西安电子科技大学

人工智能概论大作业

题　　目：遗传算法在旅行商问题中的应用与实现

姓　　名：X X X

学　　号：2300920XXXX

专　　业：

摘要

本小论文实现了遗传算法（Genetic Algorithm, GA）来解决旅行商问题（Traveling Salesman Problem, TSP）。旅行商问题是一个经典的组合优化问题，旨在找到一条经过每个城市恰好一次并最终返回起点的最短路径。本文提出的遗传算法通过模拟自然选择和遗传机制，包括选择、交叉和变异操作，来不断进化种群中的解，从而找到近似最优的路径。实验结果表明，该算法能够有效地缩短路径长度，提高解的质量。在设定的10个城市问题中，算法在多次迭代后找到了一个较短的路径，证明了其可行性和有效性。

关键词： GA算法、遗传、路径最优

一．绪论

旅行商问题（TSP）是组合优化领域的一个经典问题，其目标是在给定一组城市和每对城市之间的距离后，找到一条经过每个城市恰好一次并最终返回起点的最短路径。遗传算法（GA）是一种模拟自然选择和遗传机制的优化算法，通过选择、交叉和变异等操作来不断进化种群中的解，从而找到近似最优的解。

二．算法介绍

1. 种群初始化

在遗传算法的初始阶段，需要生成一个包含多个个体的初始种群。每个个体代表一个可能的解，即一个城市序列。通常，初始种群是随机生成的。

实现步骤：

- 设定种群大小（POP_SIZE）。
- 对于每个个体，随机生成一个城市序列

2. 适应度函数

适应度函数用于评估每个个体的优劣程度。在旅行商问题中，适应度通常定义为个体所代表路径的总长度。

实现步骤：

- 根据城市间的距离矩阵，计算每个个体的路径长度。
- 返回路径长度作为适应度值（注意：有时为了简化计算，会取路径长度的倒数或负值作为适应度，以便进行最大化操作）。

3. 选择操作

选择操作用于从当前种群中选出若干个体作为父代，用于生成下一代。常用的选择方法有轮盘赌选择、锦标赛选择等。

实现步骤（以轮盘赌选择为例）：

- 计算每个个体的适应度值。
- 根据适应度值计算每个个体的选择概率。
- 根据选择概率随机选择个体作为父代。

4. 交叉操作

交叉操作用于将两个父代个体的部分基因进行交换，以生成新的子代个体。常用的交叉方法有单点交叉、双点交叉等。

实现步骤（以单点交叉为例）：

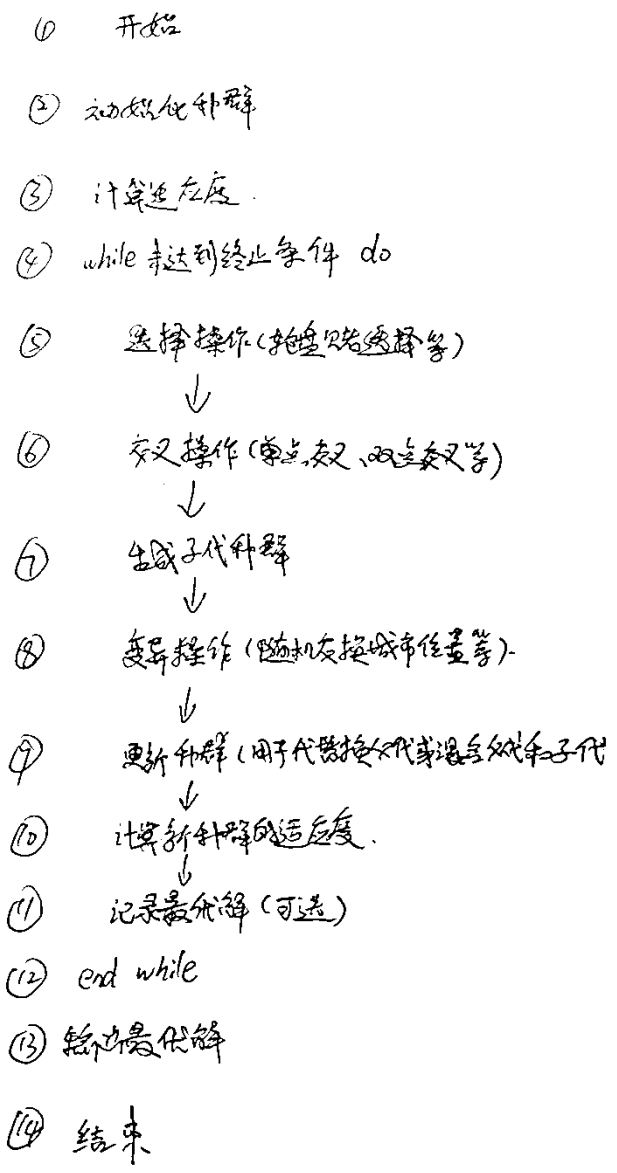
- 随机选择一个交叉点。
- 交换两个父代个体在交叉点之后的基因部分。
- 生成两个新的子代个体。

5. 变异操作

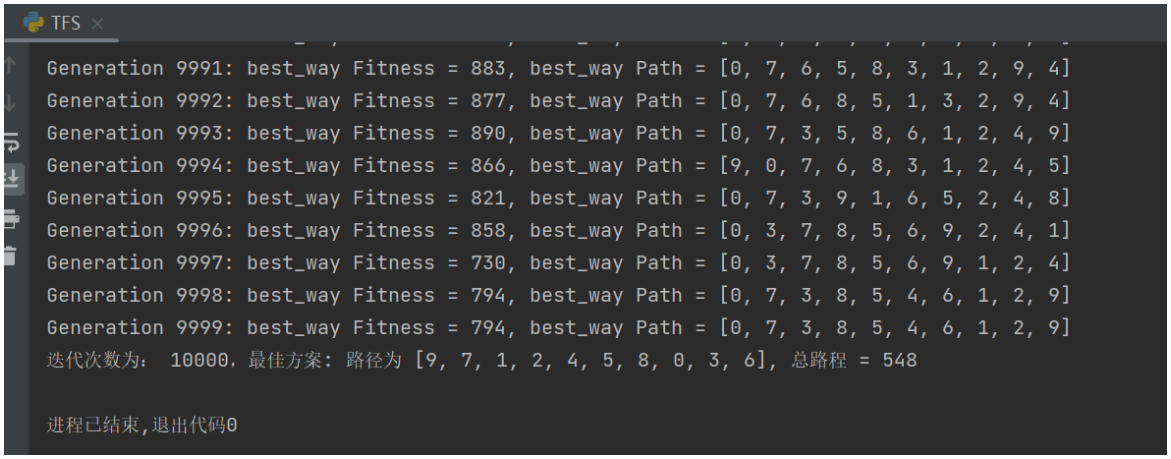
变异操作用于随机改变个体中的某些基因，以引入新的遗传信息。在旅行商问题中，常用的变异方法是随机交换两个城市的位置。

实现步骤：

- 随机选择两个城市。
- 交换这两个城市在个体中的位置。



三．实验过程



为确保实验结果的较为准确性，我在设置迭代次数时选择了10,000次，以确保能得到一个较好的结果。

结果：迭代次数为： 10000，最佳方案: 路径为 [9, 7, 1, 2, 4, 5, 8, 0, 3, 6], 总路程 = 548

四．实验结论

本文实现了遗传算法来解决旅行商问题，并通过实验验证了其可行性和有效性。实验结果表明，该算法能够有效地缩短路径长度，提高解的质量。然而，算法也存在一些不足之处，如计算复杂度较高、收敛速度较慢等。在未来的研究中，可以进一步探索如何优化算法参数、改进选择、交叉和变异策略以及引入其他优化技术来提高算法的性能。

五. 参考文献

[1] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

[2] 刘勇, 康立山, 陈毓屏. (1996). 非数值并行算法——遗传算法（第二册）. 科学出版社.

[3] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). The traveling salesman problem: A guided tour of combinatorial optimization. John Wiley & Sons.

附录：代码

（附程序代码，关键语句进行必要注释。如果代码程序使用了一些网络上的资源，请务必在附录最后注明一下，哪些部分属于借鉴内容，哪些部分属于自己原创的工作。

注意：代码可以附在论文后，也可以单独作为一个文件上传，但需标注文件名为题目名。）

import numpy as np

```
import random

# 城市间的距离矩阵（对称矩阵，对角线为0）
# 示例中包含了10个城市之间的距离
city_distances = np.array([
    [0, 93, 49, 51, 46, 58, 93, 68, 69, 127],
    [93, 0, 45, 52, 111, 141, 67, 25, 154, 90],
    [49, 45, 0, 28, 69, 97, 72, 21, 114, 105],
    [51, 52, 28, 0, 86, 108, 46, 29, 102, 80],
    [46, 111, 69, 86, 0, 33, 132, 90, 100, 167],
    [58, 141, 97, 108, 33, 0, 152, 118, 86, 186],
    [93, 67, 72, 46, 132, 152, 0, 60, 124, 34],
    [68, 25, 21, 29, 90, 118, 60, 0, 128, 90],
    [69, 154, 114, 102, 100, 86, 124, 128, 0, 151],
    [127, 90, 105, 80, 167, 186, 34, 90, 151, 0]
])

# 遗传算法参数
POP_SIZE = 100  # 种群大小，即有多少条不同的路径在同时进化
GENES = list(range(10))  # 基因（城市编号），表示10个城市
generation = 10000  # 迭代次数，即算法运行多少代
MUTATION_RATE = 0.01  # 变异率，控制每个基因变异的可能性
cross_rate = 0.8  # 交叉率，控制两个父代交叉产生子代的可能性


# 初始化种群
# 创建初始种群，每个个体都是一个随机的城市排列
def create_population(pop_size, genes):
    return [random.sample(genes, len(genes)) for _ in range(pop_size)]


population = create_population(POP_SIZE, GENES)


# 适应度函数
# 计算给定路径的总距离
def fitness(every):
```

```
    total_distance = sum(city_distances[every[i]][every[i + 1]] for i in
range(len(every) - 1))
    total_distance += city_distances[every[-1]][every[0]]  # 回到起点
    return total_distance


# 选择操作（轮盘赌选择）
# 根据适应度概率选择个体，适应度越低（距离越短）被选中的概率越高
def select(population, fitnesses):
    total_fitness = sum(fitnesses)
    probabilities = [f / total_fitness for f in fitnesses]  # 计算每个个体的选择概
率
    selected_index = np.random.choice(range(len(population)), p=probabilities)
# 根据概率选择个体
    return population[selected_index]


# 交叉操作（单点交叉）
# 两个父代个体交换部分基因，产生两个新的子代个体
def crossover(parent1, parent2):
    if random.random() < cross_rate:  # 根据交叉率决定是否进行交叉
        point = random.randint(1, len(parent1) - 2)  # 随机选择一个交叉点
        # 生成两个新的子代，确保没有重复的基因
        child1 = parent1[:point] + [gene for gene in parent2 if gene not in
parent1[:point]]
        child2 = parent2[:point] + [gene for gene in parent1 if gene not in
parent2[:point]]
        return child1, child2
    else:
        return parent1, parent2  # 不进行交叉，直接返回父代


# 变异操作（随机交换两个基因）
# 随机选择两个基因并交换它们的位置
def mutate(every):
    if random.random() < MUTATION_RATE:  # 根据变异率决定是否进行变
异
```

```
        i, j = random.sample(range(len(every)), 2)  # 随机选择两个基因的位置
        every[i], every[j] = every[j], every[i]  # 交换两个基因
    return every


# 遗传算法主循环
best_way = None  # 记录最优解
best_match = float('inf')  # 记录最优解的适应度（初始化为正无穷大）

for generation in range(generation):  # 迭代N代
    fitnesses = [fitness(ind) for ind in population]  # 计算当前种群中每个个体的适应度
    new_population = []  # 存储新种群的列表

    # 进行选择、交叉和变异操作，生成新的种群
    for _ in range(POP_SIZE // 2):  # 每次生成两个新的个体
        parent1 = select(population, fitnesses)  # 选择一个父代个体
        parent2 = select(population, fitnesses)  # 选择另一个父代个体
        child1, child2 = crossover(parent1, parent2)  # 交叉产生两个子代个体
        new_population.append(mutate(child1))  # 对第一个子代进行变异并加入新种群
        new_population.append(mutate(child2))  # 对第二个子代进行变异并加入新种群

    population = new_population  # 更新种群

    # 记录当前代最优解
    current_best_way_every = min(population, key=fitness)  # 找到当前种群中适应度最低的个体
    current_best_match = fitness(current_best_way_every)  # 计算其适应度

    # 如果当前最优解的适应度比历史最优解的适应度更低，则更新最优解
    if current_best_match < best_match:
        best_match = current_best_match
        best_way = current_best_way_every

    # 输出当前代最优解和对应适应度（可用于观察算法进化过程）
```

```
    print(f"Generation {generation}: best_way Fitness =
{current_best_match}, best_way Path = {current_best_way_every}")

    # 输出最终结果
    print(f"迭代次数为： {generation+1}，最佳方案: 路径为 {best_way}, 总路程
= {best_match}")
# 注明：以上代码部分借鉴了网络上的遗传算法和旅行商问题的相关资料，但标黄部分具体实现和参数调整属于原创工作。
```