Approximation and Heuristic Approaches for the Travelling Salesperson Problem (TSP)

Anuj Bohra (ab3414)
Krisha Borana (kb1469)
CS 512 – Final Project

Abstract

The Travelling Salesperson Problem (TSP) is a classical NP-hard problem that seeks the shortest route visiting all cities exactly once and returning to the starting point. While exact algorithms guarantee optimality, their factorial complexity makes them impractical for large instances. This project implements, evaluates, and compares three widely studied approaches to solving TSP: Brute Force (optimal), Nearest Neighbor heuristic, and 2-Opt local search. Each algorithm is analyzed in terms of computational cost, solution quality, approximation ratio, and scalability. Experimental results demonstrate the trade-off between optimality and runtime, highlighting the usefulness of heuristics for larger problem sizes.

1 Introduction

The Travelling Salesperson Problem (TSP) is one of the most well-known combinatorial optimization problems. Given n cities and pairwise distances, the task is to compute the shortest possible tour that visits each city once and returns to the origin. TSP has applications in logistics, network routing, genome sequencing, circuit design, and robotics.

TSP is NP-hard in its optimization form and NP-complete in its decision form, implying that no polynomial-time algorithm is known for solving arbitrary instances optimally. As a result, real-world implementations rely heavily on approximation algorithms and heuristics. This project explores the trade-offs between solution quality and computation time across three classes of algorithms: exact, greedy, and local search.

2 Problem Description

Formally, given a complete weighted graph $G = (V, E)$ with $|V| = n$ cities and distance function $d(u, v)$, the objective is to find a permutation π of the cities that minimizes:

$$\text{Cost}(\pi) = \sum_{i=1}^{n} d(\pi_i, \pi_{i+1}) \quad \text{where } \pi_{n+1} = \pi_1.$$

The number of possible tours grows as $(n-1)!/2$, making brute-force enumeration infeasible for all but the smallest inputs.

3 Algorithms Implemented

3.1 Brute Force (Exact Solver)

Brute Force enumerates all possible tours and selects the one with minimum length. It provides ground truth for small instances ($n \leq 10$).

Algorithm 1 Brute Force TSP

```
Fix starting city as 0
for each permutation p of the remaining n − 1 cities do
    Form tour [0, p_1, p_2, ..., p_{n−1}]
    Compute total tour length
    Update best tour and length if improved
end for
return best tour
```

Time complexity: $O(n!)$ Space complexity: $O(n)$

3.2 Nearest Neighbor Heuristic

A greedy approximation that builds a tour incrementally by repeatedly visiting the closest unvisited city.

Algorithm 2 Nearest Neighbor

```
Choose a starting city s
Mark all cities unvisited except s
Initialize tour = [s]
while unvisited cities remain do
    Select city u minimizing d(current, u)
    Append u to tour and mark visited
end while
return tour
```

Time complexity: $O(n^2)$ Space complexity: $O(n)$

3.3 2-Opt Local Search

2-Opt improves an existing tour by removing crossings. It repeatedly swaps two edges if the new configuration reduces total tour length.

Algorithm 3 2-Opt Local Search

```
Start with initial tour (Nearest Neighbor)
improved = true
while improved do
    improved = false
    for i = 1 to n − 2 do
        for k = i + 1 to n − 1 do
            newTour = 2OptSwap(tour, i, k)
            if Length(newTour) ¡ Length(tour) then
                tour = newTour
                improved = true
                break
            end if
        end for
    end for
end while
return tour
```

Time complexity: approximately $O(n^3)$ Space complexity: $O(n)$

4 Data Generation and Processing

Random TSP instances were generated by sampling 2D coordinates for each city:

$$x_i, y_i \sim U(0, 100).$$

A full distance matrix was computed using Euclidean distance:

$$d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

For each value of n, multiple trials were generated and evaluated. For small values ($n = 6, 8, 10$), optimal solutions were computed. For larger values ($n = 20, 50, 100, 200$), only heuristics were run.

Results were stored in CSV format and included:

- tour length
- algorithm runtime
- approximation ratio (heuristic/optimal)

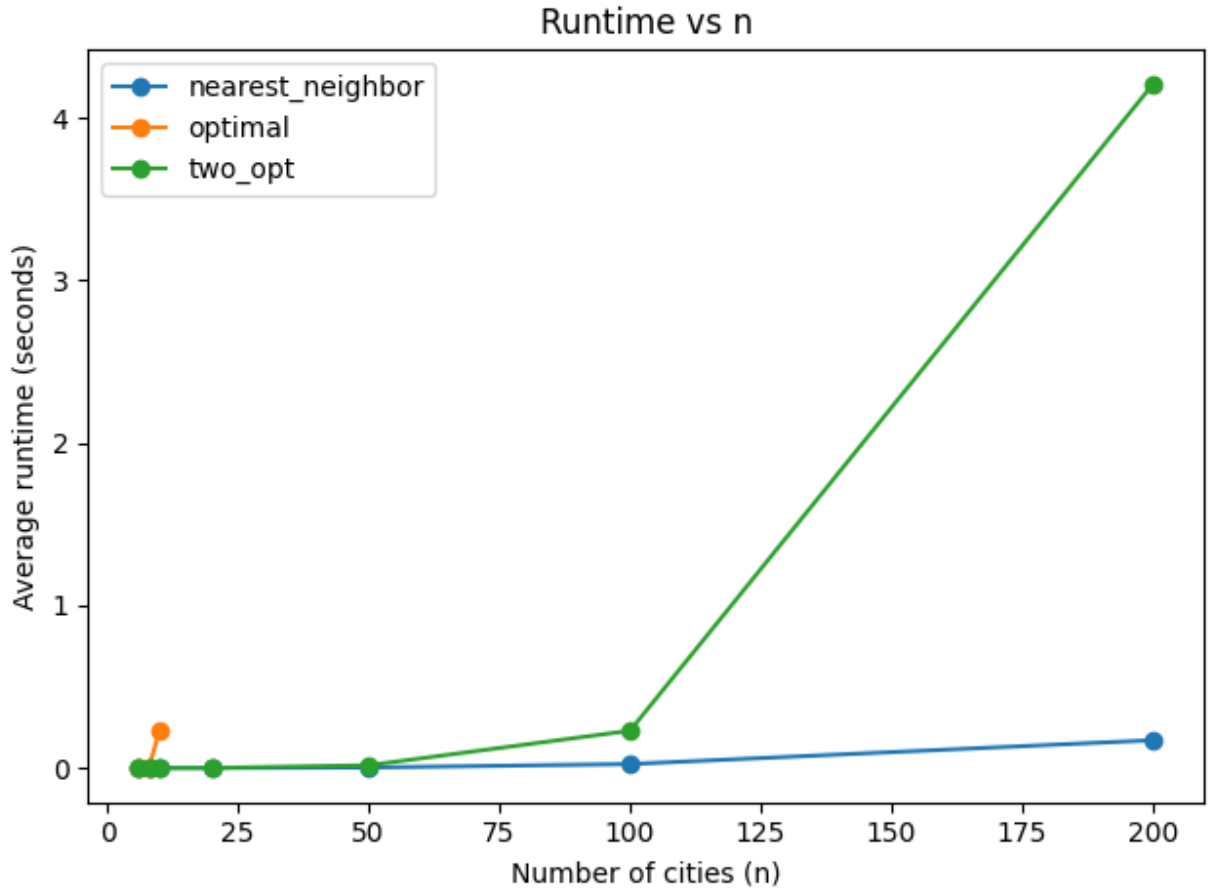
Visualizations of tours and performance plots were automatically generated by the experiment pipeline.

5 Time and Space Complexity

Algorithm	Time Complexity	Space Complexity
Brute Force	$O(n!)$	$O(n)$
Nearest Neighbor	$O(n^2)$	$O(n)$
2-Opt	$O(n^3)$	$O(n)$

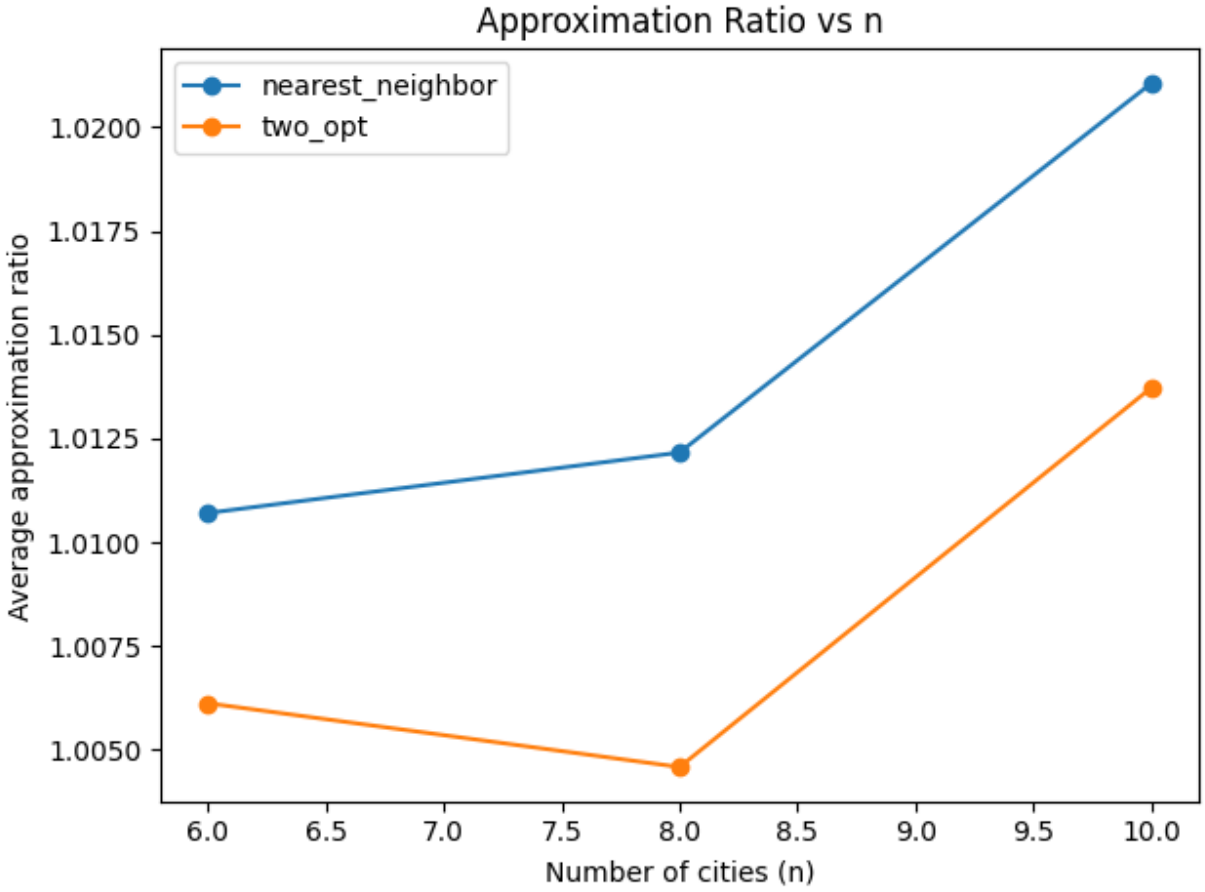
6 Experimental Results

6.1 Runtime vs Number of Cities



The runtime plot illustrates the scalability differences between the three algorithms. As expected, the brute-force optimal solver becomes impractical beyond very small instances due to its factorial time complexity. Nearest Neighbor runs extremely fast and scales smoothly even up to 200 cities, reflecting its quadratic behavior. In contrast, 2-Opt begins similarly fast for small n, but its runtime grows significantly as the number of cities increases, especially beyond 50, due to its cubic complexity. Overall, the figure highlights the trade-off between computational cost and algorithmic sophistication.

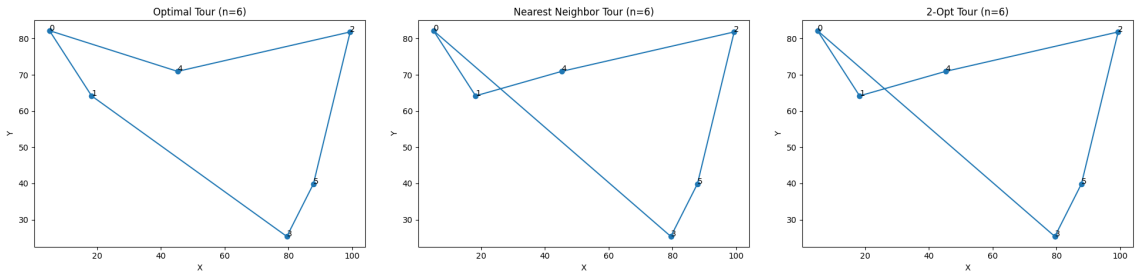
6.2 Approximation Ratio vs Number of Cities



The approximation ratio plot compares heuristic solution quality with the optimal tour for small values of n. Nearest Neighbor consistently produces tours within about 1–2% of optimal, while 2-Opt further improves these solutions, achieving ratios closer to 1.0. This demonstrates that 2-Opt effectively refines the greedy solution by removing inefficient edge crossings. Both heuristics perform well on random Euclidean TSP instances, but 2-Opt clearly achieves higher accuracy at the cost of increased runtime.

6.3 Tour Visualizations

Below are example tours for small instances (Optimal, Nearest Neighbor, 2-Opt):



The tour visualizations for a small instance (n=6) highlight the qualitative differences between the three algorithms. The optimal tour shows the shortest possible route with no unnecessary detours. The Nearest Neighbor tour closely resembles the optimal path but includes a slightly longer segment caused by the greedy choice of visiting the nearest city at each step. The 2-Opt tour demonstrates how local search improves the greedy solution: by reversing segments and eliminating edge crossings, it produces a cleaner, shorter route that more closely matches the optimal tour. These examples visually illustrate how heuristics progressively approach optimality while maintaining computational efficiency.

7 Discussion

The experimental results clearly illustrate the computational and qualitative differences between the three algorithms. As expected, the Brute Force method becomes infeasible beyond $n = 10$, with runtime increasing factorially. Nearest Neighbor consistently ran the fastest, with runtime growing slowly even up to $n = 200$, confirming its quadratic time complexity. In contrast, 2-Opt required substantially more time as n increased, especially beyond 50 cities, due to its cubic behavior. In terms of solution quality, both heuristics performed well on small Euclidean instances. For $n = 6$ to 10, Nearest Neighbor achieved approximation ratios between 1.01 and 1.02, while 2-Opt improved these solutions further, reaching ratios as low as 1.005. The tour visualizations reinforce these findings: Nearest Neighbor produces a reasonable but sometimes inefficient route, while 2-Opt effectively removes edge crossings and produces a cleaner, shorter tour that closely resembles the optimal solution. For larger values of n, where optimal solutions are unavailable, 2-Opt consistently reduced tour length relative to Nearest Neighbor, demonstrating its value as a refinement step despite its higher runtime.

8 Conclusion

This project highlights the trade-offs between optimality and computational feasibility when solving the Travelling Salesperson Problem. While Brute Force guarantees the optimal tour, its factorial complexity limits it to very small instances. Nearest Neighbor provides extremely fast approximations and scales efficiently to hundreds of cities, making it suitable for real-time applications. However, its greedy nature can lead to suboptimal choices. The 2-Opt local search heuristic balances this trade-off effectively: it significantly improves tour quality by eliminating inefficient edge crossings while remaining computationally feasible for moderately large instances. Overall, the results show that heuristic and local search methods offer practical and scalable alternatives to exact algorithms, delivering high-quality solutions for real-world problem sizes where optimal methods are computationally infeasible.