厦門大學


# 软件学院

## 《人工智能导论》实验报告

题　　目 遗传算法

姓　　名 陈澄

学　　号 32420212202930

班　　级 软工三班

实验时间 2024/03/20

**2024　年　03　月　20　日**

# 1 实验目的

遗传算法(GeneticAlgorthm,GA）起源于对生物系统所进行的计算机模拟研究。其本质是一种高效、并行、全局搜索的方法，能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最佳解。本实验通过解决旅行商问题，帮助学生更好的熟悉和掌握遗传算法。

# 2 实验内容

**利用遗传算法解决旅行商问题**

旅行商问题即 TSP 问题（Traveling Salesman Problem）又译为旅行推销员问题、货郎担问题，是数学领域中著名问题之一。假设有一个旅行商人要拜访 n 个城市，每两座城市之间的距离是不同的，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

# 3 实验步骤

## 3.1 相关数据结构以及说明

城市坐标采用平面直角坐标系的 xy 坐标，便于后续总路径计算

```
// 定义城市坐标结构体
struct City {
    int x, y;
};
```

城市坐标集合即为上述坐标的一维数组集合

路线的表示方法为一维数组，第 i 个数据元素存储第 i 次旅行的目标城市（因为最后一次旅行一定是回到起点，因此不必表示）

如：01234 表示：0->1->2->3->4->0

种群的为路线的集合，因此是二维数组

```
// 最优路线
vector<int> best = { 0,1,2,3,4 };

// 定义城市坐标
vector<City> cities = { {0, 2}, {1, 9}, {3, 0}, {2, 4}, {4, 7} };

// 初始化种群
int populationSize = 60;
vector<vector<int>> population = initializePopulation(populationSize, cities.size());

// 迭代次数
int numGenerations = 200;
```

## 3.2 选择测试用的目标函数

对于旅行商问题，测试目标函数设计为路径的总长度，因为我们的目标是找到一条路径，使得旅行商访问所有城市后回到起点，并且路径长度最短。

```
// 计算两个城市之间的距离
double distance(City city1, City city2) {
    return sqrt(pow(city1.x - city2.x, 2) + pow(city1.y - city2.y, 2));
}

// 计算路径的总距离
double totalDistance(const vector<int>& path, const vector<City>& cities) {
    double total = 0.0;
    for (size_t i = 0; i < path.size() - 1; ++i) {
        total += distance(cities[path[i]], cities[path[i + 1]]);
    }
    total += distance(cities[path.back()], cities[path.front()]);
    return total;
}
```

## 3.3 设计有效的遗传算子

各个遗传算子及其说明如下

## 3.4 初始化函数

将种群中每个个体都初始化为依次为 0-numCities 的顺序数组，再通过 random_shuffle 打乱顺序，由此生成初始的旅行路径集合。随机排列的目的是为了增加种群的多样性。

```cpp
// 初始化种群
vector<vector<int>> initializePopulation(int populationSize, int numCities) {
    vector<vector<int>> population(populationSize, vector<int>(numCities));
    for (int i = 0; i < populationSize; ++i) {
        for (int j = 0; j < numCities; ++j) {
            population[i][j] = j;
        }
        random_shuffle(population[i].begin(), population[i].end());
    }
    return population;
}
```

### 3.5 适应度函数

适应度函数设计为总路径长度的倒数，因此总路径越长，适应度越差，越容易被淘汰。

```cpp
// 计算适应度
vector<double> fitness(const vector<vector<int>>& population, const vector<City>& cities) {
    vector<double> fitnessValues(population.size());
    for (size_t i = 0; i < population.size(); ++i) {
        fitnessValues[i] = 1.0 / totalDistance(population[i], cities);
    }
    return fitnessValues;
}
```

### 3.6 复制函数

复制函数即选择种群中的亲代生成子代的过程。

选择亲代的函数如下：计算平均适应度，后随机选择适应度高于平均值的亲代，并返回其 index。确保选择出的 parent 都是适应值高的优秀个体。

选择两个亲代，通过交叉函数交叉即可生成子代，实现复制。

```cpp
// 选择函数
int selectParent(const vector<double>& fitnessValues) {
    double sumFitness = 0.0;
    for (double fitness : fitnessValues) {
        sumFitness += fitness;
    }
    double avgFitness = sumFitness / fitnessValues.size();
    while(true) {
        int i = rand() % fitnessValues.size();
        if (fitnessValues[i] ≥ avgFitness)return i;
    }
}
```

### 3.7 交换函数

选择交叉点 startPos 和 endPos，交叉两点中间的交叉段。

```
// 交叉函数
vector<int> crossover(const vector<int>& parent1, const vector<int>& parent2) {
    int startPos = rand() % parent1.size();
    int endPos = rand() % parent1.size();
    if (startPos > endPos) {
        swap(startPos, endPos);
    }
    vector<int> child(parent1.begin() + startPos, parent1.begin() + endPos);
    for (int city : parent2) {
        if (find(child.begin(), child.end(), city) == child.end()) {
            child.push_back(city);
        }
    }
    return child;
}
```

### 3.8 变异函数

选择两个基因点，将两点的基因交换。

```
// 变异函数
void mutate(vector<int>& individual) {
    int pos1 = rand() % individual.size();
    int pos2 = rand() % individual.size();
    swap(individual[pos1], individual[pos2]);
}
```

### 3.9 主函数

初始化种群；

选择一个迭代次数；

每次迭代都使用选择函数，选择两个亲代个体，后通过交叉函数生成子代个体，重复以上过程直到子代数目达到种群数量；

随机选择 5%的个体变异，使用变异函数变异；

新种群替代原种群，选择最佳个体；

```cpp
int main() {
    // 最优路线
    vector<int> best = { 0,1,2,3,4 };

    // 定义城市坐标
    vector<City> cities = { {0, 2}, {1, 9}, {3, 0}, {2, 4}, {4, 7} };

    // 初始化种群
    int populationSize = 600;
    vector<vector<int>> population = initializePopulation(populationSize, cities.size());

    // 迭代次数
    int numGenerations = 100;

    for (int generation = 0; generation < numGenerations; ++generation) {
        // 计算适应度
        vector<double> fitnessValues = fitness(population, cities);

        // 选择新一代种群
        vector<vector<int>> newPopulation;
        for (int i = 0; i < populationSize; ++i) {
            int parent1 = selectParent(fitnessValues);
            int parent2 = selectParent(fitnessValues);
            newPopulation.push_back(crossover(population[parent1], population[parent2]));
        }

        // 变异
        for (vector<int>& individual : newPopulation) {
            if (static_cast<double>(rand()) / RAND_MAX < 0.05) {
                mutate(individual);
            }
        }

        // 更新种群
        population = newPopulation;

        // 选择最佳个体
        double bestFitness = 0.0;
        int bestIndex = 0;
        for (size_t i = 0; i < population.size(); ++i) {
            double currentFitness = 1.0 / totalDistance(population[i], cities);
            if (currentFitness > bestFitness) {
                bestFitness = currentFitness;
                bestIndex = i;
            }
        }
        double oldFitness = 1.0/ totalDistance(best, cities);
        if (oldFitness < bestFitness)best = population[bestIndex];
    }

    // 输出结果
    cout << "最佳路径: ";
    for (int city : best) {
        cout << city << " ";
    }
    cout << endl;
    cout << "路径长度: ";
    cout << totalDistance(best, cities) << endl;

    return 0;
}
```

## 4 实验遇到的问题及其解决方法

无

## 5 我的体会

在本次实验中，我们通过利用遗传算法解决旅行商问题，深入学习了遗传算法的基本原理和应用。通过这个过程，我深刻体会到了遗传算法作为一种通用的优化方法，具有较强的适应性和灵活性，能够应用于各种复杂的优化问题中。掌握了遗传算法的基本原理和应用技巧，对于解决实际问题具有重要的指导意义，也为我今后的学习和研究打下了坚实的基础。