Tech Challenge <02>

Curso

Pós graduação em IA para Devs

Grupo 2

- Julio Cesar Scheidt Santos (julio.scheidt96@gmail.com)
- Julio Cesario de Paiva Leão (julio0023@live.com)
- Luis Gustavo Bueno Colombo (luisgustavobuenocolombo@gmail.com)
- Paulo Ubirajara de Mattos Neto (pauloubirajaraneto@hotmail.com)

URLs importantes

- Vídeo apresentando a aplicação
- Repositório do projeto no GitHub

Começando

Clone o projeto para sua máquina:

```
git clone https://github.com/luisgustavobueno/Tech-challenge-II.git
cd Tech-challenge-II
```

Instale as depedências:

> Recomendamos utilizar um ambiente virtual (venv) para instalar as dependências do projeto.

```
pip install -r requirements.txt
```

E execute a aplicação:

```
python main.py
```

Desafio

O desafio consiste em projetar, implementar e testar um sistema que utilize Algoritmos Genéticos para otimizar uma função ou resolver um problema complexo de otimização. Você pode escolher problemas como otimização de rotas, alocação de recursos e design de redes neurais.

Algoritmo Genético para Rotas de Drones

Introdução

Este projeto implementa um algoritmo genético para encontrar a rota mais eficiente para um drone que precisa entregar pacotes em vários pontos de um grid. O algoritmo utiliza operações como seleção, crossover e mutação para otimizar a rota do drone, minimizando a distância percorrida entre os pontos de entrega. A simulação gráfica é feita usando a biblioteca Pygame.

Objetivo

O objetivo deste algoritmo é minimizar a distância total percorrida pelo drone ao visitar todos os pontos de entrega em um grid com ou sem obstáculos. Ele usa um algoritmo genético que busca a solução mais eficiente através de várias gerações, aplicando operadores genéticos como seleção, crossover e mutação.

Preparação do Ambiente

Dependências

Antes de executar o projeto, certifique-se de ter as seguintes dependências instaladas:

- Python 3.x
- Bibliotecas necessárias:
  - `pygame`
  - `numpy`
  - `matplotlib`

Você pode instalar as dependências usando o seguinte comando:

```
pip install -r requirements.txt
```

Execução da Aplicação

Para executar a aplicação, basta rodar o arquivo Python principal:

```
python main.py
```

Durante a execução, você verá uma interface gráfica que mostrará a movimentação do drone no grid.

Explicação das Funções

1. Definição de Constantes

No início do código, são definidas constantes e parâmetros do algoritmo:

```
POP_SIZE = 100
MUTATION_RATE = 0.8
NUM_GENERATIONS = 5
ELITISM_COUNT = 5
CELL_SIZE = 40
GRID_SIZE = 16
SLEEP = 0.01
```

- POP_SIZE: Tamanho da população inicial de rotas.
- MUTATION_RATE: Taxa de mutação aplicada às rotas.
- NUM_GENERATIONS: Número de gerações para o algoritmo genético.
- ELITISM_COUNT: Número de melhores indivíduos mantidos em cada geração.
- CELL_SIZE: Tamanho de cada célula no grid.
- GRID_SIZE: Tamanho da grade (16x16).
- SLEEP: Intervalo de tempo entre os movimentos do drone.

2. Definição do `BASE_GRID`

O `BASE_GRID` é uma matriz que representa o grid no qual o drone se move. Cada célula da matriz tem um valor específico que pode representar um obstáculo, uma célula livre, a posição do drone, ou a posição dos destinos de entrega.

Exemplo de Definição do Grid:

```
# Definição de constantes que representam diferentes elementos do grid
EMPTY = 0
OBSTACLE = 1
DRONE_POSITION = 2
DESTINATION = 3

# Definição do BASE_GRID
BASE_GRID = [
    [2, 0, 0, 1, 2, 0, 1, 0, 2, 0, 0, 1, 2, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 0, 2],
    [0, 2, 1, 0, 0, 2, 0, 0, 1, 2, 0, 0, 0, 2, 0, 1],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [2, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1],
    [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 3, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 2, 0, 1, 0, 2, 0, 1, 0, 2, 0, 0, 1, 2, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
    [2, 0, 1, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 2, 0, 1, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
]
```

Explicação:

- **EMPTY (0)**: Representa uma célula livre, onde o drone pode se mover.
- **OBSTACLE (1)**: Representa uma célula com um obstáculo, que o drone não pode atravessar.
- **DRONE_POSITION (2)**: Representa a posição inicial do drone. Esta célula será atualizada durante a simulação à medida que o drone se move.
- **DESTINATION (3)**: Representa um destino ou ponto de entrega. Este valor será atualizado durante a simulação.

3. Função `calculate_distance`

Calcula a distância euclidiana entre dois pontos no grid:

```
def calculate_distance(point1, point2):
    return math.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)
```

4. Função `calculate_fitness`

Calcula o fitness de uma rota, que corresponde à distância total percorrida pelo drone:

```
def calculate_fitness(route):
    total_distance = 0
    current_position = start_position
    for point in route:
        total_distance += calculate_distance(current_position, point)
        current_position = point
    return total_distance
```

O fitness é a métrica usada para selecionar as rotas mais eficientes. Quanto menor a distância total, melhor o fitness.

5. Função `selection`

Implementa a seleção por torneio, escolhendo os pais para o crossover:

```
def selection(population, fitnesses):
    selected = random.sample(list(zip(population, fitnesses)), k=2)
    return min(selected, key=lambda x: x[1])[0]
```

Esta função seleciona dois indivíduos da população e escolhe aquele com menor fitness para continuar no processo.

6. Função `crossover`

Realiza o crossover ordenado (Ordered Crossover - OX) entre dois pais para gerar um novo filho:

```
def crossover(parent1, parent2):
    cut_point = random.randint(1, len(parent1) - 1)
    child = parent1[:cut_point]
    for point in parent2:
        if point not in child:
            child.append(point)
    return child
```

O crossover combina genes (pontos de entrega) de dois pais para criar uma nova rota.

7. Função `mutate`

Realiza a mutação, trocando dois pontos de entrega aleatoriamente:

```
def mutate(route):
    if random.random() < MUTATION_RATE:
        i, j = random.sample(range(len(route)), 2)
        route[i], route[j] = route[j], route[i]
    return route
```

8. Função `generate_population`

Gera uma população inicial de rotas aleatórias:

```
def generate_population():
    population = []
    for _ in range(POP_SIZE):
        route = random.sample(delivery_points, len(delivery_points))
        if route not in population:
          population.append(route)
    return population
```

Esta função cria várias rotas aleatórias como ponto de partida para o algoritmo genético.

9. Função `draw_grid`

Desenha o grid com a posição atual do drone e obstáculos:

```
def draw_grid(grid, target):
    for y in range(GRID_SIZE):
        for x in range(GRID_SIZE):
            rect = pygame.Rect(x * CELL_SIZE, y * CELL_SIZE, CELL_SIZE, CELL_SIZE)
            if grid[y][x] == OBSTACLE:
                pygame.draw.rect(screen, BLACK, rect)
            elif grid[y][x] == DRONE_POSITION:
                pygame.draw.rect(screen, BLUE, rect)
            elif [y, x] == target:
                pygame.draw.rect(screen, ORANGE, rect)
            elif grid[y][x] == DESTINATION:
                pygame.draw.rect(screen, GREEN, rect)
            else:
                pygame.draw.rect(screen, WHITE, rect)
            pygame.draw.rect(screen, RED, rect, 1)
    pygame.display.flip()
```

Essa função é responsável por renderizar a simulação gráfica no Pygame.

10. Função `move_drone`

Move o drone célula por célula até o destino, considerando as rotas definidas:

```
def move_drone(route):
    current_position = start_position.copy()
    grid = [row[:] for row in BASE_GRID]
    positions = []
    previous_position = None
    for point in route:
        while current_position != point:
            # Lógica de movimentação do drone
            # Atualiza e desenha a nova posição do drone no grid
            draw_grid(grid, point)
            time.sleep(SLEEP)
```

O drone é movido conforme a rota definida pela melhor solução encontrada pelo algoritmo genético.

11. Função Principal `main`

Executa o algoritmo genético, gerando rotas, selecionando os melhores indivíduos e movendo o drone:

```
def main():
    global delivery_points, start_position

    # Inicializa os pontos de entrega e a posição inicial do drone
    delivery_points = ...
    start_position = ...

    # Gera a população inicial
    population = generate_population()

    for generation in range(NUM_GENERATIONS):
        # Avalia a população e executa o algoritmo genético
        fitnesses = [calculate_fitness(route) for route in population]
        new_population = ...

        # Simula o movimento do drone com a melhor rota
        best_route = population[0]
        move_drone(best_route)

    pygame.quit()
```

Estrutura do Algoritmo Genético

1. Gerar população inicial de rotas.
2. Avaliar a população calculando o fitness (distância total).
3. Selecionar os melhores indivíduos (elitismo).
4. Aplicar crossover e mutação para criar novas rotas.
5. Repetir o processo por várias gerações até encontrar a melhor rota.
6. Simular o movimento do drone usando a melhor rota encontrada.

Conclusão

Este projeto demonstra o uso de um algoritmo genético para otimizar rotas de drones em um ambiente com obstáculos. A cada geração, o algoritmo busca rotas mais eficientes, ajustando as posições de entrega para minimizar a distância total percorrida. A visualização gráfica da simulação é feita com Pygame, proporcionando uma experiência interativa.