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 address its computational intractability, a wide range of 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 including 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

Due to the problem's intractability, various approximation algorithms have been proposed for the metric TSP. MST-based methods offer theoretical guarantees, while Greedy and 2-opt heuristics perform well in practice. Our method builds on these ideas by combining global flow structure with local refinement.

3 EXISTING ALGORITHMS

3.1 Held-Karp (Dynamic Programming)

The Held-Karp algorithm [6] computes the exact TSP solution via dynamic programming by storing the minimal cost $C(S, j)$ of reaching city j through subset S. It avoids enumerating all $n!$ permutations by using the recurrence $C(S, j) = \min_{k \in S \setminus \{j\}} [C(S \setminus \{j\}, k) + c_{kj}]$, with base case $C(\{1, j\}, j) = c_{1j}$. Assuming city 1 is the start, the optimal tour cost is $\min_{j \neq 1} [C(\{1, \ldots, n\}, j) + c_{j1}]$.

Algorithm 1 Held-Karp-TSP

for $j = 2$ to n **do**
  $C[\{1, j\}][j] \leftarrow c_{1j}$
end for
for $s = 3$ to n **do**
  for each subset $S \subseteq \{1, \ldots, n\}$ of size s containing 1 **do**
    for $j \in S \setminus \{1\}$ **do**
      $C[S][j] \leftarrow \min_{k \in S \setminus \{j\}} \left(C[S \setminus \{j\}][k] + c_{kj} \right)$
    end for
  end for
end for
Reconstruct tour T by backtracking through C
return Hamiltonian tour T

Time and Space Complexity.

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

Advantages. It provides the exact optimal solution for small instances.

Limitations. Its exponential time and space complexity makes it infeasible beyond $n > 30$, and it is generally superseded by solvers like Concorde.

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. It is simple to implement and offers a provable 2-approximation guarantee under the triangle inequality.

Limitations. It can produce tours nearly twice as long as optimal in the worst case, and the guarantee fails if the triangle inequality does not hold.

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. Despite its simplicity, Greedy offers no worst-case or approximation guarantee, and may produce tours arbitrarily worse than optimal. However, it performs well on Euclidean or clustered data and often performs better than 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. It is extremely fast, easy to implement, and performs well on Euclidean or spatially clustered inputs.

Limitations. It may still perform poorly in adversarial or non-metric instances due to the lack of global planning.

3.4 2-Opt Local Optimization

2-opt [2] improves a tour by iteratively swapping two edges to reduce total cost. It is commonly used for post-processing heuristic tours.

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.

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

Empirically, $k = O(1)$ for structured tours and $O(n)$ for random ones, yielding time between $O(n^2)$ and $O(n^3)$.

Advantages. It is an effective local search heuristic that consistently improves tour quality.

Limitations. It may converge to local optima and can be slow when applied to poorly constructed initial tours.

4 PROPOSED ALGORITHM

Existing heuristics like Greedy and MST tend to focus on local optimization and may miss global structure. We adopt MCMF to better capture global cost patterns by generating a one-to-one matching across the entire set of cities. To compensate for potential loss of local structure, we enhance the solution with Greedy-like subtour merging and 2-opt refinement.

4.1 Flow-based Cycle Cover via MCMF

The core idea is to model the TSP as a cycle cover problem. We construct a bipartite flow graph with two copies of the city set and apply MCMF to find a minimum-cost one-to-one assignment between cities. This produces a set of disjoint cycles covering all nodes.

We then iteratively merge cycles into a single tour by connecting the closest pair of endpoints across subtours.

Algorithm 5 Flow-Cycle-Cover-TSP

```
Construct bipartite graph with source s, sink t, and city sets L, R
Connect s → L_i and R_j → t with capacity 1 and cost 0
Connect L_i → R_j with capacity 1 and cost c_ij for all i ≠ j
Run MCMF from s to t to obtain flow-based matching
Extract subtours from flow result
while more than one subtour remains do
    Merge the two closest subtours
end while
return merged tour T
```

Time and Space Complexity. Using SPFA [5], a queue-based heuristic improvement over Bellman-Ford, the total complexity becomes:

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

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

			Base Tour			+2-opt Applied			
Dataset	**Opt**	**Algorithm**	**Length**	**Approx**	**Time (s)**	**Length**	**Approx**	**Time (s)**	**2opt Iters**
a280	2579	Random	33736	13.0741	-	2774	1.0756	0.022929	1368
		Greedy	3157	1.2244	0.000144	2767	1.0729	0.002989	57
		MST	3492	1.3540	0.000271	2908	1.1276	0.004490	80
		Flow	3417	1.3251	0.016223	2705	1.0489	0.019008	66
		Flow_kNN	3348	1.2979	0.006696	2696	1.0453	0.011764	82
xql662	2513	Random	53168	21.1507	-	2762	1.0989	0.277021	3945
		Greedy	3124	1.2430	0.000812	2693	1.0716	0.031972	116
		MST	3593	1.4299	0.001196	2763	1.0996	0.039341	237
		Flow	3862	1.5373	0.064118	2719	1.0819	0.093078	267
		Flow_kNN	3931	1.5640	0.034103	2737	1.0893	0.069999	301
kz9976	1061882	Random	133724845	125.9204	-	1154441	1.0868	3582.804296	119612
		Greedy	1358249	1.2790	0.061593	1141502	1.0752	146.796040	3340
		MST	1456572	1.3719	0.127581	1162397	1.0947	171.800400	4638
		Flow	1707487	1.6081	210.406593	1138579	1.0731	537.880652	5619
		Flow_kNN	1719092	1.6193	21.786337	1146693	1.0799	318.389075	6231

4.2 kNN Sparsification for Scalability

To reduce computational cost, we construct a sparse version of the bipartite graph by retaining only the k-nearest neighbors per node. This preserves much of the MCMF structure while significantly improving scalability.

Time and Space Complexity.

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

In our implementation, we set $k = 20$ to balance sparsity and quality while preserving local structure for effective MCMF.

4.3 Refinement with 2-opt

The merged tour from MCMF often contains long or suboptimal edges due to greedy merging. To improve the result, we apply 2-opt as a post-processing step. While 2-opt is local in nature, it effectively eliminates crossings and shortens long edges.

This combination of global structure from MCMF and local refinement from 2-opt achieves superior solution quality with moderate additional cost.

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]

[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

5.2 Runtime Comparison

As shown in Table 1, runtime mainly depends on the cost of initial tour construction and 2-opt refinement.

Greedy is the fastest, completing in milliseconds due to its simple nearest-neighbor rule. **MST** is slightly slower but remains efficient. **Flow-based** methods are much more expensive; even with kNN sparsification, they require seconds to minutes depending on instance size due to the cost of solving a minimum-cost flow problem. Full Flow becomes impractical for large instances. The cost of **2-opt** varies by initialization: Greedy and MST need fewer refinements, while Flow-based methods often result in longer refinement time due to less structured outputs.

5.3 Solution Quality

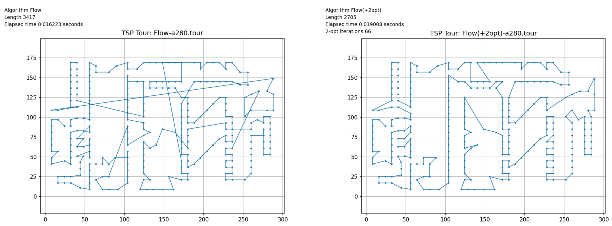
Before refinement, **Greedy** consistently outperforms other structured heuristics, achieving the lowest approximation ratios among base tours. Despite lacking theoretical guarantees, it benefits from strong local coherence, especially in spatially clustered instances. **MST**, while offering a 2-approximation bound, often results in longer tours due to detours inherent in its tree-based construction. Although 2-opt improves it to some extent, suboptimal edges often remain. **Flow-based** methods show relatively poor performance before refinement, primarily due to short subtours with many repeated cycles. However, after applying 2-opt, both Flow and Flow_kNN exhibit the largest improvements among all algorithms. Post-refinement, their solution performs better than MST and approaches that of Greedy. This large improvement is primarily due to the poor quality of their initial solutions, which leave more room for optimization.

The initial solution produced by MCMF often contains many short subtours, particularly 2-node cycles. This fragmentation weakens the intended global matching effect, forcing the algorithm to rely heavily on 2-opt for recovery, which in turn increases iteration counts and overall runtime.

5.4 Ablation: Flow-based Cycle Cover Variants

Table 1 highlights how *k*NN sparsification reduces the number of edges processed during MCMF, achieving a clear runtime benefit. As a result, **Flow_kNN** scales better than the full Flow method, particularly on large datasets like `kz9976` and `mona_lisa100K`.

Figure 1: Effect of 2-opt on the Flow solution for `a280`.



(a) Flow - a280 **(b) Flow (+2opt) - a280**

With Flow alone, the resulting tour may contain excessively long edges when connecting cities. In such cases, **2-opt** eliminates these long edges by swapping segments between cities, leading to a more efficient and compact route. This improvement is clearly visible in Figure 1, where the post-refinement tour for `a280` is significantly cleaner and shorter.

Both **kNN sparsification** and **2-opt** operate in $O(n^2)$ time, so their combination offers strong solution quality without increasing overall complexity. We thus find that **Flow_kNN + 2opt** is a practical and scalable hybrid approach.

5.5 Additional Results

To evaluate performance at scale extremes, we include results for a small 20-city instance and a massive 100,000-city instance (Table 2).

Table 2: Held-Karp and large-scale results.

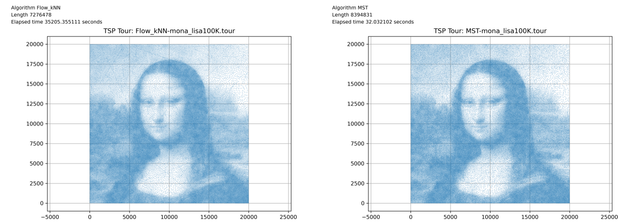
Dataset	Algorithm	Length	Time (s)
weird20	Held-Karp	439	43.25
mona_lisa100K	Best-known Result[5]	5757191	-
	Greedy	6846598	16.99
	MST	8394831	32.03
	Flow_kNN	7276478	35205.36

For `weird20`, we applied the Held-Karp dynamic programming algorithm, which guarantees the exact optimal tour. Although it becomes infeasible beyond 30 cities due to exponential runtime, it completes in 43 seconds for this case and serves as a reference for absolute optimality.

On `mona_lisa100K`, the best-known solution has length 5,757,191[6]. Among our tested methods, **Greedy** finishes fastest (17s) with a reasonable result. **MST** requires slightly more time but yields the longest tour due to inefficient tree detours. **Flow_kNN** offers improved quality over MST, though with significantly higher computational cost.

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

Figure 2: Visual comparison of Flow_kNN and MST tours.



(a) Flow_kNN - mona_lisa100K **(b) MST - mona_lisa100K**

Figure 2 illustrates these differences. Flow_kNN produces a few long edges due to early subtour merging, but most connections are short and contribute to a lower total cost. MST shows more uniform spacing but suffers from accumulated detour. This is a reversal from smaller instances, where MST typically outperforms Flow. The results suggest that global initialization becomes more effective at scale, making Flow-based methods viable even without refinement.

6 CONCLUSION

Summary of Findings. This report studied classical and heuristic algorithms for the symmetric TSP, including Held-Karp, MST-based 2-approximation, Greedy nearest-neighbor, and 2-opt local search. We also introduced a novel Flow-based Cycle Cover heuristic, leveraging Minimum-Cost Maximum-Flow (MCMF), enhanced with *k*-nearest-neighbor sparsification and 2-opt refinement.
Performance Analysis. Among traditional methods, Greedy achieved the best trade-off between runtime and solution quality, outperforming MST in most practical scenarios despite lacking theoretical guarantees. While the Flow-based method incorporates global structure through MCMF, its effectiveness is often diminished by the formation of short subtours, resulting in poor initial solutions and increased reliance on 2-opt refinement.
Limitation and Future Work. The flow-based method fails to fully exploit the global optimality potential of MCMF due to the prevalence of short subtours, which degrade the initial solution quality and necessitate substantial local refinement. A promising direction is to recursively apply MCMF by treating each subtour as a supernode and constructing a higher-level tour among them. However, defining consistent inter-subtour distances is non-trivial, as naive metrics may introduce ambiguity in how cycles should be merged. Addressing this challenge could enable full tour construction purely through recursive MCMF, leading to a fully global, flow-based approximation of the TSP.

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] Andrew V Goldberg and Tomasz Radzik. 1993. A heuristic improvement of the Bellman-Ford algorithm. *Applied Mathematics Letters* 6, 3 (1993), 3–6.

[6] Michael Held and Richard M Karp. 1962. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.* 10, 1 (1962), 196–210.