厦門大學

软件学院

《人工智能导论》实验报告

题　　目　粒子群算法

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

粒子群优化（PSO）算法是一种群体智能算法，由 Kennedy 和 Eberhart 于 1995 年用核算机模仿鸟群寻食这一简略的社会行动时，遭到启示，简化之后而提出的。本实验通过解决旅行商问题，帮助学生更好的熟悉和掌握粒子群优化算法。

2 实验内容

利用粒子群优化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

1. 选择合适的粒子数、粒子长度、粒子的范围、Vmax、学习因子和终止条件。

选择的参数如下：

粒子数选取为 50，过大会增加计算负担，太小又不能充分搜索，因此设置为 50；

其中粒子长度取决于城市数目；

粒子的范围为城市的编号的范围（0 至 n-1）；

Vmax 设置为 0.2；

学习因子分为社会因子以及认知因子两部分，都设置为 2.0；

TSP 问题不设置终止条件，找到最小路径的解即可；

```
//最大迭代次数
const int MAX_ITERATIONS = 200;
//粒子群粒子数目
const int POPULATION_SIZE = 50;
//惯性因子
const double INERTIA_WEIGHT = 0.5;
//认知因子
const double COGNITIVE_WEIGHT = 2.0;
//社会因子
const double SOCIAL_WEIGHT = 2.0;
//最大速度
const double V_MAX = 0.2;
//终止阈值
const double TERMINATION_THRESHOLD = 1e-6;
```

2. 编程解决旅行商问题

(1) 粒子结构体如下，包含：

position 城市的访问顺序

pbest 粒子跟踪的极值之一

fitness 适应度

velocity 速度

```
//粒子结构体
struct Particle {
    vector<int> position;
    vector<int> pbest;
    double fitness;
    vector<double> velocity;
};
```

(2) 城市定义

用临接矩阵存储城市之间的距离，城市 i 与城市 j 之间的距离为 distances[i][j]

```
//城市位置定义
vector<vector<double>> distances = {
    {0, 10, 15, 20},
    {10, 0, 35, 25},
    {15, 35, 0, 30},
    {20, 25, 30, 0}
};
```

(3) 初始化粒子群

初始路径设置为 0-(n-1)随机打乱顺序后得到的序列

pbest 初始化为初始路径

Fitness 适应度初始化为极大值

velocity 速度初始化为 0

```cpp
//初始化粒子群
vector<Particle> initializeParticles(int populationSize, int cityCount) {
    vector<Particle> particles;
    for (int i = 0; i < populationSize; ++i) {
        Particle particle;
        particle.position = generateRandomPermutation(cityCount);
        particle.pbest = particle.position;
        particle.fitness = numeric_limits<double>::max();
        particle.velocity.resize(cityCount, 0.0);
        particles.push_back(particle);
    }
    return particles;
}
```

(4) 粒子位置和速度更新

按照公式 3 编写即可

cognitiveComponent 即认知部分增量

socialComponent 即社会部分增量

$$v_i = \omega \times v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i)$$

```cpp
//更新粒子的位置和速度
void updateParticle(Particle& particle, const vector<int>& gbest, double inertiaWeight,
    double cognitiveWeight, double socialWeight, double vMax) {
    for (int i = 0; i < particle.position.size(); ++i) {
        double r1 = static_cast<double>(rand()) / RAND_MAX;
        double r2 = static_cast<double>(rand()) / RAND_MAX;
        double cognitiveComponent = cognitiveWeight * r1 * (particle.pbest[i] - particle.position[i]);
        double socialComponent = socialWeight * r2 * (gbest[i] - particle.position[i]);
        particle.velocity[i] = inertiaWeight * particle.velocity[i] + cognitiveComponent + socialComponent;
        if (particle.velocity[i] > vMax) {
            particle.velocity[i] = vMax;
        }
        else if (particle.velocity[i] < -vMax) {
            particle.velocity[i] = -vMax;
        }
        particle.position[i] += round(particle.velocity[i]);
    }
}
```

(5)PSO 算法

定义全局最优位置 gbest

每次迭代重新计算距离和适应度，并更新位置和速度

最后将 gbest 返回

```cpp
//PSO算法
vector<int> PSO(const vector<vector<double>>& distances, int maxIterations, int populationSize,
    double inertiaWeight, double cognitiveWeight, double socialWeight, double vMax) {
    int cityCount = distances.size();
    vector<Particle> particles = initializeParticles(populationSize, cityCount);
    vector<int> gbest = particles[0].position;
    double gbestFitness = calculatePathDistance(gbest, distances);

    for (int iter = 0; iter < maxIterations; ++iter) {
        for (int i = 0; i < populationSize; ++i) {
            double fitness = calculatePathDistance(particles[i].position, distances);
            if (fitness < particles[i].fitness) {
                particles[i].fitness = fitness;
                particles[i].pbest = particles[i].position;
                if (fitness < gbestFitness) {
                    gbest = particles[i].pbest;
                    gbestFitness = fitness;
                }
            }
        }
        for (int i = 0; i < populationSize; ++i) {
            updateParticle(particles[i], gbest, inertiaWeight, cognitiveWeight, socialWeight, vMax);
        }
    }

    return gbest;
}
```

3. 与实验二遗传算法对比，总结粒子群算法和遗传算法的优劣。

粒子群优点：

简单易实现，有良好的局部搜索能力，可以通过控制惯性权重和权重因子来平衡全局和局部搜索能力，有一定的全局搜索能力。

粒子群缺点：

容易陷入局部最优解，参数选择对算法性能影响较大。

遗传算法优点：

全局搜索能力强，能够适应不同的问题领域，并且不需要对问题的特性进行过多的假设，参数设置简单。

遗传算法缺点：

计算复杂度高，收敛速度慢，局部搜索能力较弱。

4 我的体会

通过本次实验，我了解了 PSO 算法的基本原理和实现方法，通过与上次实验的对比也了解了 PSO 算法与遗传算法各自的优劣。