南開大學

算法导论大作业

学院　计算机学院
专业　计算机科学与技术
南开大学　程伟卿
学号　2311865

1 问题描述

1.1 选题

快递路径优化问题。

1.2 选题解释

在日常生活中，快递员每天需要在多个地点间送货，希望用最短的路线完成任务。这可以抽象为一个“多点路径规划问题”，是旅行商问题（TSP）的一种简化应用。

需要设计一个确定性的串行算法，在输入为多个送货地址（以坐标形式给出）时，输出一条近似最短的巡回路线。

2 备注

- 论文中只给出伪代码，详细代码见附件或仓库。
- 仓库位置，点击跳转。

3 算法设计与优化

3.1 基础算法

采用最近邻贪心策略，用于快速构造一条初始可行路径。算法从一个起始点出发，每次选择距离当前点最近且尚未访问的点作为下一个访问目标，直到所有点都被访问一次，最后返回起点。

伪代码如下：

```
Input: 点集 P = {p1, p2, ..., pn}，其中每个点为 (x, y) 坐标
Output: 路径 path，表示依次访问点的顺序

1. 初始化：
    visited = 空集合
    path = 空列表
    current = 起始点 p1
    将 p1 加入 visited 和 path
```

```
2. while visited 中点数 < n:
    min_dist = ∞
    next_point = None
    for 每个点 p in P:
        if p 不在 visited 中:
            d = 计算 current 到 p 的距离
            if d < min_dist:
                min_dist = d
                next_point = p
    将 next_point 加入 visited 和 path
    current = next_point

3. 将起始点 p1 加入 path（形成回路）

4. 返回 path
```

该算法时间复杂度为 $O(n^2)$，不能保证最优解，存在改进空间。

3.2 优化算法一

回溯法求解 TSP（全排列 + 剪枝）的优化算法可以在 n 较小时确保求解得最优解，但时间复杂度来到了 $O(n!)$，伪代码如下：

```
Input: 点集 P = {p1, p2, ..., pn}
Output: 最短路径 shortest_path, 最短距离 min_dist

1. 初始化:
    min_dist = ∞
    shortest_path = 空列表
    path = [p1]          // 从 p1 出发
    visited = {p1}       // 已访问点集

2. 定义递归函数 dfs(current_point, path, visited, current_dist):
    if 所有点都访问完:
        total_dist = current_dist + 距离(current_point, 起点)
        if total_dist < min_dist:
            min_dist = total_dist
            shortest_path = path + [起点]
```

```
        return

    for 每个点 p in P:
        if p 不在 visited 中:
            next_dist = current_dist + 距离(current_point, p)
            if next_dist >= min_dist:  // 剪枝
                continue
            visited 加入 p
            dfs(p, path + [p], visited, next_dist)
            visited 删除 p

3. 调用 dfs(p1, [p1], {p1}, 0)

4. 返回 shortest_path, min_dist
```

3.3 优化算法二

我们思考进一步的优化算法，兼顾解的质量和算法的时间复杂度。考虑空间加速贪心 +2-opt 局部优化。

核心思想是：

1. 用 KD-Tree 结构快速查找最近的未访问点，将寻找最近点的时间复杂度降到 $O(logn)$。
2. 构造初始路径后，使用 2-opt 算法进行局部路径交换优化。
3. 整体时间复杂度降至 $O(nlogn + k * n)$，其中 k 是优化迭代次数。

伪代码如下：

```
Input: 点集 P = {p1, p2, ..., pn}
Output: 优化路径 path_opt

1. 构建 KD-Tree 索引结构，用于支持最近未访问点查询
2. 初始化 visited = 空集，path = 空列表
3. 从起点 p1 出发：
    while visited.size < n:
        next_point = 查询KD-Tree中当前点的最近未访问点
        加入 path 和 visited
```

```
10
11 4. 执行 2-opt 优化直到收敛或达到迭代上限
12
13 5. 返回 path_opt
```

算法步骤如下：

1. **KD-Tree 最近邻贪心构造路径**：

   利用 KD-Tree 建立所有送货点的空间索引结构，从起点出发，每次在 KD-Tree 中查找距离当前位置最近且尚未访问的点，逐步构造一条完整路径。该过程的查询效率约为 $O(\log n)$，总构造复杂度为 $O(n \log n)$。

2. **2-opt 局部路径优化**：

   在初始路径构造完成后，应用 2-opt 算法对路径进行局部交换优化。该算法通过反转路径中两个点之间的段，判断是否能减少总路径长度，从而不断逼近更优解。为了控制时间复杂度，通常设置最大迭代次数或收敛条件。

3. **时间复杂度分析**：

   该优化算法整体时间复杂度约为 $O(n \log n + kn)$，其中 k 为 2-opt 的迭代轮数（通常远小于 n）。相比基础贪心算法的 $O(n^2)$，本算法在大幅提升路径质量的同时显著降低了时间成本。

4 测试结果

总点数	路径总长度	运行时间（毫秒）
1000	29828.08	23
2000	41209.18	47
3000	50333.78	121
4000	57361.94	174
5000	63356.15	305

表 1: 普通算法性能测试结果

总点数	路径总长度	运行时间（秒）
10	2996.29	0.02
12	2704.45	0.18
15	3760.32	11.43
15	2828.56	8.25
16	3496.81	67.96

表 2: 优化算法一（暴力最优）性能测试结果

总点数	2-opt 最大迭代次数 k	路径总长度	运行时间（毫秒）
1000	50	28064.12	323
1000	100	27020.11	922
1000	200	25360.95	2467
1000	500	24483.06	5558
2000	200	38982.78	6630
2000	500	34426.60	26593

表 3: 优化算法二（2-opt）性能测试结果

5 结果分析

分析上述结果，普通算法耗时短，但实际时间复杂度高于优化算法二，优化算法二的时间消耗主要为迭代优化部分，且普通算法的路径不是最优解，优化算法二可以求出最优路径，但是耗时长，点数超过 20 后几乎无法运行，优化算法二可以在较短时间求出较优路径解。

6 总结反思

本次实验通过对旅行商问题的求解，分别实现了基础贪心算法、穷举最优算法与 2-opt 启发式优化算法，深入体会了不同算法在时间复杂度与解的质量上的权衡。基础算法虽然运行速度快，但解的质量较差；穷举算法能得到最优解，但计算量随点数指数增长，适用范围有限；2-opt 优化算法在保证较优解的同时显著提升了处理规模，展现出良好的实用性。实验不仅加深了对算法设计与复杂度分析的理解，也反映出合理选择算法在实际应用中

的重要性，值得在后续学习中继续探索更高效的近似优化方法。