一、算法设计题

1. 优化算法设计

（1）算法设计思想和伪代码

设计思想： 旅行商问题（TSP）属于NP难问题。为了在合理时间内找到高质量解，我采用了模拟退火算法。

1. 模拟退火（SA）：模仿物理退火过程。算法从一个随机路径和高温开始。
2. Metropolis 准则： 在每一步迭代中，对当前路径进行微扰产生新路径。如果新路径更短，则直接接受；如果更长，则以概率 $P = e^{-\Delta E/T}$ 接受（其中 ΔE 是距离增量，T 是当前温度）。这种机制允许算法跳出局部最优陷阱。
3. 2-opt 邻域操作： 通过随机选择两个节点并将中间的路径片段 反转 来产生新路径。这种方法能有效消除路径中的“交叉”现象，非常适合欧几里得TSP问题。
4. 降温策略： 温度 T 按冷却系数 α（如 0.995）逐渐降低，直到趋于稳定。

伪代码：

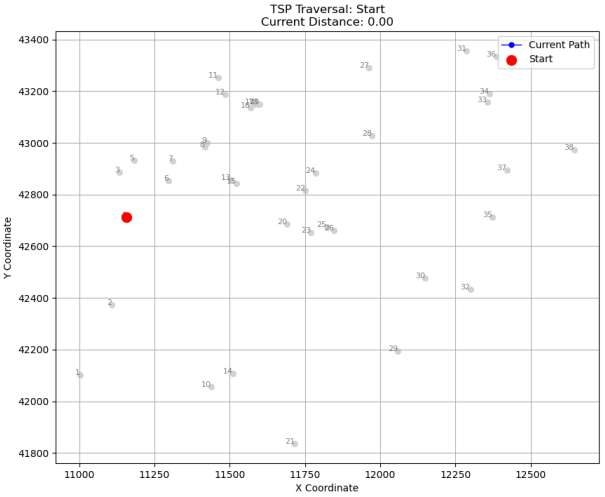
```
算法：模拟退火 (Simulated Annealing with 2-opt)

输入：城市列表 Cities
输出：最短路径 BestTour，最短距离 MinDistance

1. 初始化：
   CurrentTour = 随机排列(Cities)
   BestTour = CurrentTour
   T = 初始温度 (例如 10000)
   Min_T = 终止温度 (例如 0.001)
   Alpha = 冷却系数 (例如 0.995)

2. 循环 (当 T > Min_T 时):
     重复执行 (每个温度下的迭代次数):
         a. 产生邻域解 (2-opt):
            随机选择两个索引 i, j
            NewTour = 将 CurrentTour 中 i 到 j 的片段反转

         b. 计算距离变化:
            Delta = 距离(NewTour) - 距离(CurrentTour)

         c. 接受判断 (Metropolis准则):
            如果 Delta < 0 或者 Random(0,1) < exp(-Delta / T):
                CurrentTour = NewTour
                如果 距离(CurrentTour) < 距离(BestTour):
                    BestTour = CurrentTour

     d. 降温:
        T = T * Alpha

3. 返回 BestTour，距离(BestTour)
```

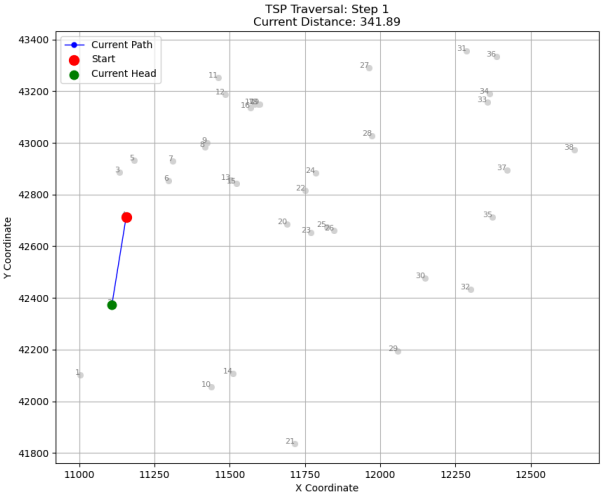
（2）最短距离及对应可视化遍历方案

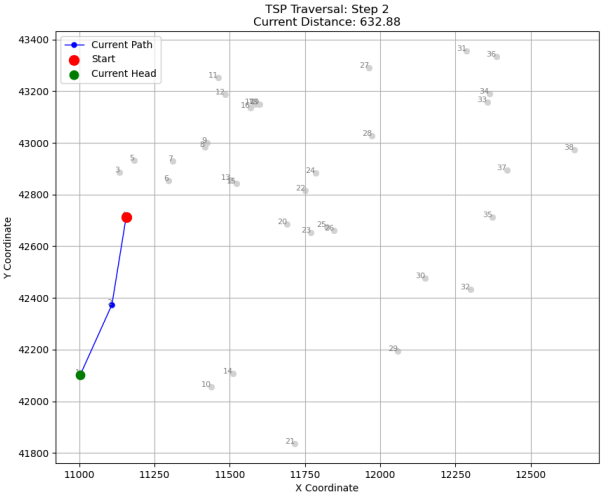
最短距离：6659.4315

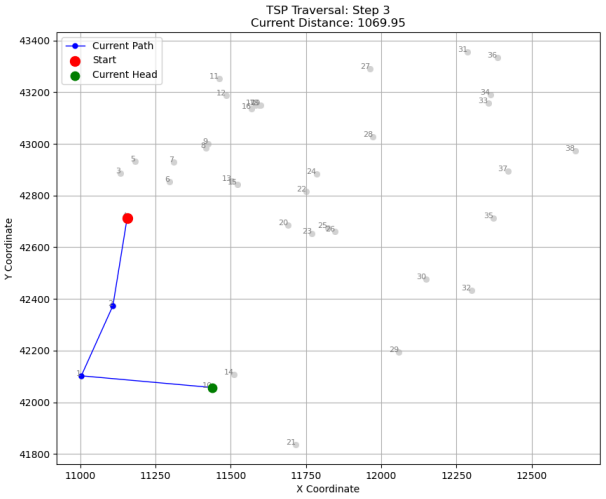
路径：[4, 2, 1, 10, 14, 21, 29, 30, 32, 35, 37, 38, 33, 34, 36, 31, 27, 28, 24, 22, 25, 26, 23, 20, 15, 13, 16, 17, 18, 19, 11, 12, 9, 8, 7, 6, 5, 3]

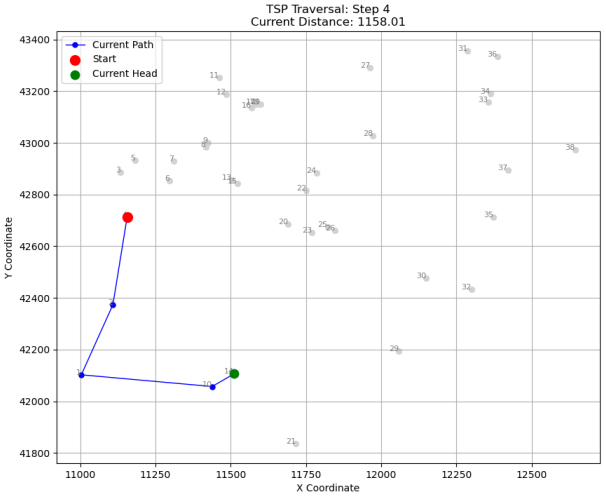
可视化遍历方案：

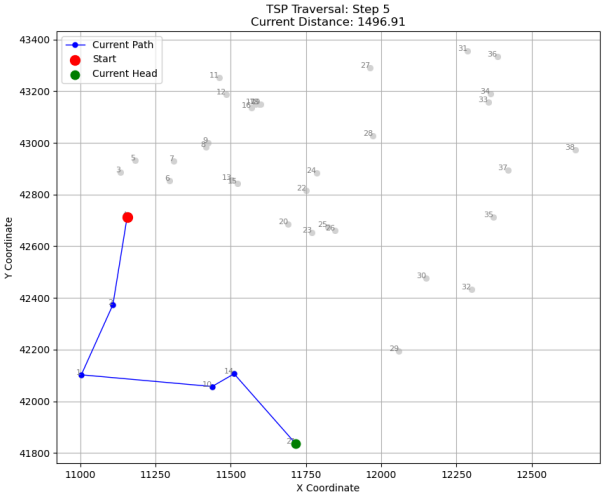


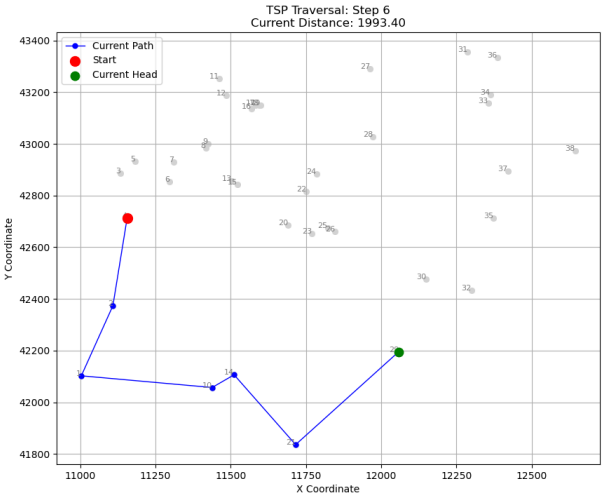


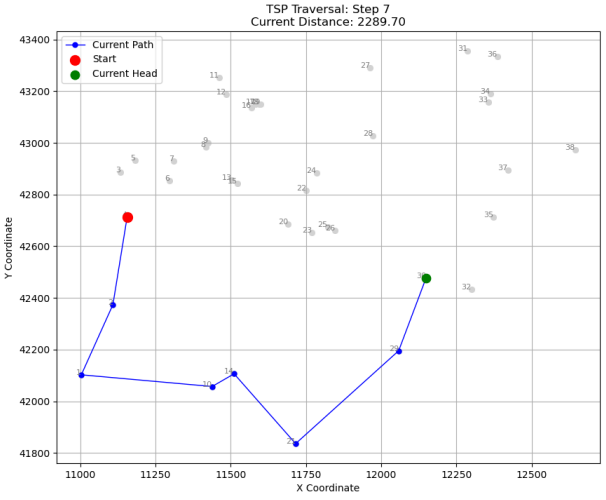


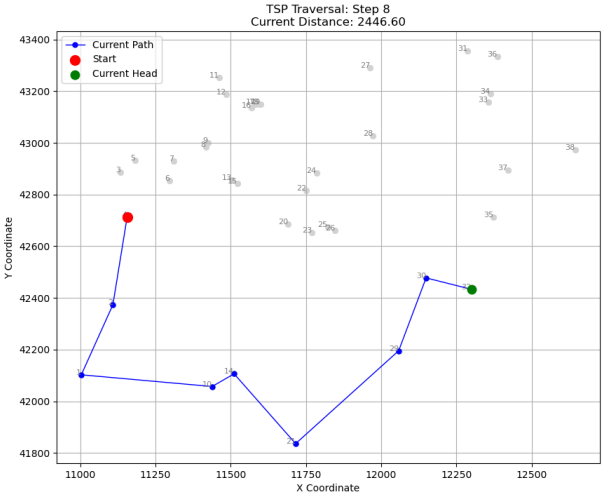

TSP Traversal: Step 3
Current Distance: 1069.95
Current Path
Start
Current Head
Y Coordinate
X Coordinate

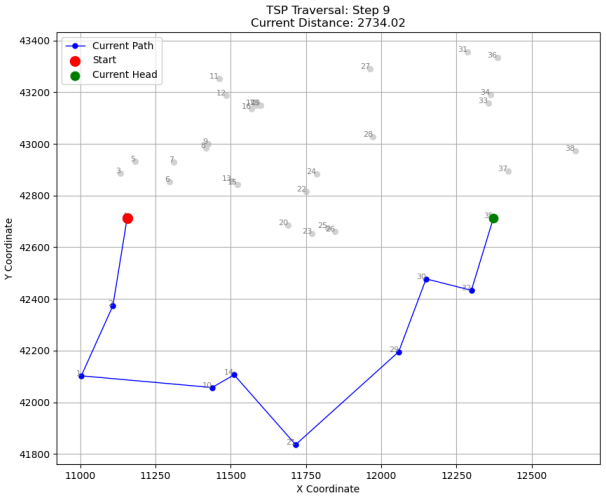

TSP Traversal: Step 4
Current Distance: 1158.01
Current Path
Start
Current Head
Y Coordinate
X Coordinate

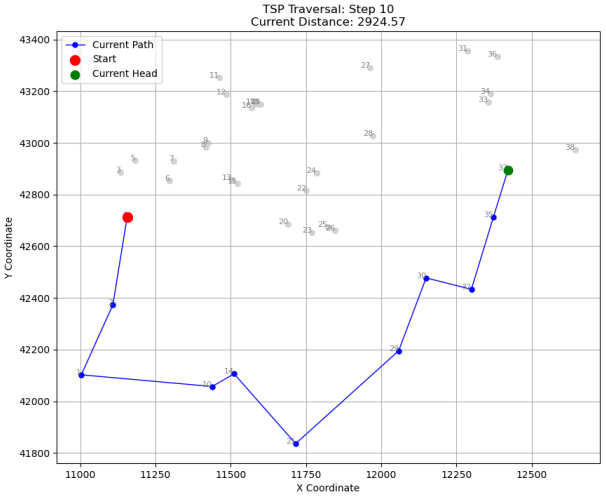

TSP Traversal: Step 5
Current Distance: 1496.91
Current Path
Start
Current Head
Y Coordinate
X Coordinate

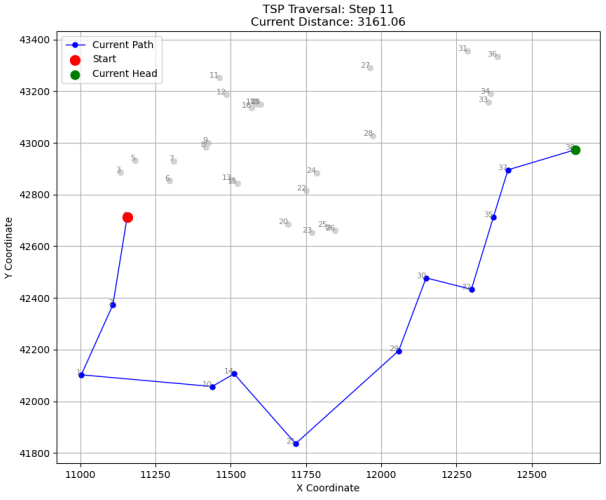

TSP Traversal: Step 6
Current Distance: 1993.40
Current Path
Start
Current Head
Y Coordinate
X Coordinate

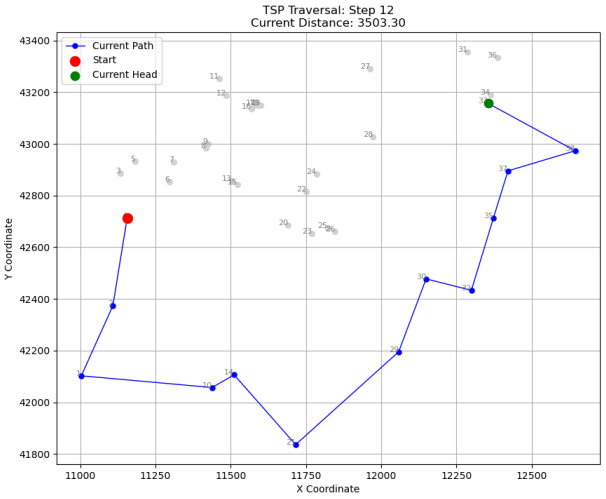

TSP Traversal: Step 7
Current Distance: 2289.70
Current Path
Start
Current Head
Y Coordinate
X Coordinate

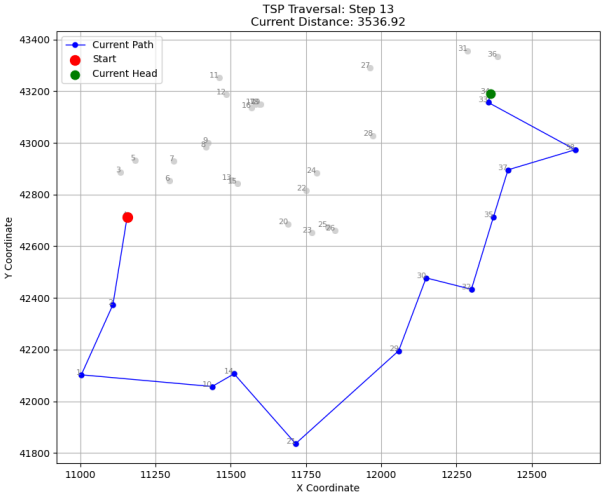

TSP Traversal: Step 8
Current Distance: 2446.60
Current Path
Start
Current Head
Y Coordinate
X Coordinate

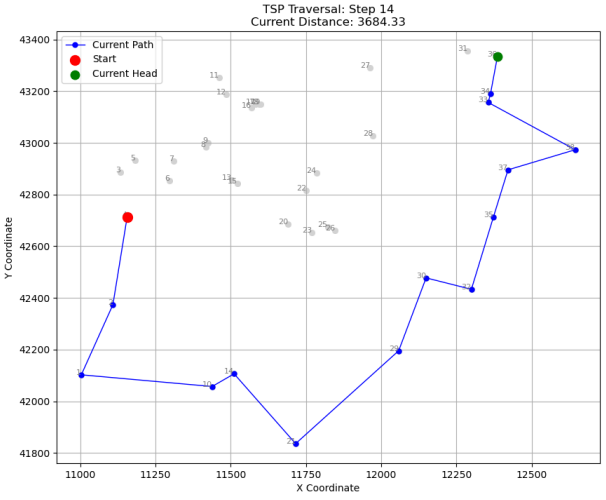

TSP Traversal: Step 9
Current Distance: 2734.02
Current Path
Start
Current Head
Y Coordinate
X Coordinate

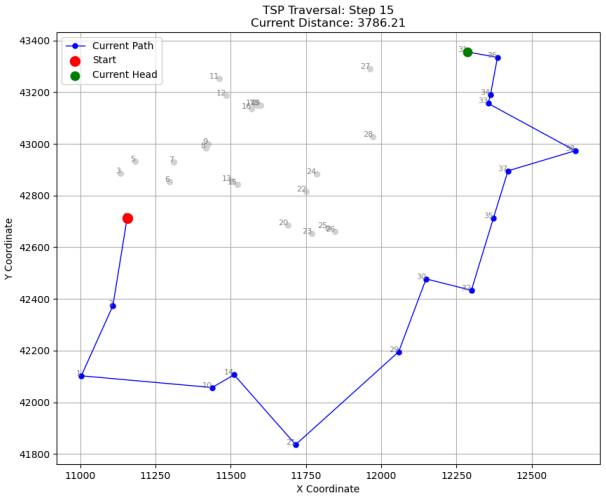

TSP Traversal: Step 10
Current Distance: 2924.57
Current Path
Start
Current Head
Y Coordinate
X Coordinate

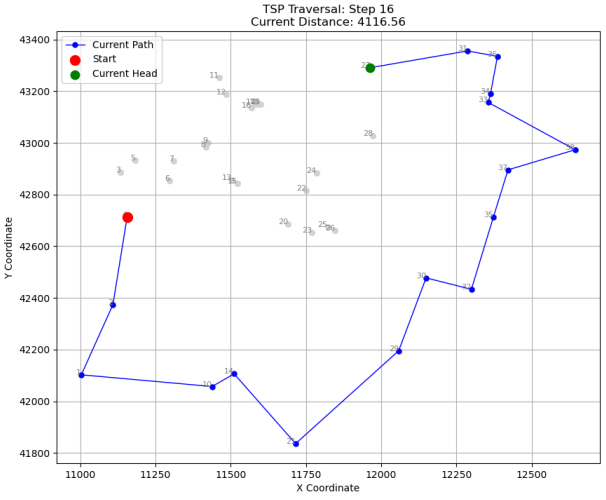

TSP Traversal: Step 11
Current Distance: 3161.06
Current Path
Start
Current Head
Y Coordinate
X Coordinate

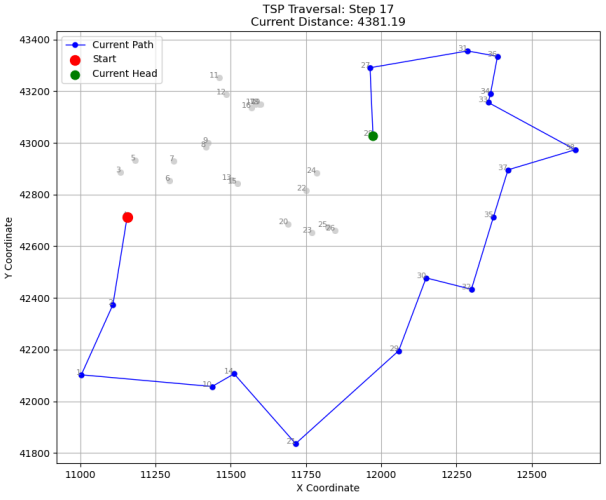

TSP Traversal: Step 12
Current Distance: 3503.30
Current Path
Start
Current Head
Y Coordinate
X Coordinate

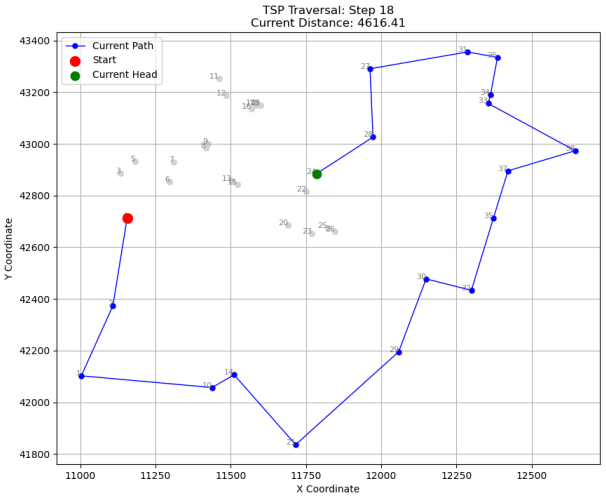

TSP Traversal: Step 13
Current Distance: 3536.92
Current Path
Start
Current Head
Y Coordinate
X Coordinate

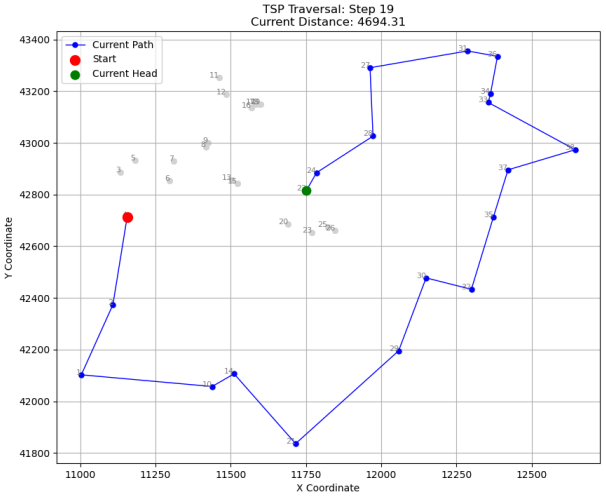

TSP Traversal: Step 14
Current Distance: 3684.33
Current Path
Start
Current Head
Y Coordinate
X Coordinate

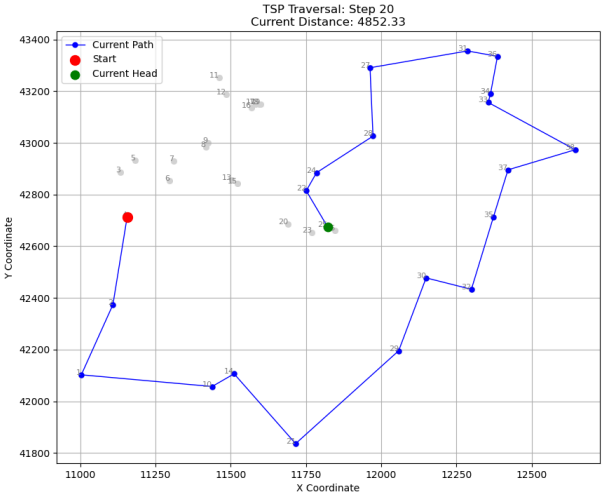

TSP Traversal: Step 15
Current Distance: 3786.21
Current Path
Start
Current Head
Y Coordinate
X Coordinate

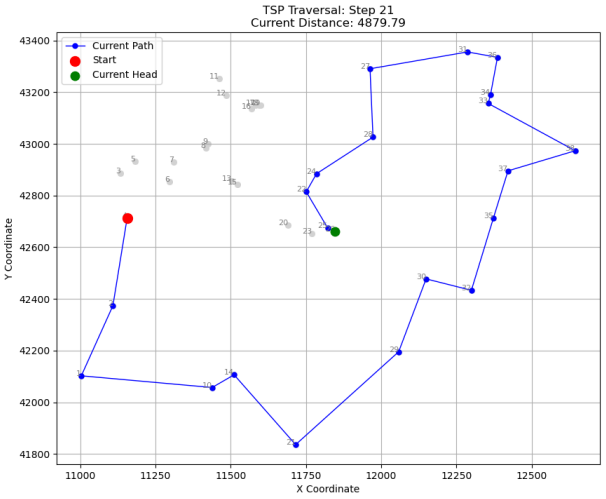

TSP Traversal: Step 16
Current Distance: 4116.56
Current Path
Start
Current Head
Y Coordinate
X Coordinate

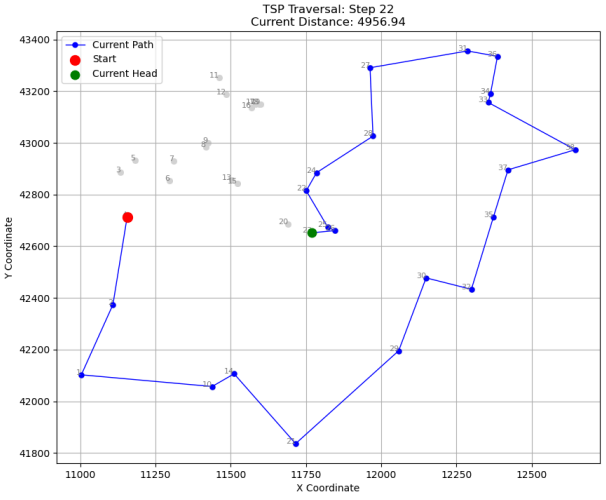

TSP Traversal: Step 17
Current Distance: 4381.19
Current Path
Start
Current Head
Y Coordinate
X Coordinate

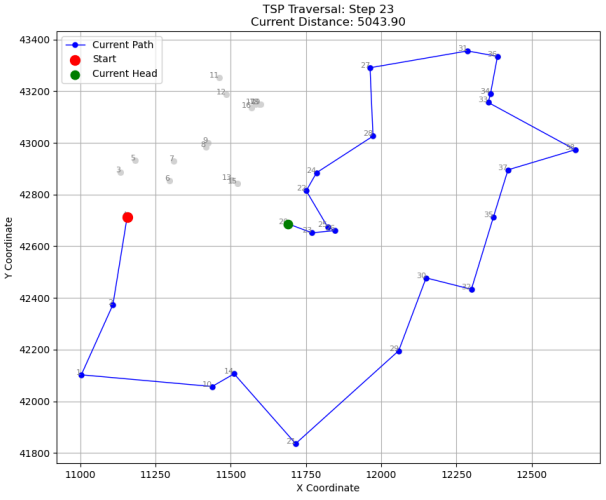

TSP Traversal: Step 18
Current Distance: 4616.41
Current Path
Start
Current Head
Y Coordinate
X Coordinate

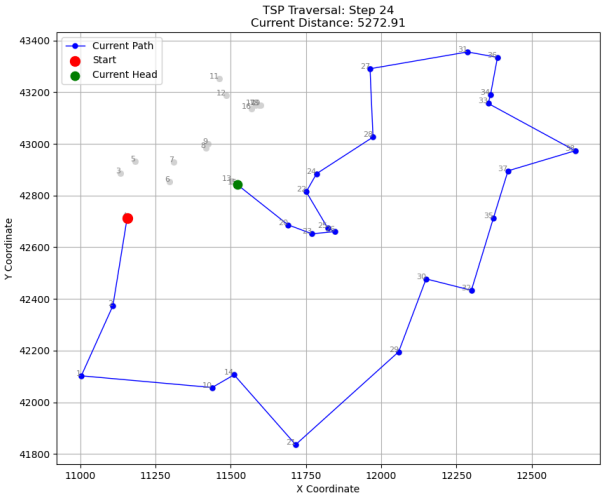

TSP Traversal: Step 19
Current Distance: 4694.31
Current Path
Start
Current Head
Y Coordinate
X Coordinate

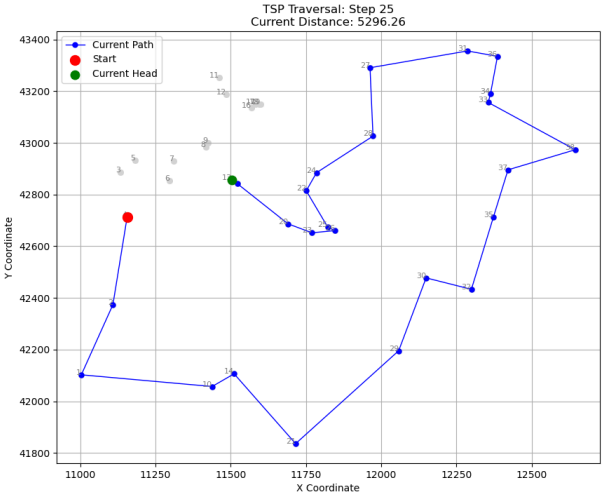

TSP Traversal: Step 20
Current Distance: 4852.33
Current Path
Start
Current Head
Y Coordinate
X Coordinate

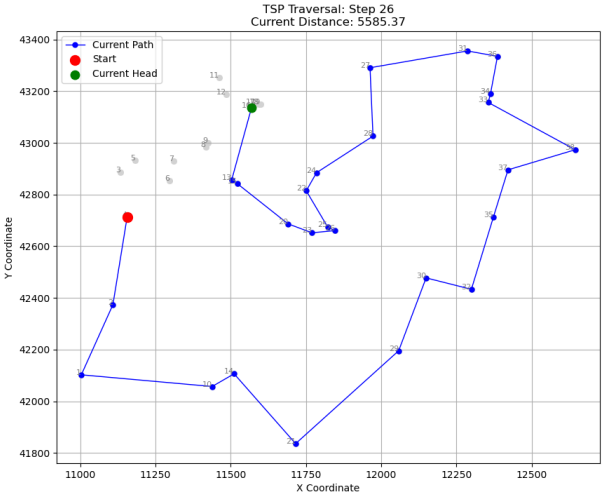

TSP Traversal: Step 21
Current Distance: 4879.79
Current Path
Start
Current Head
Y Coordinate
X Coordinate

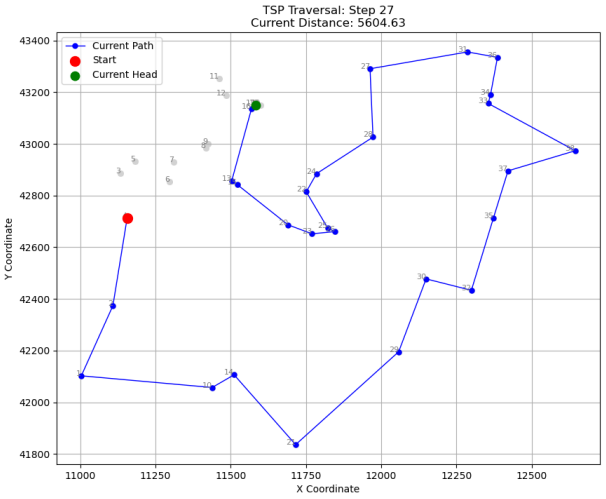

TSP Traversal: Step 22
Current Distance: 4956.94
Current Path
Start
Current Head
Y Coordinate
X Coordinate

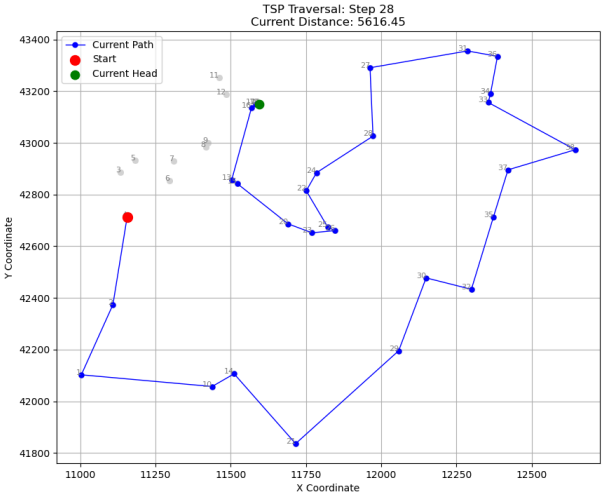

TSP Traversal: Step 23
Current Distance: 5043.90
Current Path
Start
Current Head
Y Coordinate
X Coordinate

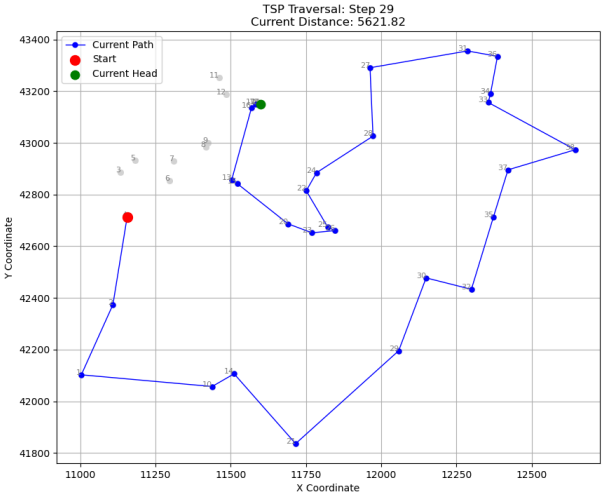

TSP Traversal: Step 24
Current Distance: 5272.91
Current Path
Start
Current Head
Y Coordinate
X Coordinate

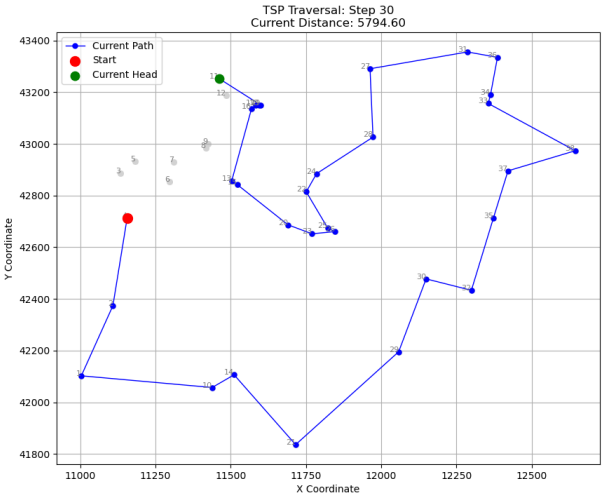

TSP Traversal: Step 25
Current Distance: 5296.26
Current Path
Start
Current Head
Y Coordinate
X Coordinate

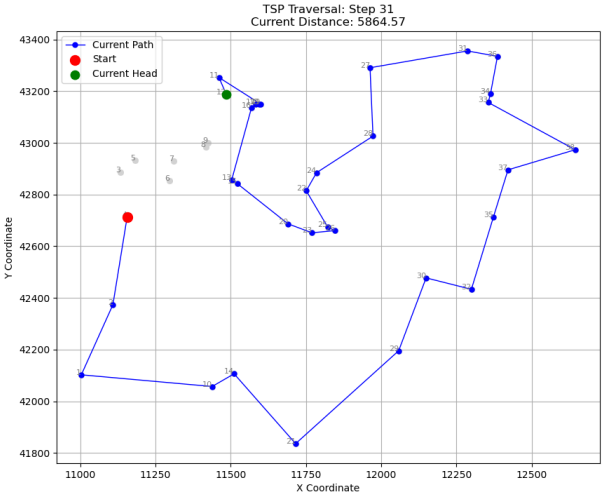

TSP Traversal: Step 26
Current Distance: 5585.37
Current Path
Start
Current Head
Y Coordinate
X Coordinate

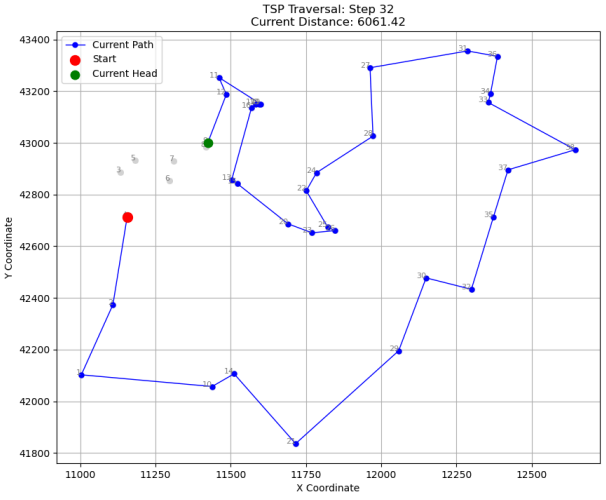

TSP Traversal: Step 27
Current Distance: 5604.63
Current Path
Start
Current Head
Y Coordinate
X Coordinate

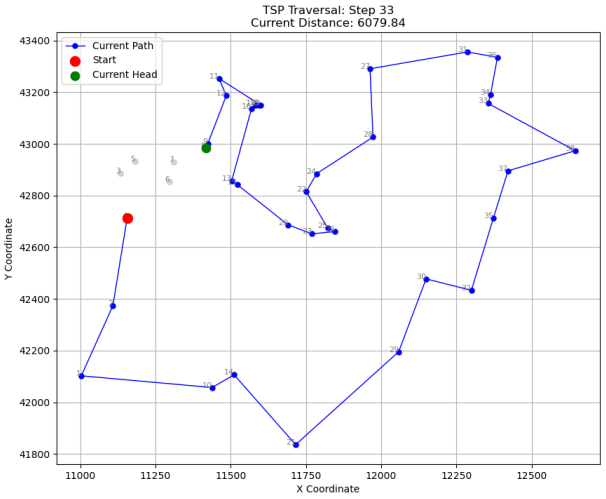

TSP Traversal: Step 28
Current Distance: 5616.45
Current Path
Start
Current Head
Y Coordinate
X Coordinate

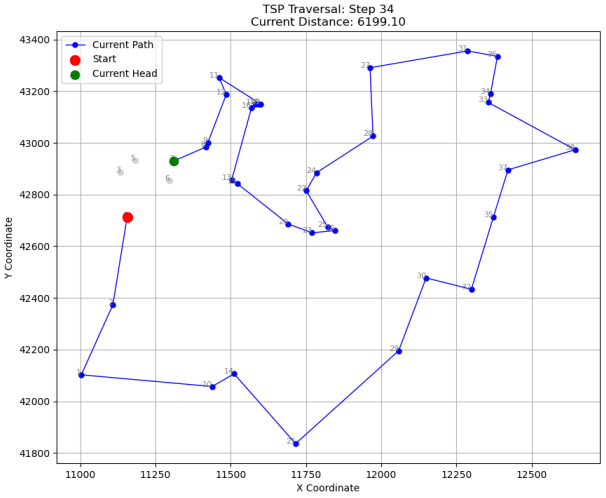

TSP Traversal: Step 29
Current Distance: 5621.82
Current Path
Start
Current Head
Y Coordinate
X Coordinate

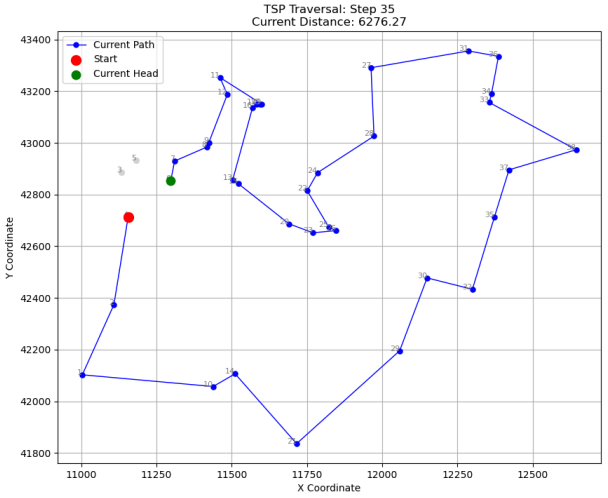

TSP Traversal: Step 30
Current Distance: 5794.60
Current Path
Start
Current Head
Y Coordinate
X Coordinate

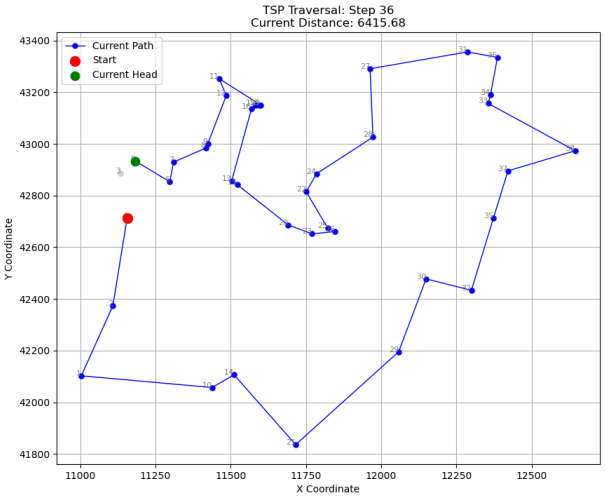

TSP Traversal: Step 31
Current Distance: 5864.57
Current Path
Start
Current Head
Y Coordinate
X Coordinate

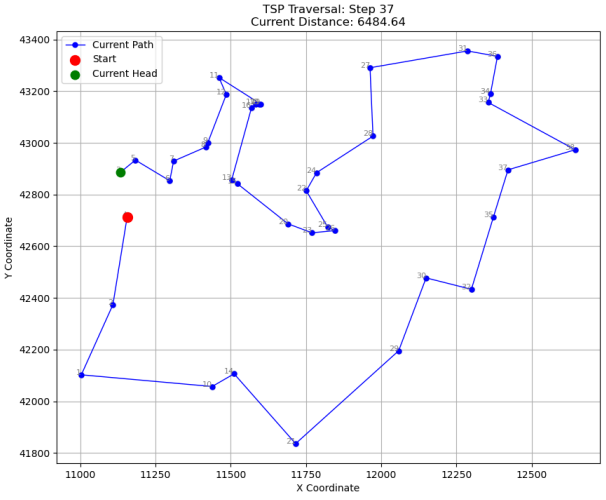

TSP Traversal: Step 32
Current Distance: 6061.42
Current Path
Start
Current Head
Y Coordinate
X Coordinate

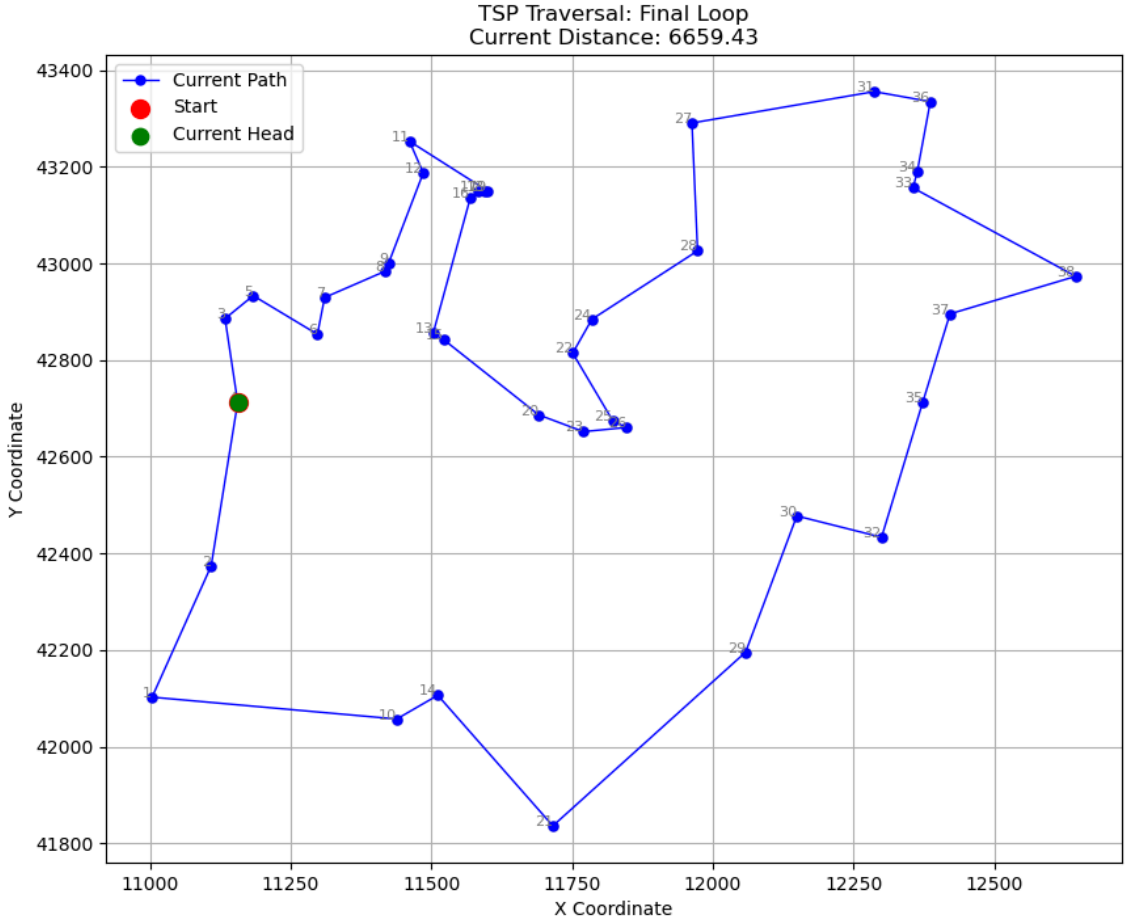

TSP Traversal: Step 33
Current Distance: 6079.84
Current Path
Start
Current Head
Y Coordinate
X Coordinate


TSP Traversal: Step 34
Current Distance: 6199.10
Current Path
Start
Current Head
Y Coordinate
X Coordinate


TSP Traversal: Step 35
Current Distance: 6276.27
Current Path
Start
Current Head
Y Coordinate
X Coordinate


TSP Traversal: Step 36
Current Distance: 6415.68
Current Path
Start
Current Head
Y Coordinate
X Coordinate
43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11250
11500
11750
12000
12250
12500

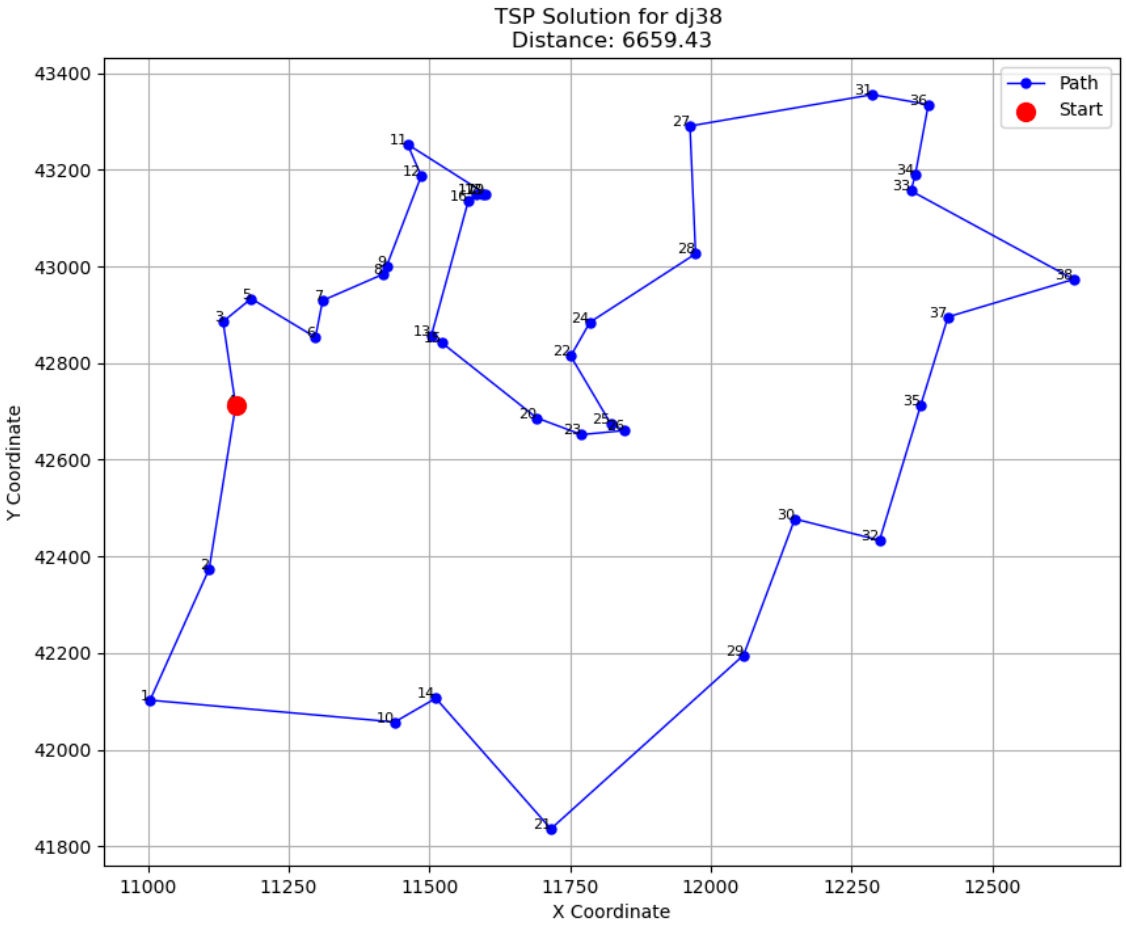

TSP Traversal: Step 37
Current Distance: 6484.64
Current Path
Start
Current Head
Y Coordinate
X Coordinate
43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11250
11500
11750
12000
12250
12500



（3）代码运行结果截图

问题 输出 调试控制台 终端

```
                                python solve_tsp.py
Loaded 38 cities from dj38.tsp
Starting SA... Initial Distance: 30054.64
Finished in 135.36s

==============================
Optimal Solution Found:
Shortest Distance: 6659.4315
   QT_SCREEN_SCALE_FACTORS to set per-screen factors.
   QT_SCALE_FACTOR to set the application global scale factor.
Final solution image saved as 'tsp_final_solution.png'
Generating traversal frames in 'frames/'...
Saved 39 frames to 'frames/'.
```



2. 推荐算法

（1）算法设计思想和伪代码

算法设计思想：MF-Markov 混合推荐模型

在 MOOC 课程推荐场景中，用户的行为受两种主要因素驱动：

1. 长期兴趣 (Long-term Interest)：用户偏好某一类主题（如“计算机科学”、“历史”）。这部分偏好是静态且全局的，适合用 矩阵分解 (Matrix Factorization, MF) 来捕捉。
2. 短期序列模式 (Short-term Sequential Pattern)：用户在学习一门课后，往往会紧接着学习其后续课程（如“微积分 I” -> “微积分 II”）。这种转移概率非常强，适合用 马尔可夫链 (Markov Chain) 来捕捉。

本算法采用线性加权融合** (Linear Ensemble) 策略，结合了两者的优势：

- 使用 BPR-MF 学习用户和物品的隐向量，捕捉个性化偏好。
- 使用一阶马尔可夫链 (First-order Markov Chain) 统计物品间的转移概率，捕捉课程间的强关联。
- 最终分数是两者的加权和：$Score_{final} = \alpha \cdot Score_{MF} + (1 - \alpha) \cdot Score_{Markov}$。

算法伪代码：

```
算法 MF_Markov_Ensemble_Recommender
    输入:
        train_data (训练集: 用户-物品交互序列)
```

```
        test_data (测试集: 用户-物品交互)
        alpha (融合权重, 0 <= alpha <= 1)
        K (推荐列表长度)
    输出:
        HR@K, NDCG@K (评测指标)

    // 阶段 1: 训练 BPR-MF 模型 (捕捉长期兴趣)
    初始化 用户矩阵 P, 物品矩阵 Q (随机小数值)
    循环 epoch 从 1 到 Max_Epochs:
        从 train_data 中采样三元组 (u, i, j) // u:用户, i:正样本, j:负样本
        计算预测差值: x_uij = (P[u]·Q[i]) - (P[u]·Q[j])
        计算梯度并更新 P[u], Q[i], Q[j] (使用 BPR 损失函数)
    结束循环

    // 阶段 2: 训练 Markov 模型 (捕捉序列模式)
    初始化 转移矩阵 Transitions (字典或稀疏矩阵)
    对于 train_data 中的每个用户 u:
        按时间排序该用户的交互物品序列 [item_1, item_2, ..., item_T]
        对于 t 从 1 到 T-1:
            prev_item = item_t
            next_item = item_{t+1}
            Transitions[prev_item][next_item] += 1
        结束循环
    结束循环

    // 阶段 3: 混合预测与评估
    初始化 Hits = 0, NDCGs = 0
    对于 test_data 中的每个用户 u 及目标物品 target_item:
        last_item = 用户 u 在训练集中最后交互的物品
        candidates = {target_item} + 99个随机负样本

        对于 candidates 中的每个 item:
            // 计算 MF 分数
            score_mf = P[u] · Q[item]

            // 计算 Markov 分数 (如果是 last_item 的后续课程则得分高, 否则为0)
            score_markov = Transitions[last_item][item]

        // 分数归一化 (Min-Max Normalization)
        norm_mf = Normalize(score_mf_list)
        norm_markov = Normalize(score_markov_list)

        // 加权融合
        final_scores = alpha * norm_mf + (1 - alpha) * norm_markov

        // 排序并计算指标
        rank = Rank(final_scores, target_item)
        如果 rank <= K:
            Hits += 1
            NDCGs += 1 / log2(rank + 1)

    返回 Hits/Num_Test, NDCGs/Num_Test
结束算法
```

（2）推荐结果的评测结果

在测试集上，我们使用 Hit Ratio (HR@10) 和 NDCG@10 作为评价指标。测试策略为：对于测试集中的每个用户，将其真实交互的一个课程与 99 个随机负样本课程混合，模型需要将真实课程排在前 10 名。

- HR@10 衡量能不能推荐对 （是否出现在前10名里）。
- NDCG@10 衡量推荐得准不准 （在出现的前提下，排名是否靠前）。

根据运行日志，**最佳融合权重为 Alpha = 0.83** (即 83% BPR-MF + 17% Markov)，此时效果最好：

指标 (Metric)	Top-K	结果 (Score)	说明
Hit Ratio (HR)	10	0.6848	约 68.48% 的测试样本被成功推荐到了前 10 位。
NDCG	10	0.4776	归一化折损累计增益，反映了推荐结果的排序质量。

算法模型	Hit Ratio (HR@10)	NDCG@10	评价
随机猜测 (Random)	~0.1000	~0.0300	理论下限，完全无推荐能力
热门推荐 (Popularity)	0.5686	0.3411	强基准，代表 “随大流” 的效果
纯 Markov	0.6268	0.4348	当用户行为符合大众序列时，Markov 权重生效。
纯 BPR-MF	0.6801	0.4674	当序列断裂或冷启动时，MF 提供的个性化偏好。
MF-Markov	0.68.48	0.4776	**显著优于基准**

- **热门推荐**的数据来自数据集上跑出的实测代码结果 。
- **随机猜测**的数据来自针对测试协议（1正99负）的 数学理论推导 。

附：热门推荐伪代码：

```
算法：基于热门度的推荐基准 (Popularity Baseline)

输入：训练集 TrainData，测试集 TestData，负样本集 TestNegatives，推荐数量 K
输出：Hit Ratio (HR)，NDCG 指标

1. 计算热门度 (Training Phase):
   ItemCounts = 字典{}
   对于 TrainData 中的每条交互记录 (User, Item):
       如果 Item 在 ItemCounts 中:
           ItemCounts[Item] += 1
       否则:
           ItemCounts[Item] = 1

2. 评估阶段 (Evaluation Phase):
   Hits = 0
```

```
    NDCG_Sum = 0
    NumTest = TestData 的样本数量

    对于 TestData 中的每条记录 (User, GT_Item) 及对应的 Neg_Items:

        a. 构建候选集合:
           Candidates = [GT_Item] + Neg_Items  (共 100 个课程)

        b. 预测分数 (基于热门度):
           Scores = []
           对于 Candidates 中的每个 Item:
               Score = ItemCounts.get(Item, 0)  (如果训练集中没出现过，热度为0)
               Scores.添加(Score)

        c. 排序与计算排名:
           GT_Score = Scores[0]  (真实课程的热度分)
           Rank = 1
           对于 Scores 中的其他分数 s:
               如果 s > GT_Score:
                   Rank += 1

        d. 计算指标:
           如果 Rank <= K:
               Hits += 1
               NDCG_Sum += 1 / log2(Rank + 1)

3. 输出结果:
    HR = Hits / NumTest
    NDCG = NDCG_Sum / NumTest
    返回 HR, NDCG
```

（3）代码运行结果截图