Project Round 2 (TSP)

Kristian Fatohi, Razvan Vass, Safa Orhan, Samer Bassam Salim Jameel

December 23, 2024

Kattis Submission: 15025552 (score: 28.231696, Aiming for grade D)

Problem Overview

Given a set of N points in a 2D plane, represented by coordinates (x, y), the goal is to find a tour that visits each point exactly once and returns to the starting point. The problem constraints are:

- $1 \leq N \leq 1000$
- The coordinates (x, y) have an absolute value bounded by 10^6.

The distance between any two points (x_1, y_1) and (x_2, y_2) is calculated using the Euclidean distance formula:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This distance is rounded to the nearest integer.

1 Failed Attempts

Throughout our development process of achieving a fast approach to solving the Travelling Salesperson 2D Problem, we encountered several challenges that led to failed attempts. These failures were not in vain and brought us unique insights and made us learn a bit more to achieve a faster way of solving the problem. Below we show, and argue our failed attempts and why they failed while discussing the approach taken and what we gained from the attempts.

1.1 Attempt 1: Greedy Approach

We first chose to start simple, we thought it would be easier to work our way up to the more complex algorithms if we could understand the easy implementations. The first was kind of suggested from the assignment on Kattis. This attempt uses a greedy approach and follows a straightforward idea. We add the shortest edge that joins two components of the graph that were not previously connected to the list.

1.1.1 Algorithm Description

The algorithm proceeds as follows:

1. Start at the first point (index 0) and mark it as visited.
2. Initialize a list called `tour` to store the sequence of visited points.
3. For each step from 1 to $N - 1$:
    - Find the closest unvisited point to the current point.
    - Add this closest point to the tour and mark it as visited.
4. Output the indices of the points in the order they are visited.

1.1.2 Complexity

- **Time Complexity:** $O(N^2)$
  - The outer loop runs N times.
  - The inner loop iterates through all unvisited points, taking $O(N)$ time.
  - Total time complexity: $O(N \times N) = O(N^2)$.
- **Space Complexity:** $O(N)$
  - Besides the points, the algorithm uses additional arrays of size N for `used` and `tour`.

1.1.3 Limitations and Potential Improvements

With this implementation, we managed to get a very low score of around X. This is not enough but we kind of expected it to not be enough because of its simplicity. For better results, more sophisticated algorithms such as **2-opt**, **3-opt**, or **Christofides' algorithm** can be used to refine the tour.

1.2 Attempt 2: Greedy + 2-Opt Swap

As stated earlier, we wanted a simple base that we could work on, and in our second attempt, we did just that. We used our previous work and added a 2-Opt Swap to further improve our solution.

1.2.1 Concept of 2-opt Swap

The 2-opt swap that is being implemented attempts to improve the tour by removing two edges and reconnecting the segments in a different way to reduce the total length of the tour. [1] Specifically:

1. Select two edges (a, b) and (c, d) in the tour.
2. Check if swapping the edges to form (a, c) and (b, d) reduces the total distance.
3. If the swap results in a shorter tour, reverse the segment between b and c to apply the change.

The process is repeated until no further improvement can be made.

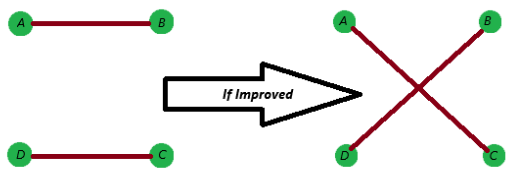

Figure 1: Visual Representation of a 2-opt Swap

1.2.2 Complexity

- **Time Complexity:** The 2-opt optimization has a time complexity of $O(N^2)$ for each iteration. Since it may require multiple iterations to reach a local optimum, the worst-case time complexity is $O(N^3)$.
- **Space Complexity:** The space complexity remains $O(N)$, as the function only modifies the existing tour vector.

1.2.3 Limitations and Potential Improvements

Although 2-opt improves our first attempt with the greedy solution because of how it reduces the length of the *tour* by eliminating crossing edges and and obtaining a more optimal path in a lot of cases, it does have some limitations as we managed to only get a score of **11 points**:

Now the limitation of this solution, is that we could get stuck in the local minimum instead of achieving the global minimum. The increased time complexity does also plays a role. Finally, the limited edge swapping, the algorithm only considers 2-edge swaps, which may not be enough for some instances needing more complex rearrangements. Here we enter the potential improvements. Extending the swaps to **3-opt** could offer better results or using **Christofides'**. We also found that using Metaheuristics such as **Simulated Annealing** could help us escape the local minimum.

1.3 Attempt 3: Matrix

Originally this attempt was not meant to be included in the report since it was us just trying to improve the performance in terms of complexity. We thought it would not be as relevant as trying **3-opt** or **Christofides'**. However, achieving the gain in score from this simple method does deserve some recognition.

We manage to achieve a much higher score with this simple change. More specifically, we managed to achieve a score of **18 points**.

1.3.1 Efficient Lookup

In this attempt, we incorporated a form to efficiently lookup the distance. Using a form of distance matrix we can precompute and store the distances instead of what we've done previously which is just calculating distances on the fly every time a new pair of points is evaluated.

Now this is very effective in our case because we use 2-opt swaps, which involves numerous comparisons and without the distance matrix, calculating each distance would lead to higher computational costs. Specifically if we have to check many edges for improvements, we initially would have to re-compute the differences, but with a matrix this issue gets solved. [2]

1.3.2 Complexity

Doing it this way we gain a lot in terms of time complexity. The overall complexity is the same as stated in #1.2.2 but the difference here is that the matrix lets us achieve a $O(1)$ lookup time for each distance reference within the 2-opt swap.

The space complexity increases from $O(N)$ to $O(N^2)$.

2 Successful Attempts

Step by step, we made several successful attempts, which led us to positive results. Below, we explain and detail our successful efforts, providing insights into how we achieved them.

2.1 Attempt 4: Distance Matrix + 2-Opt with Early Stopping

In this attempt, we made one more improvement by adding an early stopping condition to the 2-opt optimization. The idea was to stop the algorithm if no improvements were made after a full pass through the tour. In this way, we can avoid unnecessary computations when the tour is already close to the best possible solution.

Adding this improvement, the score achieved is **22 points**.

2.1.1 Algorithm Description

The overall process is the same as before, but now we add an early stop to the 2-opt optimization. The algorithm works like this:

1. Precompute the distance between all pairs of points and store them in a distance matrix.
2. Start with an initial greedy tour, where at each step we select the nearest unvisited point.
3. Use the 2-opt algorithm to try to improve the tour by swapping pairs of edges.
4. If no improvements are found after a complete pass through all pairs of edges, stop the optimization early.
5. Output the final tour.

2.1.2 Complexity

The 2-opt optimization step still has a time complexity of $O(N^3)$, but with the early stopping, we might stop earlier, saving time in practice.

The space complexity remains the same, $O(N^2)$.

2.2 Attempt 5: Multiple Starts with Randomization

On top of previous attempts, we improved our solution by introducing randomness to the starting point. Until now, we started just from the node 0, now we are trying to explore multiple tours starting from different points in the input. This allows the algorithm to potentially avoid getting stuck in local minima.

We achieved a score of **24 points**.

2.2.1 Algorithm Description

The algorithm proceeds as follows:

1. Precompute the distance between all pairs of points and store them in a distance matrix.
2. Repeat the following steps for a fixed number of attempts (e.g., 10):
   - Choose a random starting point from the set of points.
   - Generate a greedy tour starting from the random point.
   - Apply the 2-opt optimization to improve the tour.
   - Keep track of the best tour found across all attempts.
3. After all attempts, compare and output the best tour found.

2.2.2 Complexity

The time complexity remains $O(N^2)$ for the greedy step, and $O(N^3)$ for the 2-opt optimization. Since the algorithm repeats the process for a fixed number of random starts, the total time complexity is $O(K \times N^3)$, where K is the number of attempts.

The space complexity is still $O(N^2)$.

2.3 Attempt 6: Simulated Annealing with Multiple Starts

In the sixth attempt, we introduced a new method, Simulated Annealing. This approach, combined with multiple random starts, find a good global solution by accepting worse solutions with a certain probability. The idea is to simulate the annealing process in which the system explores a wide range of possible solutions early on, gradually narrowing down the search space as it cools down.

We achieved a score of **26 points** in this attempt.

2.3.1 Algorithm Description

The simulated annealing approach works as follows:

1. Precompute the distance between all pairs of points and store them in a distance matrix.
2. Repeat the following steps for a fixed number of attempts:
   - Choose a random starting point from the set of points.
   - Generate a greedy tour starting from the random point.
   - Apply the Simulated Annealing algorithm to optimize the tour:
     - At each step, randomly select two points in the tour and swap them.
     - If the new tour is shorter, accept the new tour.
     - If the new tour is longer, accept it with a probability based on the current temperature, which gradually decreases over time.
     - The temperature is reduced according to a cooling schedule, such as an exponential decay.
   - After the SA process, apply 2-opt to refine the solution further.
   - Keep track of the best tour found across all attempts.
3. After all attempts, compare and output the best tour found.

2.3.2 Complexity

The simulated annealing requires $O(N^2)$ time to swap pairs of points, and each swap is followed by a computation of the tour length, so another $O(N^2)$). The cooling process slowly reduces the number of swaps required, but the worst-case complexity is still around $O(N^3)$ for each attempt. And again, since we run multiple attempts, the total time complexity is $O(K \times N^3)$.

The space complexity remains $O(N^2)$.

2.3.3 Limitations

While Simulated Annealing was a nice improvement, the complexity still limits its performance for very large inputs. The key limitation here is that Simulated Annealing is not guaranteed to find the global optimum and might still get stuck in local minima.

2.4 Attempt 7: Parallel Simulated Annealing with Multiple Starts

For our seventh attempt and the last one, we parallelized the Simulated Annealing approach explained before. By running multiple instances of simulated annealing concurrently on different threads, we found more solutions in a shorter time. Each thread starts with a random starting point and performs simulated annealing independently.

This method gave us the best score, **28 points**.

2.4.1 Algorithm Description

The parallelized simulated annealing works as follows:

1. Precompute the distance between all pairs of points and store them in a distance matrix.
2. Launch multiple threads to run the Simulated Annealing process concurrently:
   - Each thread starts from a random point and generates a greedy tour.
   - Simulated Annealing is applied on the tour to optimize it:
     - The algorithm performs random swaps of two points.
     - A new tour is accepted if it results in a shorter distance, or if it is worse with a certain probability based on the temperature.

- The temperature gradually decreases based on a cooling schedule.
- After the Simulated Annealing step, 2-opt optimization is performed to refine the tour.
- The best tour found by all threads is tracked and updated as needed.

3. After all threads have finished, output the best tour found.

2.4.2 Complexity

Since each thread performs an $O(N^3)$ operation, the total complexity across all threads is still $O(K \times N^3)$, where K is the number of parallel threads. However, because the threads work in parallel, the overall runtime is significantly reduced, and the amount of work is higher.

The space complexity is still $O(N^2)$, as the distance matrix is shared across all threads.

3 Another Attempts

In addition to the approaches mentioned earlier, we explored several other methods in an attempt to find a better solution. However, most of these methods led to "Time Limit Exceeded" errors due to their higher time complexity.

For example, we tried with 3-opt optimization, which showed some improvement in certain cases, but it did not give a significant increase in the score. After further attempts to combine 3-opt with other strategies, we encountered "Time Limit Exceeded" errors, indicating that the complexity outweighed the potential benefits.

We also tried integrating 3-opt with 2-opt, expecting a better balance between optimality and performance. However, this combination did not provide a noticeable improvement in the path length, and the time complexity remained high, offering no real advantage.

Using Christofides' Algorithm also produced similar results to 3-opt. When combined with 2-opt, it did not lead to better optimality or shorter paths, suggesting that the added complexity was not worthwhile for this specific problem.

To further refine our solution, we increased the number of attempts for the 2-opt algorithm, running it multiple times and comparing the results. While this approach initially showed an improving trend, it eventually reached the 2-second time limit, limiting its effectiveness.

We also explored other algorithms, such as Lin–Kernighan combined with 2-opt. However, like the previous methods, it did not result in any meaningful improvement in optimality.

Lastly, we attempted to use more advanced techniques such as Genetic Algorithms and Ant Colony Optimization. Unfortunately, both of these methods resulted in "Time Limit Exceeded" errors, primarily due to their high time complexity. Despite the efforts to optimize the solution further with these methods, the complexity and time limitations ultimately hindered any significant improvements.

4 Conclusion

Through our iterative process, we managed to improve our solution, from a simple greedy approach to more complex techniques such as 2-opt, simulated annealing, and parallelization. Despite the inherent complexity and limitations of each approach, combining these methods allowed us to achieve a final score of **28 points**.

References

[1] A. S. Brain, "Traveling Salesman Problem with the 2-opt Algorithm," *Medium*, Available at: https://slowandsteadybrain.medium.com/traveling-salesman-problem-ce78187cf1f3.

[2] Z. Zhang et al., "An Improvement to the 2-Opt Heuristic Algorithm for Approximation of Optimal TSP Tour," *MDPI*, vol. 13, no. 12, pp. 7339, 2023. Available at: https://www.mdpi.com/2076-3417/13/12/7339.

[3] Wikipedia contributors. *Lin–Kernighan Heuristic*. Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic.

[4] S. Lin and B. W. Kernighan, "An Efficient Heuristic Procedure for the Traveling-Salesman Problem," *Operations Research*, vol. 21, no. 2, pp. 498-516, 1973. Available at: https://ieeexplore.ieee.org/abstract/document/5200345.

[5] S. Huang et al., "Recent Advances in Optimization for the Traveling Salesman Problem," *arXiv*, 2023. Available at: https://arxiv.org/abs/2302.06889.

[6] A. Kr, "Christofides' Algorithm," Available at: https://alon.kr/posts/christofides.

[7] Christofides' Algorithm Explained. *YouTube*, Available at: https://www.youtube.com/watch?v=UCUh5D7W6_s.