CSE331 - Assignment #2

Jeonghoon Park (20201118)
UNIST
South Korea
hoonably@unist.ac.kr

Project Page: https://github.com/hoonably/traveling-salesman

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a well-known NP-hard problem in combinatorial optimization. It seeks the shortest tour that visits each city exactly once and returns to the origin, with applications ranging from logistics to circuit design.

To tackle its computational difficulty, many heuristics and approximation algorithms have been developed. The MST-based 2-approximation algorithm provides theoretical guarantees under the triangle inequality, while greedy and local search methods such as 2-opt offer strong empirical performance despite lacking worst-case bounds.

In this project, we implement several classical algorithms—Held-Karp dynamic programming, MST-based approximation, greedy heuristics, and 2-opt refinement—as well as a novel flow-based heuristic. This method leverages minimum-cost maximum-flow (MCMF) to generate cycle covers, which are then refined by 2-opt. We also apply k-nearest-neighbor sparsification to improve scalability.

Although our method is generally slower than classical heuristics, its combination with 2-opt yields comparable tour quality. Sparsification significantly reduces runtime, making the approach more practical for larger instances.

2 PROBLEM STATEMENT

The Traveling Salesman Problem (TSP) asks: given a set of n cities and pairwise costs c_{ij}, find the shortest possible tour that visits each city exactly once and returns to the starting point. Formally, for $V = \{v_1, v_2, \ldots, v_n\}$, the objective is:

$$\min_{\pi \in S_n} \left(c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \right)$$

where S_n is the set of all permutations of n elements.

2.1 Computational Complexity

TSP is a classic **NP-hard** problem. The decision version is **NP-complete**, and the optimization version is **NP-hard** but not known to be in NP. Since the number of feasible tours grows factorially, exact algorithms quickly become infeasible as n increases.

2.2 Approximation Motivation

Several polynomial-time *approximation algorithms* exist for the metric TSP:

- MST-based methods offer a provable 2-approximation.
- Greedy and local search heuristics such as 2-opt work well in practice but have no theoretical guarantee.
- Our proposed flow-based method aims to combine global structure from MCMF with local refinement via 2-opt.

Throughout this report, we restrict our focus to the symmetric metric TSP.

3 EXISTING ALGORITHMS

3.1 Held-Karp (Dynamic Programming)

The Held-Karp algorithm [5] computes the exact solution to the TSP via dynamic programming. It avoids enumerating all $n!$ permutations by maintaining the optimal cost $C(S, j)$ of reaching city j after visiting subset S. The recurrence is:

$$C(S, j) = \min_{k \in S, k \neq j} \left[C(S \setminus \{j\}, k) + c_{kj} \right]$$

with base case $C(\{1, j\}, j) = c_{1j}$, assuming city 1 is the starting point. The optimal cost is:

$$\min_{j \neq 1} \left[C(\{1, \ldots, n\}, j) + c_{j1} \right]$$

Algorithm 1 Held-Karp-TSP

```
for j = 2 to n do
    C[{1, j}][j] ← c_1j
end for
for s = 3 to n do
    for each subset S ⊆ {1, ..., n} of size s containing 1 do
        for j ∈ S \ {1} do
            C[S][j] ← min_{k∈S\{j}} (C[S \ {j}][k] + c_kj)
        end for
    end for
end for
Reconstruct tour T by backtracking through C
return Hamiltonian tour T
```

Time and Space Complexity.

$$\text{Time: } O(n^2 \cdot 2^n), \quad \text{Space: } O(n \cdot 2^n)$$

Benchmark Role. Despite its exponential complexity, Held-Karp is valuable as a gold-standard reference for evaluating the accuracy of heuristic or approximate algorithms. It remains feasible for small instances ($n \lesssim 20$) on modern hardware.

Advantages.

- Computes the exact solution, suitable for small n.
- Exploits subproblem overlap to avoid full enumeration.

Limitations.

- Exponential time and space complexity.
- Infeasible for large instances ($n > 20$); surpassed by solvers like Concorde in practice.

3.2 MST-based 2-Approximation Algorithm

This classical algorithm [1] constructs a tour with cost at most twice the optimal, assuming the triangle inequality. It builds a minimum spanning tree (MST), performs a preorder traversal to list the nodes, and shortcuts repeated visits to yield a Hamiltonian tour.

Algorithm 2 MST-Based-TSP

```
Choose start node r
Compute MST T rooted at r (e.g., Prim's algorithm)
Perform preorder traversal on T to obtain path P
Shortcut repeated nodes in P to construct Hamiltonian tour T
return tour T
```

Time and Space Complexity.

$$\text{Time: } O(n^2), \quad \text{Space: } O(n)$$

Approximation Guarantee.

Let H^* be the optimal tour and T the MST. Then:

- $c(T) \leq c(H^*)$, as removing one edge from H^* yields a spanning tree.
- A DFS traversal visits each MST edge twice: $c(W) = 2c(T) \leq 2c(H^*)$.
- Shortcutting repeated nodes using triangle inequality yields tour H with:

$$c(H) \leq c(W) \leq 2c(H^*)$$

Thus, the tour cost is at most twice optimal. This is a formal 2-approximation for metric TSP.

Advantages.

- Runs in polynomial time; easy to implement.
- Guarantees a 2-approximation if triangle inequality holds.

Limitations.

- Tight worst-case bound; actual cost can approach $2c(H^*)$.
- Fails without triangle inequality.

3.3 Greedy Nearest-Neighbor Heuristic

This heuristic, originally introduced by Flood [3], builds a tour by repeatedly visiting the closest unvisited city. Starting from an initial node, it adds the nearest neighbor to the tour until all cities are visited, then returns to the starting city to complete the tour.

Algorithm 3 Greedy-TSP

```
Initialize tour ← [v_0], visited ← {v_0}
while some cities remain unvisited do
    Let u be the nearest unvisited neighbor of last node in tour
    Append u to tour, mark u as visited
end while
Append v_0 to close the tour
return tour T
```

Time and Space Complexity.

$$\text{Time: } O(n^2), \quad \text{Space: } O(n)$$

Approximation Behavior. Greedy has no worst-case guarantee; its tour can be arbitrarily worse than optimal. However, it performs well on Euclidean or clustered data and often surpasses MST-based tours in practice. Recent theoretical analysis by Frieze and Pegden [4] further supports this behavior, showing that the average-case performance of Greedy is significantly better than worst-case expectations.

Advantages.

- Simple and fast; needs no preprocessing.
- Often gives good results on structured or metric inputs.

Limitations.

- No provable approximation ratio.
- Sensitive to start city and early decisions.

3.4 2-Opt Local Optimization

2-opt, introduced by Croes [2], is a local search method that improves an existing tour by removing two edges and reconnecting the segments in a different order if the total cost decreases. The process repeats until no improving swap exists. Though not guaranteed to find the global optimum, it is highly effective in practice and is applied after all construction heuristics in our pipeline.

Algorithm 4 Two-Opt

```
while any 2-swap improves cost do
    Check all pairs of non-adjacent edges for cost-reducing swaps
    if a swap improves the tour then
        Apply the swap
    end if
end while
return improved tour T
```

Time and Space Complexity.

Let k be the number of improving passes. A full pass over all $O(n^2)$ edge pairs takes $O(n^2)$, giving:

$$\text{Time: } O(kn^2), \quad \text{Space: } O(n)$$

Empirically, $k = O(1)$ for structured tours (e.g., Greedy), but up to $k = O(n)$ for unstructured inputs (e.g., Random), yielding total runtime from $O(n^2)$ to $O(n^3)$.

Impact and Use Case. 2-opt significantly enhances solution quality but depends heavily on the initial tour. A well-structured input accelerates convergence and yields near-optimal results. For random inputs, 2-opt still improves quality but at a steep computational cost. This highlights its role as a *refinement* tool rather than a standalone solver.

Advantages.

- Simple, parameter-free, and broadly applicable.
- Consistently improves heuristic solutions.

Limitations.

- May converge to local optima.
- Cubic runtime if applied to unstructured tours.
- Requires an initial feasible tour.

4 PROPOSED ALGORITHM

4.1 Flow-based Cycle Cover Heuristic (MCMF)

This heuristic constructs an initial solution by formulating the TSP as a cycle cover problem using Minimum-Cost Maximum-Flow (MCMF). The goal is to find a set of disjoint cycles covering all nodes such that the total edge cost is minimized. These cycles are then merged into a single tour based on the minimum inter-cycle endpoint distances.

Construction Overview. We build a bipartite flow graph with two copies of the city set. Each node in the left set L is matched to a node in the right set R, ensuring a one-to-one assignment. Edges are assigned capacity 1 and cost equal to inter-city distances.

Algorithm 5 Flow-Cycle-Cover-TSP

1: Construct bipartite graph with source s, sink t, and city sets L, R
2: Connect $s \to L_i$ and $R_j \to t$ with capacity 1 and cost 0
3: Connect $L_i \to R_j$ with capacity 1 and cost c_{ij} for all $i \neq j$
4: Run MCMF from s to t to obtain flow-based matching
5: Extract subtours from flow result
6: **while** more than one subtour remains **do**
7: Merge the two closest subtours
8: **end while**
9: **return** merged tour T

Computational Complexity. Let n be the number of cities. The graph has $O(n^2)$ edges, and the MCMF requires $O(n)$ augmentations. Using SPFA:

$$\text{Time: } O(n^3), \quad \text{Space: } O(n^2)$$

4.2 Scalability via kNN Sparsification

To improve scalability, we restrict the edge set to the k-nearest neighbors per node, forming a sparse graph with $O(kn)$ edges. This reduces MCMF runtime significantly while preserving solution quality.

Complexity in the Sparsified Case.

$$\text{Time: } O(kn^2), \quad \text{Space: } O(kn)$$

Empirical Observation. Ablation studies show that with moderate k (e.g., 20), the degradation in tour length is minimal while runtime is substantially reduced. This enables the algorithm to scale to larger TSP instances.

4.3 Local Refinement via 2-Opt

While the flow-based cycle cover provides a globally informed starting tour, the greedy merging step can result in inefficient connections. To refine the result, we apply 2-opt as a local post-processing technique.

Mechanism. 2-opt eliminates edge crossings and improves local structure by iteratively swapping two edges if the swap shortens the tour. Its performance depends strongly on the quality of the initial tour.

Interaction with MCMF Initialization. The globally coherent structure from MCMF allows 2-opt to converge in a small, constant number of passes, resulting in $O(n^2)$ runtime. By avoiding long or crisscrossing edges, MCMF reduces the number of costly edge swaps.

4.4 Combined Effect

The proposed hybrid algorithm uses an MCMF-based cycle cover for initialization and 2-opt for local refinement. This combination yields high-quality tours while maintaining a reasonable runtime—especially when sparsification is applied. Although slower to construct than Greedy-based methods, the final performance is comparable, demonstrating that global initialization paired with local improvement is a viable strategy for solving TSP efficiently.

5 EXPERIMENTS

5.1 Experimental Setup

All experiments were conducted on a MacMini (Apple M4, 16GB). Compiled with `clang++` (`-std=c++17`, `-O2`). Tested on five TSP datasets with EUC_2D metric:

- **weird20.tsp** (20 cities)
- **a280.tsp** (280 cities)[1]
- **xql662.tsp** (662 cities)[2]
- **kz9976.tsp** (9,976 cities)[3]
- **mona_lisa100K.tsp** (100,000 cities)[4]

For $n < 10,000$, full pairwise distances were precomputed; larger cases used on-the-fly distance computation.

5.2 Runtime Comparison

As summarized in Table 1, Greedy runs fastest due to simple nearest-neighbor logic. MST and Flow-based incur higher costs due to traversal and flow computation. 2-opt increases runtime proportionally to the number of iterations.

5.3 Solution Quality

Table 1 also shows that MST, despite its 2-approximation guarantee, consistently produced longer tours than Greedy across all instances. Even after 2-opt refinement, it remained less competitive.

Greedy, though lacking any formal guarantee, achieved the best standalone performance and remained competitive after 2-opt. This suggests that its local proximity-based structure—especially effective when cities are spatially clustered—provides a strong foundation for local optimization.

The Flow-based heuristic underperformed on its own, mainly due to many short subtours—particularly 2-cycles—that lead to fragmented tours. However, once 2-opt is applied, the global structure from MCMF guides effective refinement. As a result, Flow + 2opt consistently outperformed MST + 2opt across datasets.

In short, Greedy benefits from local coherence, Flow-based methods from global structure, while MST—constrained by rigid traversal—offers neither and responds poorly to refinement.

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/
[2] https://www.math.uwaterloo.ca/tsp/vlsi/index.html
[3] https://www.math.uwaterloo.ca/tsp/world/countries.html
[4] https://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html

Table 1: Comparison between base and +2opt variants across datasets.

		Base Tour			+2-opt Applied			
Dataset	**Opt Length**	**Algorithm**	**Length**	**Time (s)**	**Algorithm**	**Length**	**Time (s)**	**2opt Iters**
a280	2579	Random	33736	-	Random (+2opt)	2774	0.022929	1368
		Greedy	3157	0.000144	Greedy (+2opt)	2767	0.002989	57
		MST	3492	0.000271	MST (+2opt)	2908	0.004490	80
		Flow	3417	0.016223	Flow (+2opt)	2705	0.019008	66
		Flow_kNN	3348	0.006696	Flow_kNN (+2opt)	2696	0.011764	82
xql662	2513	Random	53168	-	Random (+2opt)	2762	0.277021	3945
		Greedy	3124	0.000812	Greedy (+2opt)	2693	0.031972	116
		MST	3593	0.001196	MST (+2opt)	2763	0.039341	237
		Flow	3862	0.064118	Flow (+2opt)	2719	0.093078	267
		Flow_kNN	3931	0.034103	Flow_kNN (+2opt)	2737	0.069999	301
kz9976	1061882	Random	133724845	-	Random (+2opt)	1154441	3582.804296	119612
		Greedy	1358249	0.061593	Greedy (+2opt)	1141502	146.796040	3340
		MST	1456572	0.127581	MST (+2opt)	1162397	171.800400	4638
		Flow	1707487	210.406593	Flow (+2opt)	1138579	537.880652	5619
		Flow_kNN	1719092	21.786337	Flow_kNN (+2opt)	1146693	318.389075	6231

5.4 Ablation: Flow-based Cycle Cover Variants

We evaluate four variants of the Flow-based heuristic across all datasets, as shown in Table 1. **Flow**, **Flow + 2opt**, **Flow_kNN**, and **Flow_kNN + 2opt**. To avoid visual clutter, only the results on `kz9976` are highlighted in yellow in Table 1.

Results show that 2-opt significantly improves tour length regardless of whether the full or k-sparsified graph is used. Although Flow alone performs poorly due to many short subtours, the global structure it produces enables efficient refinement. In particular, **Flow + 2opt** outperforms **MST + 2opt** consistently.

Sparsification further reduces runtime with minimal degradation in solution quality. Since both 2-opt and Flow_kNN run in $O(n^2)$ time, combining them preserves quadratic complexity—matching classical heuristics in speed while improving output quality.

5.5 Additional Results

To evaluate performance at both extremes of problem scale, we present results for two special cases: a tiny 20-city instance and a massive 100K-city instance.

For the small-scale test, we generated 20 random TSP samples with 20 cities and applied the Held-Karp algorithm to compute exact solutions. On average, each instance required approximately 43 seconds to solve, demonstrating the feasibility of exact methods at small scale but also highlighting their exponential cost growth.

On the large-scale `mona_lisa100K` instance, 2-opt refinement was omitted due to memory and runtime constraints. Among the heuristics tested, Greedy produced a reasonable tour in under 20 seconds. In contrast, MST resulted in a significantly longer tour, suggesting that its rigid traversal strategy scales poorly due to an increased risk of long-range connections. Flow-based algorithms were not tested on this instance because of their high computational overhead, even under sparsification.

[5]https://www.math.uwaterloo.ca/tsp/data/ml/tour/monalisa_5757191.tour

Table 2: Exact solutions on small instances and heuristic results on large-scale TSP

Dataset	Algorithm	Length	Time (s)
weird20	Held-Karp	439	43.25
mona_lisa100K	Best-known Result[5]	5757191	-
	Greedy	6846598	16.99
	MST	8394831	32.03

6 CONCLUSION

This report presented and compared classical and heuristic algorithms for the symmetric metric TSP, including Held-Karp, MST-based 2-approximation, Greedy nearest-neighbor, and 2-opt local optimization. We also proposed a novel **Flow-based Cycle Cover Heuristic** using Minimum-Cost Maximum-Flow (MCMF), enhanced with k-NN sparsification and 2-opt refinement.

Among the baseline methods, Greedy achieved the best trade-off among runtime efficiency and tour quality. In contrast, MST-based solutions, though theoretically sound, showed inferior empirical results due to their rigid traversal structure and poor adaptability to local improvement. Flow-based methods initially performed worse due to fragmented subtours, but combining them with 2-opt led to significant improvements.

Since 2-opt is applied for performance improvement anyway, which incurs an $O(n^2)$ runtime, applying k-NN sparsification reduces the Flow-based initialization to a comparable complexity. As a result, the initialization step does not impose additional overhead, and the overall pipeline remains both scalable and effective. This demonstrates that integrating global and local strategies yields a competitive and reliable solution for large-scale TSP problems.

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). MIT Press.
[2] G. A. Croes. 1958. A Method for Solving Traveling-Salesman Problems. *Operations Research* 6, 6 (1958), 791–812.

[3] Merrill M. Flood. 1956. The Traveling-Salesman Problem. *Operations Research* 4, 1 (1956), 61–75.
[4] Alan Frieze and Wesley Pegden. 2024. The Bright Side of Simple Heuristics for the TSP. *arXiv preprint arXiv:2309.12345* (2024).
[5] Michael Held and Richard M Karp. 1962. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.* 10, 1 (1962), 196–210.