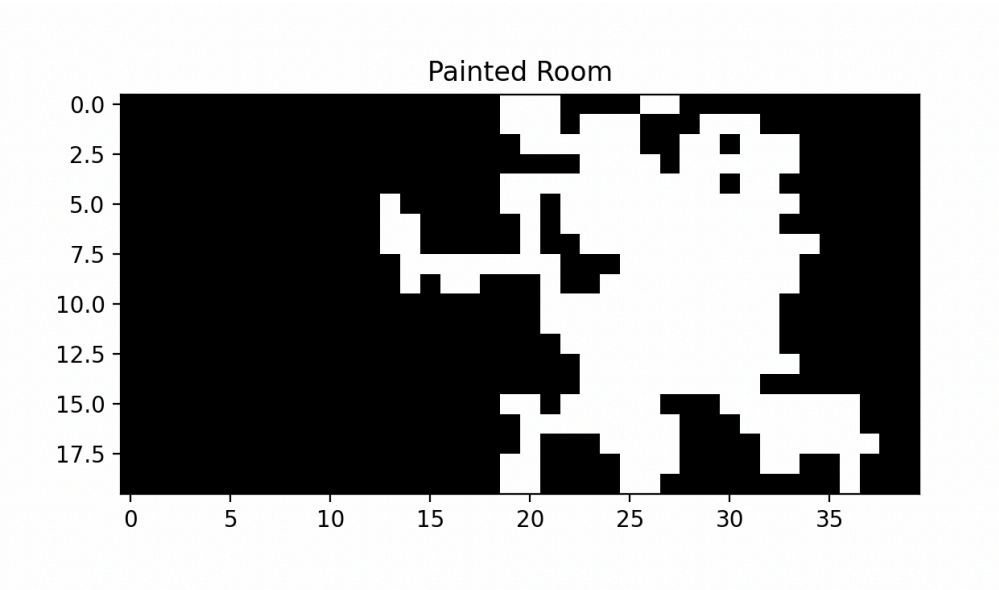

Figure 3: Kết quả sơn phòng của robot dựa trên nhiễm sắc thể tốt nhất

4. Thực hành

Ta đã xây dựng một chương trình giúp robot sơn phòng bằng giải thuật di truyền. Căn phòng cần sơn ở điều kiện lý tưởng là không có đồ vật trong phòng, cũng như ta không cần quan tâm đến số bước di chuyển của robot để sơn phòng, và toàn bộ căn phòng được sơn cùng một màu. Tuy nhiên, thực tế lại không như vậy. Hãy sửa đổi code mẫu theo các yêu cầu sau:

Q1: Thêm các vật cản (đồ đạc) vào phòng, chẳng hạn như bàn, ghế, tủ; sau đó thực hiện sơn phòng trong trường hợp phòng có đồ đạc.

Q2: Tối ưu hóa số bước di chuyển của robot để giảm thiểu thời gian sơn phòng. Điều này có thể được thực hiện bằng cách thêm một yếu tố phạt vào hàm đánh giá độ thích nghi nếu robot di chuyển quá nhiều bước.

Q3: Theo yêu cầu thiết kế một căn phòng bắt mắt, robot cần sơn một căn phòng với hai màu xen kẽ là xanh lam và vàng.