CSE331 - Assignment #2

Deokhyeon Kim (20201032)
UNIST
South Korea
ejrgus1404@unist.ac.kr

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the most fundamental and widely studied combinatorial optimization problems. It has numerous real-world applications in areas such as logistics, circuit design, robotics, and DNA sequencing. The problem asks: given a list of cities and the distances between each pair, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Due to its NP-hard nature, solving the TSP optimally is computationally expensive, especially as the number of nodes increases. Therefore, both exact and approximate algorithms have been developed to address it under different scenarios.

This project aims to study the TSP by implementing two well-known algorithms (Held-Karp and MST-based approximation), as well as designing and analyzing novel heuristics inspired by MO's algorithm and Hilbert-MO. We compare these algorithms based on their runtime and approximation quality over several benchmark datasets, including both small and large samples of them.

2 PROBLEM STATEMENT

The objective of the Traveling Salesman Problem (TSP) is to find the shortest possible tour that visits a set of n cities exactly once and returns to the starting city. The TSP can be modeled as a complete weighted graph $G = (V, E)$ with $|V| = n$, where each vertex $v_i \in V$ represents a city, and each edge $(v_i, v_j) \in E$ has an associated distance $d(v_i, v_j)$.

The distance function d is assumed to satisfy the following properties:

- (Identity) $d(u, v) = 0 \iff u = v$
- (Symmetry) $d(u, v) = d(v, u)$ for all u, v
- (Triangle inequality) $d(u, v) \leq d(u, w) + d(w, v)$ for all u, v, w

These conditions ensure that d is a metric, which is crucial for approximation guarantees in many TSP algorithms.

In this project, we focus on the 2D geometric version of TSP, where each city corresponds to a point (x_i, y_i) in the 2D plane. The distance between two cities is computed using either the L_1 (Manhattan) or L_2 (Euclidean) norm, both of which are valid metrics satisfying the properties of identity, symmetry, and the triangle inequality:

$$\mathrm{L}_1(i, j) = |x_i - x_j| + |y_i - y_j|$$

$$\mathrm{L}_2(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

The goal is to compute a Hamiltonian cycle (i.e., a closed cycle that visits every node exactly once) with minimal total cost. Due to the NP-hard nature of the problem, exact algorithms become impractical for large instances. Therefore, this project focuses on designing and analyzing approximation algorithms that aim to balance solution quality with runtime efficiency.

3 EXISTING ALGORITHMS

3.1 Held-Karp Algorithm

The Held-Karp algorithm solves the TSP exactly using dynamic programming with a time complexity of $O(n^2 2^n)$. Let $dp[cur][visited]$ denote the minimum cost of a tour that is currently at city cur and has visited all cities in the set $visited$ (represented as a bitmask). The base case occurs when all cities are visited; in that case, the cost is the distance from the current city back to the starting city. The final result is obtained by computing $dp[0][\{0\}]$ recursively.

The recurrence relation is defined as:

$$\mathrm{dp}[cur][visited] = \min_{\mathrm{next} \notin visited} (\mathrm{dp}[next][visited \cup \{next\}] + \mathrm{dist}(cur, next))$$

Correctness:

The correctness of the Held-Karp algorithm follows from the principle of optimality in dynamic programming.

For any state, we consider all possible cities $next \notin visited$ and recursively compute the cost of extending the tour by visiting $next$, adding the transition cost from cur to $next$.

The recurrence relation ensures that the minimum cost is selected among all valid extensions of the current path, and since overlapping subproblems are memoized, each state is computed only once. By induction on the size of $visited$, this guarantees that the optimal cost is computed for all possible states, ultimately yielding the minimal Hamiltonian cycle.

Algorithm 1 Held-Karp Algorithm

Require: List of nodes $N = \{n_0, n_1, ..., n_{n-1}\}$
Ensure: Minimum tour cost

```
Initialize dp[cur][visited] ← ∞ for all cur and visited
Define function HK(cur, visited):
if all nodes are visited then
    return dist(cur, 0)
end if
if dp[cur][visited] is already computed then
    return dp[cur][visited]
end if
for each node i not in visited do
    cost ← HK(i, visited ∪ {i}) + dist(cur, i)
    dp[cur][visited] ← min(dp[cur][visited], cost)
end for
return dp[cur][visited]
Call: HK(0, {0})
```

Time Complexity:

The algorithm maintains $O(n \cdot 2^n)$ distinct states, where n is the number of cities. For each state, up to $O(n)$ transitions are required to explore possible next cities. Therefore, the overall time complexity is $O(n^2 \cdot 2^n)$. Due to its exponential growth, this algorithm is feasible only for small instances (typically $n \leq 20$).

3.2 MST-Based Approximation Algorithm

This is a classical 2-approximation algorithm for metric TSP.
It constructs a Minimum Spanning Tree (MST) over the graph, then generates a Hamiltonian cycle by skipping duplicate visits during an Euler tour of it.

Procedure:

- Build a complete graph using L_1 or L_2 distance as edge weight.
- Compute the MST using Kruskal's algorithm.
- Perform a full Euler tour traversal (DFS) on the MST.
- Construct the final TSP path by skipping visited nodes.
- Close the cycle by returning to the start node.

Approximation Ratio:

Let T^* denote the cost of the optimal TSP tour, and let M denote the cost of the Minimum Spanning Tree (MST) constructed on the given graph using L_1 (or L_2) distance.

Since the MST is a subgraph of the complete graph that connects all vertices without cycles, we have:

$$M \leq T^*$$

Performing a full Euler tour on the MST visits each edge exactly twice, resulting in a walk of total cost $2M$. Let W denote the sequence of nodes in this Euler tour.

We then construct a Hamiltonian cycle H by skipping already visited nodes in W (i.e., performing a shortcut), relying on the triangle inequality.

By the triangle inequality, the cost does not increase when skipping nodes. Thus, the total cost of H is at most the cost of W:

$$\text{cost}(H) \leq \text{cost}(W) = 2M \leq 2T^*$$

Hence, the final tour H produced by the algorithm satisfies:

$$\text{cost}(H) \leq 2 \cdot T^*$$

Therefore, the algorithm achieves a 2-approximation ratio.

Time Complexity:

The algorithm performs the following steps:

- $O(n^2)$ to construct the complete graph (all pairwise distances)
- $O(n^2 \log n)$ to sort $O(n^2)$ edges for Kruskal's algorithm
- $O(n)$ for depth-first traversal (Euler tour) and shortcutting to construct the final tour

Thus, the overall time complexity is $O(n^2 \log n)$.

Algorithm 2 MST-Based Approximation

Require: List of nodes $N = \{n_0, n_1, ..., n_{n-1}\}$
Ensure: Approximate TSP tour using MST and Euler tour

```
Build a complete graph
Compute MST using Kruskal's algorithm
Build adjacency list from MST edges
Perform Euler tour (DFS) starting from node 0 and record sequence
Initialize visited[n] ← false, tour ← []
for each node u in Euler tour do
    if visited[u] = false then
        Append u to tour and mark visited[u] ← true
    end if
end for
Add edge from last node in tour to first node to complete cycle

return total cost of tour
```

4 PROPOSED ALGORITHM(S)

4.1 Background: Mo's Algorithm & Hilbert MO

Suppose we are given a static array of length N and a list of Q offline queries of the form: *"What is the most frequent element in the subarray $[s_i, e_i]$?"*

A naive approach would compute the answer for each query independently in $O(N)$ time, resulting in $O(NQ)$ total time—clearly inefficient for large Q.

However, if we already know the result for a query $[s, e]$, we can compute the result for neighboring queries such as $[s, e+1]$ or $[s, e-1]$ by simply adding or removing one element at a time—called **push** and **pop** operations. This local update property motivates a more efficient technique: **Mo's algorithm**.

Mo's algorithm processes all queries in a special order that minimizes total push/pop operations. The idea is to divide the array into $\sqrt{N}$ buckets and sort the queries by:

- the bucket index of the left endpoint s (i.e., $\lfloor s/B \rfloor$), and
- the right endpoint e within each bucket.

This ordering guarantees that the total movement of the $[s, e]$ window across all queries is bounded by $O((N+Q)\sqrt{N})$.

Hilbert MO further refines this strategy by choosing a more balanced bucket size: $B = N/\sqrt{Q}$. This minimizes the worst-case pointer movement to $O(N\sqrt{Q})$, which is provably optimal under this model. Although originally derived from ideas related to space-filling curves, Hilbert MO can be interpreted purely in terms of bucket size optimization.

In this project, we adapt these ideas to TSP by treating each node as a "query" and sorting them in a Mo-like order that minimizes geometric movement in the tour.

4.2 MO's Based TSP Heuristic

In the 2D TSP setting with L_1 distance, moving from one node to another is similar in spirit to the Mo's algorithm model: moving a query window left or right along an axis.

Suppose the x-range is $[x_{\min}, x_{\max}]$, we divide it into $\sqrt{x_{\max} - x_{\min}}$ buckets. Each point is associated with a pair (bucket index, y). Then sort the points by this pair to obtain a visiting order.

- Within each bucket, points are sorted in increasing y-coordinate.
- The final tour is constructed by connecting the sorted points sequentially, and the last point is connected back to the first to complete the Hamiltonian cycle.
- We repeat this procedure using the y-axis as the primary coordinate and select the shorter of the two tours.

Time Complexity:

Sorting the n points takes $O(n \log n)$, and tour construction is $O(n)$. Hence the overall complexity is $O(n \log n)$.

Algorithm 3 MO's Based Heuristic

Require: List of nodes with 2D coordinates
Ensure: Approximate TSP tour
1: Compute $x_{\min}, x_{\max}$ and set $B = \sqrt{x_{\max} - x_{\min}}$
2: Sort all nodes by $(\lfloor x/B \rfloor, y)$ in ascending order
3: Construct tour by connecting sorted nodes in order
4: Repeat for y-axis and choose the shorter tour
5:
6: **return** shortest tour

4.3 Hilbert MO Based TSP Heuristic

This heuristic is structurally similar to the MO's based approach, but adopts a more refined bucketing strategy inspired by the optimized variant of Mo's algorithm known as **Hilbert MO**.

Here, the bucket size B is defined as:

$$B = \left\lfloor \frac{N}{\sqrt{Q}} \right\rfloor$$

where:

- N is the range of coordinates along the axis, computed as $x_{\max} - x_{\min}$ (or $y_{\max} - y_{\min}$),
- Q is the total number of nodes.

This setting matches the bucketing strategy used in Hilbert MO, which minimizes total movement between successive queries (or nodes, in our case). The rest of the algorithm follows the same structure as in the MO's based approach.

Time Complexity:

Sorting dominates with $O(n \log n)$ complexity.

Algorithm 4 Hilbert MO Based Heuristic

Require: List of n nodes with 2D coordinates
Ensure: Approximate TSP tour
1: Compute $x_{\min}, x_{\max}$ and set $B = \lfloor (x_{\max} - x_{\min})/\sqrt{n} \rfloor$
2: Sort all nodes by $(\lfloor x/B \rfloor, y)$ in ascending order
3: Construct tour by connecting sorted nodes in order
4: Repeat using y-axis and choose the shorter tour
5:
6: **return** shortest tour

4.4 MST-Based Query Ordering

In previous section, we introduced MST-based algorithm—for approximating the solution to the Traveling Salesman Problem (TSP). Although originally designed for TSP, this algorithm can also be applied to determine the processing order of offline queries by treating each query as a node and computing an approximate tour through them.

Empirical Observation:

Experimental results show that the MST-based ordering significantly outperforms both MO's and Hilbert MO-based approaches in terms of solution quality. In particular, it achieved results close to the optimal TSP solution, demonstrating its practical effectiveness in producing near-optimal query orders.

Trade-Off Consideration:

However, this accuracy comes at a cost. Constructing the MST and deriving the query order requires $O(n^2 \log n)$ preprocessing cost, which can be prohibitively expensive for large n.

Contrast with MO's and Hilbert MO:

By comparison, MO's-based and Hilbert MO-based orderings require only $O(n \log n)$ time to sort queries. These approaches are significantly faster to compute and are well-suited for large-scale problems where preprocessing time must be minimized.

Ultimately, MST-based query ordering presents a trade-off between preprocessing overhead and execution efficiency. When the quality of the query order has a substantial impact on overall system performance, the additional setup cost can be justified.

5 EXPERIMENTS

All programs were implemented in standard C++ and compiled with g++ on Windows 10. All experiments were performed on a machine with an Intel i7-9750H 2.6GHz CPU. The full source codes, datasets, and experimental results are publicly available on GitHub[1].

5.1 Experiment Settings

To evaluate the performance of each algorithm, we run tests on four benchmark datasets: `a280`, `xql662`, `kz9976`, and `mona_lisa`. For each dataset, three test scales are used:

- **20 samples**: a small input to evaluate exact solutions using Held-Karp.
- **10000 samples**: a medium-scale input used specifically to test MST-based scalability. If the original dataset has fewer than 10000 points, it is used directly.
- **Original**: the full dataset to assess large-scale approximation performance.

Each algorithm was evaluated under both L_1 and L_2 norm distances. When available, Held-Karp was used to compute the optimal solution (ground truth) for comparison. In larger datasets, where this is computationally infeasible, MST-based approximations were used as the practical baseline.

[1] https://github.com/dh28be/CSE331_Assignment2

5.2 Experiment Results

Dataset	Algorithm	20 samples	10,000 samples	Original
a280	Ground Truth	–	2820	2820
	Held-Karp	392	–	–
	MST-based	400	4352	–
	MO's	432	6620	6620
	Hilbert MO	432	6620	6620
xql662	Ground Truth	–	3030	3030
	Held-Karp	136	–	–
	MST-based	144	4284	–
	MO's	148	5842	5842
	Hilbert MO	148	6136	6136
kz9976	Ground Truth	–	1350756	1350756
	Held-Karp	2146	–	–
	MST-based	2413	1832798	–
	MO's	3746	47435276	47435276
	Hilbert MO	3013	3129006	3129006
mona_lisa	Ground Truth	–	–	7311476
	Held-Karp	93338	–	–
	MST-based	109330	2409900	–
	MO's	156806	5963554	12100262
	Hilbert MO	153582	4644856	15275984

Table 1: Total tour cost under L_1 norm

Dataset	Algorithm	20 samples	10,000 samples	Original
a280	Held-Karp	3311ms	–	–
	MST-based	0ms	11ms	–
	MO's	0ms	0ms	0ms
	Hilbert MO	0ms	0ms	0ms
xql662	Held-Karp	3277ms	–	–
	MST-based	0ms	68ms	–
	MO's	0ms	0ms	0ms
	Hilbert MO	0ms	0ms	0ms
kz9976	Held-Karp	3235ms	–	–
	MST-based	0ms	22589ms	–
	MO's	0ms	7ms	7ms
	Hilbert MO	0ms	7ms	8ms
mona_lisa	Held-Karp	3233ms	–	–
	MST-based	0ms	21743ms	–
	MO's	0ms	10ms	118ms
	Hilbert MO	0ms	9ms	116ms

Table 2: Execution time under L_1 norm

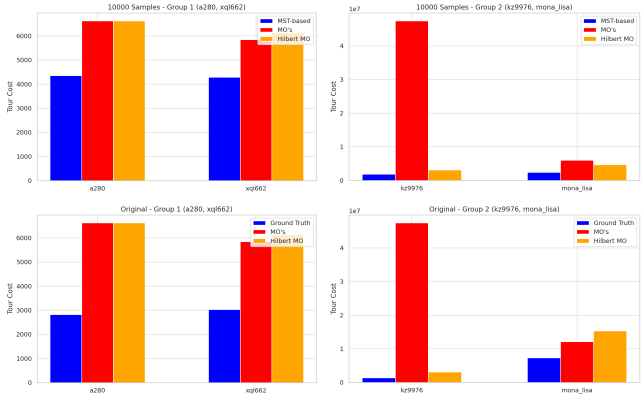

Figure 1: Performance comparison under L_1 norm. Top: 10,000-sample inputs. Bottom: original datasets. Left: small instances, Right: large instances.

Dataset	Algorithm	20 samples	10,000 samples	Original
a280	Ground Truth	–	2586.77	2586.77
	Held-Karp	332.11	–	–
	MST-based	356.16	3798.31	–
xql662	Ground Truth	–	2555.56	2555.56
	Held-Karp	132.29	–	–
	MST-based	137.23	3606.01	–
kz9976	Ground Truth	–	1061875.39	1061875.39
	Held-Karp	1701.36	–	–
	MST-based	1778.53	1459852.35	–
mona_lisa	Ground Truth	–	–	5759971.49
	Held-Karp	73624.46	–	–
	MST-based	99062.60	1902253.63	–

Table 3: Total tour cost under L_2 norm

5.3 Analysis

The results demonstrate that the MST-based algorithm consistently achieves the highest solution quality across all datasets and scales, producing total tour lengths closest to the ground truth under both L_1 and L_2 norms.

MO's and Hilbert MO show clear advantages in runtime efficiency, completing in near-zero time regardless of input size. These methods are thus well suited for scenarios where computational efficiency is critical.

Although Hilbert MO is theoretically expected to outperform MO's by reducing the number of positional transitions, experimental results reveal that MO's occasionally performs better in practice. This discrepancy is likely due to differences in point distributions or dataset-specific characteristics.

Held-Karp provides exact optimal solutions but is only practical for small inputs due to its exponential time complexity.

6 CONCLUSION

In this project, we explored both classical and novel approaches to the Traveling Salesman Problem (TSP). We implemented the exact Held-Karp algorithm, a classical MST-based approximation, and two heuristics inspired by MO's and Hilbert MO algorithms.

Empirical evaluations across multiple datasets demonstrated the strengths and limitations of each method. The MST-based approach achieved near-optimal solution quality on both small and large inputs, while MO's and Hilbert MO offered substantial speed advantages with reasonable accuracy. Despite its theoretical strengths, Hilbert MO did not consistently outperform MO's in practice, highlighting the role of dataset characteristics in heuristic performance.

Future work may investigate how approximation algorithms for TSP can be applied to optimize the execution order of offline queries, where minimizing traversal cost directly impacts performance. In particular, the MST-based ordering strategy shows possibility for such applications, as it naturally aligns with minimizing total query movement. Beyond query optimization, these algorithms could also be adapted to domains such as logistics or image processing, where spatial coherence plays a critical role.

References

[1] Gazelle. 2019. [/ TSP] 2- (Metricity 2-Approximation). https://gazelle-and-cs.tistory.com/18. Accessed: 2025-06-06.
[2] M. Held and R. M. Karp. 1962. A Dynamic Programming Approach to Sequencing Problems. *J. Soc. Indust. Appl. Math.* 10, 1 (1962), 196–210.

[3] R. Matai, S. P. Singh, and M. L. Mittal. 2010. Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches. *Traveling Salesman Problem, Theory and Applications* 1, 1 (2010), 1–25.

[4] TamRef. 2018. HilbertMO : alternative query sorting for Mo's algorithm. https://tamref.com/97. Accessed: 2025-06-06.