DSA4212: Travelling Salesperson Problem (TSP)

Group Composition:

1. Lim Jing Rui, A0233959N, jingrui.lim@u.nus.edu

1 Introduction

The Traveling Salesman Problem (TSP) is a well-known NP-hard problem. The goal is to determine the shortest possible route (tour), or the route with best (least) cost, that visits each city once and returns to the origin city. This dataset (USCA312, [Flo19]) consists of 312 cities, represented by a distance matrix G where each entry corresponds to the distance between a pair of cities. The optimal tour cost is 46008 as later discussed in 2.6

2 Algorithms

2.1 Brute force

The brute force or naive approach to find the best tour is to look at all possible permutations of tours. A dataset of 9 cities was manually created to test this. The algorithm was successfully able to find the optimal tour in 1.4 seconds. Since the algorithm works by comparing the cost of every permutation, it has a time complexity of $\mathcal{O}(n!)$ [Wik24c]. Increasing the dataset to 10 cities, we can see that the time taken increases from 1.4 seconds to 17.2 seconds. This approach starts to become infeasible from this point on.

2.1.1 Dynamic Programming (DP)

We can however optimise this brute force approach. Note that while calculating the cost of every single permutation, some sub permutations are bound to be repeated. We can avoid these redundant calculations with a top down recursive solution with memoization, where the current city and cities visited are stored. Other DP approaches include the Held-Karp algorithm [Wik24b], and these algorithms bring the time complexity down to $\mathcal{O}(2^n n^2)$. The time taken to run decreases to 147 microseconds and 9.22 seconds for 9 and 10 cities respectively.

2.2 Greedy

The simplest and most intuitive heuristic or approximation is a greedy algorithm. For every city, it makes local changes by simply choosing the closest city as it's next move [Wik24c]. It was quickly able to produce a cost of 57187 for the dataset. This runs in $\mathcal{O}(n^2)$ time. An optimization with a min-heap was attempted as it reduces the time complexity of the search for the nearest city from $\mathcal{O}(n)$ to $\mathcal{O}(log\, n)$. However, the overhead of managing the heap can outweigh the $\mathcal{O}(\log n)$ improvement for datasets of this size. For significantly larger datasets, however, the heap-based implementation may offer better performance.

2.3 2-opt

2-opt is a local search algorithm. The main idea behind a 2-opt swap is to take a route that crosses over itself and reorder it so that it does not, thereby shortening the tour length [Wik24a].

$$\begin{array}{ccc} -A & & B- \\ & \times & \\ -C & & D- \end{array} \quad \Rightarrow \quad \begin{array}{c} -A-B- \\ -C-D- \end{array}$$

In a complete 2-opt search, every possible pair of edges is considered for swapping, and any swap that reduces the total tour cost is applied. This continues until no further improvements can be found. Starting with a random tour, it was able to achieve a best cost of 48725 over 30 iterations.

2.3.1 2-opt with greedy start

Starting with the resulting tour from a greedy algorithm, the 2-opt algorithm aims to further refine the solution by eliminating crossovers. Theoretically, this combination should lead to a shorter tour, as the greedy algorithm provides a reasonable initial solution for 2-opt to improve upon. However, in practice, the best cost achieved over 30 iterations was 52,407, which is worse than using 2-opt on its own with a random initial tour. A potential reason for this is that the greedy algorithm produces a solution that is already locally optimal, and 2-opt has limited flexibility to explore alternative paths, preventing it from finding a better global solution.

2.4 Simulated Annealing

Simulated annealing (SA) is a meta heuristic computational method borrowing inspiration of the physical process of solid annealing. It is a method generally known to escape local optima. Figure 1 outlines the process [All23]. The changes in path length are accepted with probability p, controlled by a temperature parameter. However, a key challenge of this method lies in correctly identifying the optimal initial parameters — the initial temperature, cooling rate (alpha), and maximum iterations.

2.4.1 Local changes

A local change proposed was a swap, which exchanges the positions of two randomly selected cities in the tour. Starting with initial parameters of `initial_temp` = 50000, `alpha` = 0.995, and `max_iter` = 10000, the algorithm achieved a best cost of 155622 over 30 iterations. Initializing the algorithm with the result of a greedy tour brought the best cost down to 152316, which was still far from optimal.
Further experimentation at best only managed to improve the best cost to 87926 with parameters of `initial_temp` = 3000000, `alpha` = 0.9995, and `max_iter` = 1000000. This demonstrates that finding optimal parameters for SA can be a long and tedious process.

The other local change proposed was a removal and insert, where a random city is selected and moved to another random position. With the same inital parameters, the algorithm was able to achieve a best cost of 143600, and 139763 with a greedy initial tour, over 30 iterations. With our experimental best parameters, it achieved a best cost of 65066. With these improvements, it can be concluded this local change is more suited to this dataset.

2.4.2 Testing with 2-opt

Aside from testing with greedy initial tours, the algorithm was also tested with 2-opt. Over 30 iterations, a combination of the three algorithms yielded different results.

- Greedy $\rightarrow$ SA $\rightarrow$ 2-opt: Best cost of 48069
- Greedy $\rightarrow$ 2-opt $\rightarrow$ SA: Best cost of 154262

A potential reason for this discrepancy is that 2-opt may have caused the solution to settle in a local minimum, making it difficult for SA to escape.

Finally, SA was tested with 2-opt incorporated as a local swap within the SA algorithm itself. With the initial parameters, the algorithm achieved a best cost of 106559, and 103461 with a greedy initial tour, over 30 iterations. Using the experimental best parameters, it achieved a best cost of 49472, representing a significant improvement.

2.5 Genetic Algorithm

Genetic algorithms (GA) are a heuristic process based on the concept of natural selection. The basic process is as follows [She24]:

1. Initialise the Population
2. Calculate Fitness (Cost)
3. Selection (Roulette Wheel)
4. Crossover (Creating offspring)

5. Mutation (Introducing variation)
6. Replacement (Of old with new)
7. Repeat and Terminate

Again, a key challenge of a genetic algorithm is the selection of initial parameters. In this case, these parameters are the population size (number of paths/tours) and number of generations. Given the computational expense, the algorithm was tested with a population size of up to 800, 1500 generations, and a mutation rate of 0.03. The algorithm was tested with two replacement strategies:

- **Elitist Replacement**: The top 10 percent of the old population was retained in the new population. This approach resulted in a best cost of 165624.
- **Generational Replacement**: The new population completely replaced the old population in each generation. This approach struggled with a plateauing cost and yielded a best cost of 416124.

Elitist replacement seems to be better for our dataset, but still the results are far from optimal.

2.5.1 Local Changes

(For the purposes of testing, `pop_size=200`, `n_generations=500`, `mutation_rate=0.03`, `elite_fraction=0.1`)
Local changes were tested during the crossover phase, specifically using Partially Mapped Crossovers (PMX) and Order Crossovers (OX). These methods are detailed in [Jam24]. The crossover type affects how offspring inherit genetic material from their parents.

Both crossover methods yielded approximately the same result, with a best cost of 247000. However, it was noted that OX took nearly twice as much time as PMX. This difference may be due to the fact that OX requires the preservation of order, which demands additional computational resources. However, additional testing is required.

2.5.2 Testing with other algorithms

The Genetic Algorithm (GA) was tested with an initial greedy tour as a starting solution. However, the best cost achieved with the greedy initialization was 250193, higher than that obtained with an initial random tour, albeit improved to 230803 by increasing `mutation_rate` to 0.05. This outcome is consistent with an experiment conducted by Deng et al. (2015) [DLZ15], where an initial greedy tour often resulted in poorer performance compared to a random initial tour.

A potential reason for this could be the GA's inability to escape local minima introduced by the greedy algorithm, especially given the limited number of parameters used in this setup. With further testing and parameter tuning, the underlying reasons for this performance difference could be better established.

GA was also initialised with a 2-opt swap after crossover in hopes of local improvements. Due to computational restrictions, `pop_size` was reduced to 200, and testing subsequently found the algorithm plateaued around 10 generations. It was able to achieve a best cost of 47253 which is the best cost so far.

2.6 PyConcorde

PyConcorde is a Python wrapper for *Concorde*, a software developed to solve large TSPs optimally. PyConcorde was able to get a best cost of 46008, which is believed to be the optimal solution.

To test PyConcorde, a known TSP dataset att48 was loaded and tested. PyConcorde achieve a best cost of 33523, which was consistent with the minimum cost as stated on [Flo19]

3 Conclusion

Algorithm	**Best Cost**
Greedy	57187
2-opt	48725
2-opt with Greedy Start	52407
Simulated Annealing (SA)	65066
Greedy $\rightarrow$ SA $\rightarrow$ 2-opt	48069
Greedy $\rightarrow$ 2-opt $\rightarrow$ SA	154262
Genetic Algorithm (GA)	165624
Genetic Algorithm with 2-opt	47253
PyConcorde	46008

Table 1: Best Costs Achieved by Each Algorithm

This report presented an exploration of various approaches to solving TSP. To further enhance the performance of these methods, future work could focus on experimenting with and optimizing key parameters, as well as testing the algorithms on a broader range of datasets. Additionally, other methods, such as Ant Colony Optimization and Tabu Search, can be explored as alternative approaches or in hybrid combinations with current algorithms. These methods may provide additional insights and improvements in solution quality and convergence speed for large and complex TSP instances.

References

[All23] Francis Allanah. Travelling salesman problem using simulated annealing. Medium, January 23 2023. https://medium.com/@francis.allanah/travelling-salesman-problem-using-simulated-annealing-f547a71ab3c6.

[DLZ15] Yuxin Deng, Yan Liu, and Dong Zhou. An improved genetic algorithm with initial population strategy for symmetric tsp. *Mathematical Problems in Engineering*, 2015:1–6, 2015.

[Flo19] Florida State University. TSP Data for the Traveling Salesperson Problem. TSP - Data for the Traveling Salesperson Problem. https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html, 2019. Accessed: 2024-11-14.

[Jam24] M. Jamdade. The genetic algorithm and the travelling salesman problem (tsp). Medium, March 15 2024. https://itnext.io/the-genetic-algorithm-and-the-travelling-salesman-problem-tsp-31dfa57f3b62.

[She24] R. Shendy. Traveling salesman problem (tsp) using genetic algorithm (python). Medium, January 24 2024. https://medium.com/aimonks/traveling-salesman-problem-tsp-using-genetic-algorithm-fea640713758.

[Wik24a] Wikipedia contributors. 2-opt. https://en.wikipedia.org/wiki/2-opt, August 15 2024. Accessed: 2024-11-14.

[Wik24b] Wikipedia contributors. Held–karp algorithm. https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm, June 12 2024. Accessed: 2024-11-14.

[Wik24c] Wikipedia contributors. Travelling salesman problem. https://en.wikipedia.org/wiki/Travelling_salesman_problem, November 2 2024. Accessed: 2024-11-14.

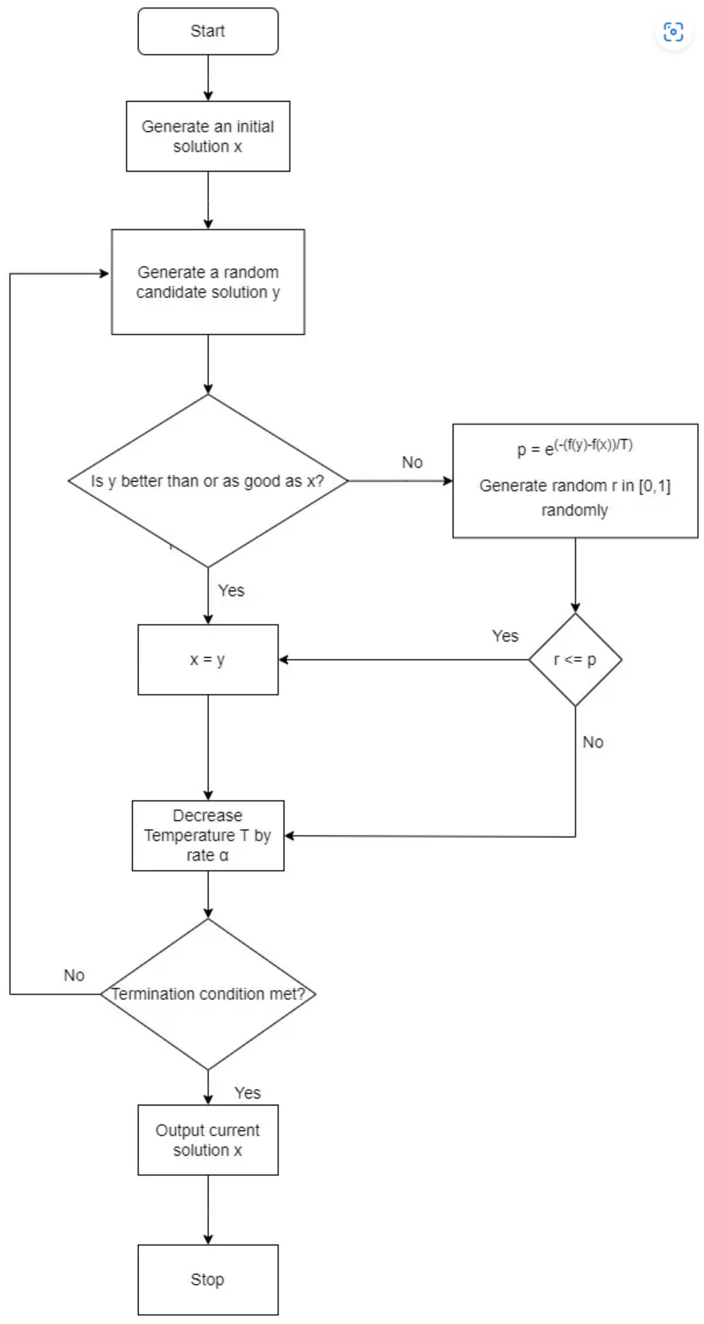
Start
Generate an initial solution x
Generate a random candidate solution y
Is y better than or as good as x?
No
$p = e^{(-(f(y)-f(x))/T)}$
Generate random r in [0,1] randomly
Yes
x = y
Yes
r <= p
No
Decrease Temperature T by rate α
No
Termination condition met?
Yes
Output current solution x
Stop

Figure 1: Simulated Annealing Process