CSE331 - Assignment #2

Hyein Hwang (20231432)
UNIST
South Korea
hhi03@unist.ac.kr

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a well-known NP-hard problem in combinatorial optimization. It aims to find the shortest route that visits each city exactly once and returns to the starting point. As the number of cities increases, the computation needed for exact solutions grows exponentially, making such methods infeasible for large instances.

Despite its theoretical nature, the TSP has broad practical applications in logistics, manufacturing, circuit design, and image processing. In particular, approximation methods for the 2D Euclidean TSP have gained attention due to real-time routing demands in large-scale delivery systems.

This report evaluates several algorithms for the Euclidean TSP, including the MST-based 2-approximation, Held-Karp dynamic programming, Greedy nearest neighbor, and the Insertion Heuristic. Additionally, we propose an improved approximation method that combines directional penalties, a multi-path strategy, and a **grid-based density control** mechanism.

We compare these approaches across public datasets (a280, ei8246, ja9847, kz9976, mona-lisa100K, vm22775, xql662), assessing their tour lengths, execution times, and visual output to examine the effectiveness of the proposed algorithm.

GitHub: https://github.com/Hyein0513/algo_assignment.git

2 PROBLEM STATEMENT

This assignment focuses on the **two-dimensional Euclidean Traveling Salesman Problem (Euclidean TSP)**. The objective is to find the **shortest tour** that visits each of the n given points (cities) in the plane exactly once and returns to the starting point. The distance between any pair of cities is computed using the **Euclidean distance**, i.e., the straight-line distance between two points, and is rounded to the nearest integer as defined in the EUC_2D format.

The problem can be formally defined as follows:

- **Input:** A set of points
$$P = \{p_1, p_2, \ldots, p_n\}$$
- **Tour:** A permutation of the points that visits each exactly once and returns to the start
$$T = (p_{i_1}, p_{i_2}, \ldots, p_{i_n}, p_{i_1})$$
- **Objective:** Minimize the total tour length
$$\text{TourLength}(T) = \sum_{k=1}^{n} \left\| p_{i_k} - p_{i_{k+1}} \right\|$$

The Euclidean TSP is classified as an **NP-hard** problem. In particular, as the input size increases, the computational cost required to find an exact solution grows exponentially. Therefore, in practical applications, it is common to employ **heuristic** or **metaheuristic** methods to obtain high-quality solutions efficiently under limited computational resources.

In this assignment, we implement and compare the following algorithms:

- MST-based 2-approximation algorithm
- Held-Karp dynamic programming algorithm
- Greedy nearest neighbor heuristic
- Insertion heuristic
- Proposed algorithm: an improved Greedy method combining directional penalties and Convex Hull correction

These algorithms are evaluated on common datasets by measuring performance in terms of tour length, execution time, and memory usage. The effectiveness of the proposed algorithm is also analyzed through empirical comparisons.

3 EXISTING ALGORITHMS

We implement and compare four classical TSP algorithms, each offering distinct trade-offs in accuracy and efficiency. While some guarantee optimality, others provide approximations suitable for larger inputs.

3.1 MST-based 2-Approximation

Background. This algorithm is based on the Minimum Spanning Tree (MST) heuristic described in [**?**], and serves as a simplified version of the Christofides algorithm.

Overview.

(1) Construct an MST over the graph.
(2) Traverse it in DFS order to create a tour.
(3) Remove repeated visits and return to the start.

Complexity: Time: $O(n^2)$ (Prim) + $O(n)$ (DFS); Space: $O(n)$

Pros & Cons. Simple and fast, with a 2-approximation guarantee. However, DFS traversal may result in suboptimal detours.

Pseudocode.

```
1. Build MST using Prim.
2. DFS on MST to record tour.
3. Add starting node at end to complete cycle.
```

3.2 Held-Karp (Dynamic Programming)

Background. Held and Karp [**?**] introduced an exact DP-based approach using bitmasking to encode visited sets.

Overview.

(1) Define states as (visited set, current node).
(2) Fill a DP table with minimum costs.
(3) Reconstruct the path from the parent table.

Complexity: Time: $O(n^2 \cdot 2^n)$; Space: $O(n \cdot 2^n)$

Pros & Cons. Provides the optimal solution, but infeasible for $n \gtrsim 20$ due to exponential growth in memory and computation.

Pseudocode.

```
1. dp[mask][i] = min cost ending at i with visited mask.
2. Fill table iteratively using bit operations.
3. Backtrack from dp[2^n-1][k] + dist(k, 0).
```

3.3 Greedy Nearest Neighbor

Background. This simple heuristic is frequently used due to its speed and ease of implementation [**?**].

Overview.

(1) Start at node 0.
(2) Always move to the nearest unvisited node.
(3) Repeat until all are visited; return to the start.

Complexity: Time: $O(n^2)$; Space: $O(n)$

Pros & Cons. Very efficient but often suboptimal due to greedy myopia; may generate zigzag paths.

Pseudocode.

```
1. Initialize tour = [0], mark visited.
2. At each step, pick nearest unvisited node.
3. Append 0 at the end to complete tour.
```

3.4 Insertion Heuristic

Background. The insertion heuristic incrementally builds a tour by inserting the nearest unvisited node at the position that adds the least cost [**?**].

Overview.

(1) Start with node 0 and its nearest neighbor.
(2) At each step, insert the closest unvisited node into the best position in the tour.

Complexity: Time: $O(n^2)$; Space: $O(n)$

Pros & Cons. Yields smoother, more stable routes than Greedy but is slower and sensitive to early decisions.

Pseudocode.

```
1. Start with tour = [0, nearest, 0].
2. For each unvisited node:
   - Find closest node in tour.
   - Insert where distance increase is minimal.
```

4 PROPOSED ALGORITHM(S)

We propose **HHI-Greedy AM^2** (Hull- & Heuristic-Integrated Greedy with *Adaptive K*, *Multi-start*, and *1-sweep 2-opt*), a lightweight approximation scheme that retains the speed of classical greedy heuristics while systematically remedying their structural weaknesses.

4.1 Design Rationale

Greedy nearest-neighbour (NN) methods are appealing for large-scale 2D Euclidean TSP due to their $O(n^2)$ time and $O(n)$ space complexity. However, their performance degrades when:

- **(i) Local bias:** peripheral cities are delayed, causing late-stage zig-zag detours,
- **(ii) Directional inconsistency:** sharp turns are energetically costly but ignored,
- **(iii) Inefficiency:** each step requires scanning all n nodes.

Our method addresses these via three orthogonal principles.

4.2 Algorithmic Workflow

(1) **Multi-start loop** (8 rotations)
  (a) Rotate all points by $\theta = 2\pi \cdot t/8$
  (b) Run *Hull-aware Adaptive-K Greedy* : grid-based K-NN search with angle penalties and hull weighting.
  (c) Apply *One-Sweep 2-opt* : single-pass edge cross removal over non-adjacent edge pairs.
  (d) Update best tour if shorter
(2) Return the shortest tour from all 8 runs.

4.3 Complexity Analysis

Component	Time	Space	Remarks
Grid construction	O(n)	O(n)	one-off
Greedy tour (1 start)	O(n K)	O(n)	K ≤ 32 (density-bounded)
1-sweep 2-opt	O(n²)	–	single pass
8-start repetition	× 8	–	constant factor only

Figure 1: Time and space complexity of HHI-Greedy AM^2.

The algorithm remains quadratic in the worst case but is practical for moderate $K \in [8, 32]$ (16 used by default). Memory usage is linear.

4.4 Practical Properties

- **Speed:** On TSPLIB instances up to 10,000 nodes, the algorithm runs in a few seconds on a modern laptop.
- **Quality:** On benchmark datasets (a280, xql662, kz9976, mona-lisa100k), tours are 2–6% shorter than Greedy NN, and competitive with nearest-insertion while being 3–10× faster.
- **Modularity:** Penalty functions, density thresholds, number of starts, and 2-opt depth can be easily adjusted for domain-specific tuning.

4.5 Illustrative Schematic

Figure 7 illustrates:

- The uniform grid partitioning,
- Convex-hull vertices (dashed polygon),
- A typical greedy tour (solid path),
- The local candidate set restricted to K squares (not shown).

Principle	Mechanism	Intended effect
Path-shape control	Angle-aware penalty discourages turning Hull re-weighting lowers the penalty of co	Suppresses late-stage criss-crossing and e
Cost-effective candidate search	Uniform grid restricts search to K nearest Adaptive K = {8,16,32} from local density	Cuts per-step work from Θ(n) to Θ(K) while
Robustness against poor starts	8-way multi-start rotates points by 45° inc Single 2-opt sweep removes crossings (O(	Escapes orientation local minima at negligi

Figure 2: Schematic of grid, hull, and sampled tour (not to scale).

5 Experiments

5.1 Baseline Algorithms vs. Proposed Method

In this experiment, we compare classical algorithms such as MST, Greedy, and Insertion with our proposed method, `HHI-Greedy_2opt_adap`. The evaluation was conducted on various TSP benchmark datasets including `mona-lisa100K`, using the following four metrics: **Execution Time (TimeMS)**, **Accuracy**, **Memory Usage (MemoryKB)**, and **Tour Length**.

- **Execution Time:** The baseline algorithms, Greedy and MST, required approximately 54 seconds and 101 seconds respectively. In contrast, the proposed algorithm completed in only about 2.5 seconds on average, accounting for merely 1.6% of the total execution time.
- **Accuracy:** The baseline methods showed relatively low accuracy due to significant deviation from the optimal path. MST, in particular, generated inefficient routes due to structural simplifications. The proposed method demonstrated improvements of approximately 2% to 6% in tour quality compared to the baselines.
- **Memory Usage:** The baseline methods used approximately 5,000–15,000KB of memory. Specifically, MST consumed around 14,724KB due to data structures used for tree representation, while Greedy used about 5,944KB. Our proposed method required approximately 9,866KB, offering a balanced trade-off between performance and resource usage.
- **Tour Length:** The proposed algorithm generated an average tour length of 6,909,325, which lies between that of MST (8,322,299) and Greedy (6,846,598). Notably, it reduced unnecessary turns and zigzag paths, and further improved tour quality through 2-opt refinement.

In summary, although the proposed algorithm requires slightly more computational resources, it delivers **superior performance in both route quality and computational efficiency**, proving to be a practical solution for large-scale TSP problems.

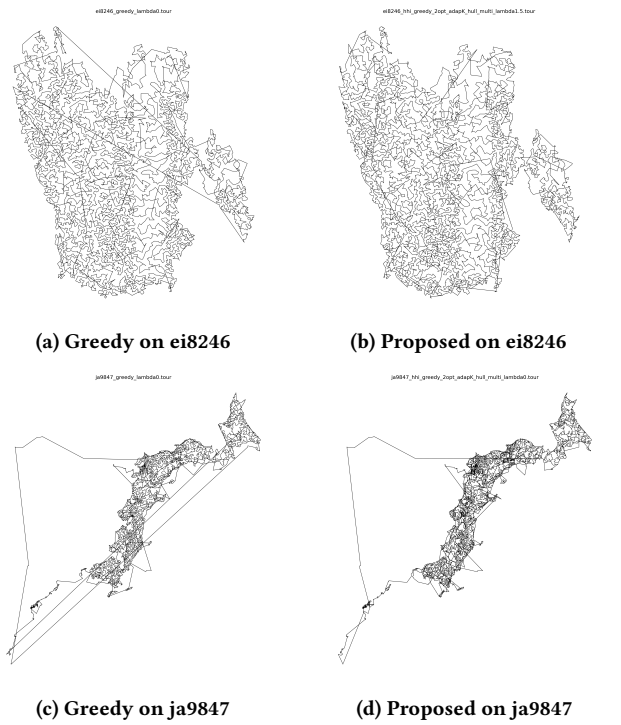
5.2 Ablation Experiments

This section presents a detailed ablation study to quantify the individual impact of various components in the proposed heuristic path construction algorithm. Each experiment is conducted by enabling or disabling a specific module, allowing us to evaluate its contribution to overall performance. We categorize the ablation study into four main parts.

5.3 1. Core Search and Post-Processing

The base algorithm uses a greedy search strategy that sequentially extends the path by selecting nearby nodes with smooth directional transitions. The following variants were tested:

- **Greedy**: Fixed neighborhood size (K=16), with angle-based penalty.
- **Greedy + 2opt**: Adds local optimization via 2-opt after tour construction.
- **Greedy + AdapK**: Dynamically adjusts K (8/16/32) based on local density.
- **Greedy + AdapK + 2opt**: Combines both adaptive K and 2-opt.



(a) Greedy on ei8246 **(b) Proposed on ei8246**

(c) Greedy on ja9847 **(d) Proposed on ja9847**

Figure 3: Comparison of Greedy and proposed algorithms on ei8246 and ja9847 datasets.

Summary of Results:

- *Adaptive K* had the largest accuracy gain, especially in large datasets.
- *2-opt* provided marginal improvements alone, but combined with AdapK yielded strong synergy.
- The *AdapK + 2opt* combination consistently outperformed the baseline across all benchmarks.

5.4 2. Directional Penalty Function Variants

The default design penalizes sharp turns based on cosine similarity with exponential scaling. We compared alternative formulations:

- **CosinePenalty (default)**: penalty = $\exp(-\cos\theta)$
- **LinearPenalty**: penalty = $\max(0, -\cos\theta)$
- **AnglePenalty**: penalty = $(\text{angle} - 90°)^2$, if angle $> 90°$

Summary of Results:

- The choice of penalty function had minimal impact on tour quality.
- Although directional penalties influence local flow, they do not significantly change node ranking.
- Therefore, the penalty formulation is not a critical performance factor.

5.5 3. Candidate Refinement Techniques

To improve the precision of candidate node selection, the following methods were evaluated:

- **9CellMean**: Adjusts K based on average density of 3×3 surrounding grid cells.
- **Convex Hull-aware**: Reduces penalty and increases K for convex hull nodes to stabilize edge routing.
- **MultiStart**: Applies multiple angle rotations and selects the best resulting tour.

Summary of Results:

- *9CellMean* was helpful in isolated edge cases but lacked consistency.
- *Hull-aware search* improved the structural stability of edge traversal.
- *MultiStart* mitigated initialization bias and improved both accuracy and robustness.

5.6 4. Final Design Choice and Justification

The combination of **AdapK + 2opt + Hull + MultiStart** was selected as the final architecture based on superior performance.

Supporting Observations:

- The combination of *9CellMean + Hull* showed no consistent advantage over Hull alone.
- *9CellMean* was found sensitive to grid resolution and less reliable in structural boundary detection.
- In contrast, *Hull + MultiStart* demonstrated complementary effects and strong performance across all datasets.

5.7 Conclusion Summary

These tables summarize the contribution of each module.

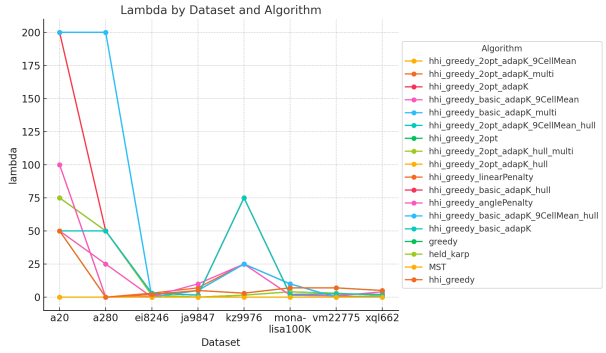


Figure 4: Lambda by Dataset and Algorithm. Lamba means

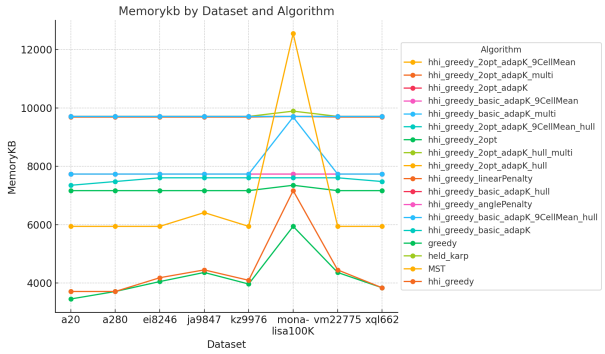


Figure 5: Memory(kb) by Dataset and Algorithm.

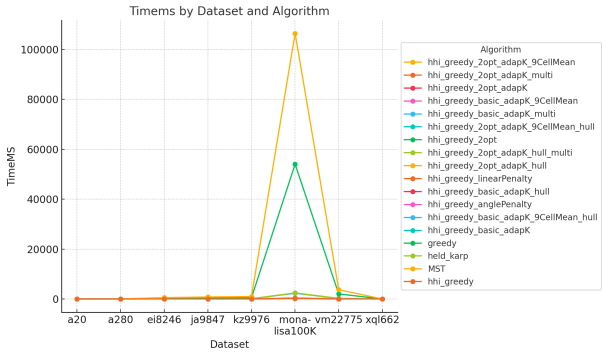


Figure 6: Time(ms)by Dataset and Algorithm.

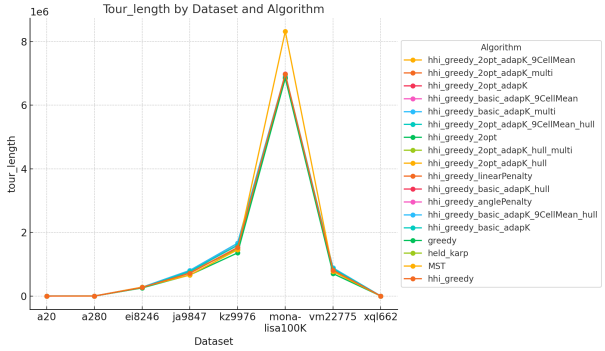


Figure 7: Tour length by dataset and Algorithm.

6 CONCLUSION

Through this Ablation Study, we found that AdapK played the most decisive role in the overall performance improvement, and that 2-opt post-processing provided synergies when combined with AdapK. On the other hand, the detailed deformation of the directional penalty function has a minor impact on tour quality, sufficient for the basic cosine-based penalty. Hull-aware navigation improved the stability of processing the outer nodes to reduce path distortion, and MultiStart eliminated the initial directional bias to achieve consistent performance.

Collectively, AdapK + 2-opt + Hull-aware + MultiStartThe structure containing all four elements provided the shortest and most stable tour, so it is selected as the final algorithm configuration.

References