# DSA4212: Traveling Salesman Problem

Bao Jiaqi, A0211257N, bao.jiaqi@u.nus.edu

April 19, 2024

## 1 Introduction

The objective of the travelling salesman problem (TSP) is to find the shortest possible tour that visits each city exactly once and returns to the origin city. Given a list of cities and the distances between each pair, this problem raises the question: ”What is the optimal tour”.

Here are some notations used in this report. We represent each solution tour as a list $s$ of length $C$. Here, $s[i]$ refers to the $i$-th city along the tour, and $C$ represents the total number of cities. Additionally, $s[i:]$ denotes the part of the tour from the $i$-th city visited to the last, while $s[i:j]$ represents the portion from the $i$-th city visited to the $j$-th city visited. Furthermore, we define $dist(s)$ as the total distance or length of a tour $s$. It is calculated by summing the distance between consecutive cities : $d(s[C], s[1]) + \sum_{i=1}^{C-1} d(s[i], s[i+1])$. Here, $d(s[C], s[1])$ represents the distance between the last city and the origin city, while $\sum_{i=1}^{C-1} d(s[i], s[i+1])$ sums up the distances between each pair of cities along the tour, and $d(x, y)$ denotes the distance between city $x$ and city $y$.

## 2 Methods

### 2.1 Greedy Algorithm

The greedy algorithm, also known as the nearest neighbor algorithm, begins by randomly selecting one city. It then chooses the closest neighboring city that has not yet been visited, continuing this process until all cities have been visited.

### 2.2 $k$-opt local search heuristics

The $k$-opt heuristics involve removing $k$ links from the current tour and reattaching the resulting segments to create an improved tour. This approach accepts the improved solution only and iterates until no further improvements are made.

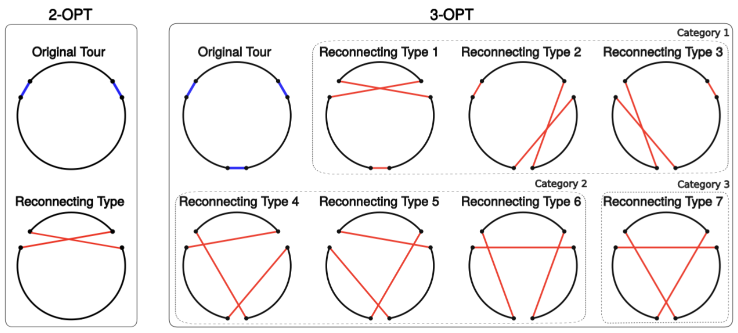


Figure 1: Reconnecting types of 2-opt and 3-opt heuristics [SDL+21].

**Algorithm 1** 2-opt heuristics (2-opt)

```
1: Input: Number of cities $C$, initial solution $s^{(0)}$.
2: Set best solution $s^* \leftarrow s$ and current solution $s \leftarrow s^{(0)}$
3: improved $\leftarrow$ True
4: while improved do
5:     improved $\leftarrow$ False
6:     for $i = 1 \dots C - 2$ do
7:         for $j = i + 1 \dots C$ do
8:             $s' \leftarrow s[i:] + s[i:j][::-1] + s[j:]$ # Here, [::-1] indicates the reversion of a list.
9:             if $dist(s') < dist(s^*)$ then
10:                Update $s^* \leftarrow s', s \leftarrow s'$, and update improved $\leftarrow$ True
11:            end if
12:        end for
13:    end for
14:    $s \leftarrow s^*$
15: end while
```

**Algorithm 1** 3-opt heuristics (3-opt)

```
Input: Number of cities $C$, initial solution $s^{(0)}$.
Set best solution $s^* \leftarrow s$ and current solution $s \leftarrow s^{(0)}$
improved $\leftarrow$ True
while improved do
    improved $\leftarrow$ False
    for $i = 1 \dots C - 4$ do
        for $j = i + 2 \dots C - 2$ do
            for $k = j + 2 \dots C - 4$ do
                $s' \leftarrow s[:i] + s[i:j][::-1] + s[j:k][::-1] + s[k:]$
                if $dist(s') < dist(s^*)$ then
                    Update $s^* \leftarrow s', s \leftarrow s'$, and update improved $\leftarrow$ True
                end if
            end for
        end for
    end for
end while
```

Figure 1 illustrates the different types of reconnections for 2-opt and 3-opt. In 2-opt, there is only one type of reconnection after removal, whereas 3-opt has 7 types of reconnections, with only type 6 will be applied in this project. The detailed algorithms for 2-opt and 3-opt are presented in the left and right panels of Algorithm 1, respectively.

## 2.3 Tabu Search

Tabu Search (TS) [Tab] is a metaheuristic search method that refines local search methods introduced in Section 2.2. It starts with an initial solution and generates a set of candidate neighbor solutions by applying local search operators to the current solution. In this project, we adopt 2-opt swaps as the local search operator. The best non-tabu neighbor solution is then accepted and added to the tabu list.

The tabu list prevents revisiting previously explored solutions, which promotes search diversification and helps escape local optima. Unlike $k$-opt heuristics that only update the current solution if an improvement is made, the tabu search algorithm accepts worsening tours if no improving tour is available to avoid becoming stuck at a local minimum. The algorithm is outlined in Algorithm 2.

**Algorithm 2** Tabu Search (TS)

```
1: Input: Initial solution $s^{(0)}$, tabu size $\lambda$, number of iterations $N$, number of cities $C$.
2: Set best solution $s^* \leftarrow s^{(0)}$.
3: Initialize the tabu list of solutions $\mathcal{T}^{(0)} \leftarrow \emptyset$
4: for each iteration $n = 1, \dots, N$ do
5:     Get all $\binom{C}{2}$ neighbor solutions $s_n$ of the current solution $s$ by swapping any two cities in $s^{(n-1)}$.
6:     Get the best non-tabu neighbor solution $s_n^* \leftarrow \arg\min_{s_n \notin \mathcal{T}^{(n-1)}} dist(s_n)$ and update current solution $s^{(n)} \leftarrow s_n^*$
7:     If $dist(s_n^*) < dist(s^*)$, then update best solution $s^* \leftarrow s_n^*$
8:     Add the best non-tabu solution to the tabu list: $\mathcal{T}^{(n)} \leftarrow \mathcal{T}^{(n-1)} \cup \{s_n^*\}$. If $|\mathcal{T}^{(n)}| > \lambda$, then remove the solution that was added earliest from $\mathcal{T}^{(n)}$.
9: end for
```

### 2.4 Simulated Annealing

Next, we will introduce nature-inspired metaheuristic methods for solving the TSP problem, which go beyond the local search heuristics discussed earlier.

Firstly, Simulated Annealing (SA) [Sim] mimics the annealing process in metallurgy by probabilistically accepting initially worse solutions and gradually reducing the acceptance probability over time. SA starts with a random initial solution and selects a random neighbor of the current solution. In the context of the TSP, a neighbor of a solution $s$ can be another solution $s'$ with only two different entries between them. If the distance of $s'$ is less than the distance of $s$, we update the current solution. Otherwise, we accept $s'$ with a probability of $\frac{1}{T}\exp(dist(s') - dist(s))$, where $T$ represents the temperature which decreases with each iteration. Therefore, the algorithm iteratively explores the solution space by accepting worse solutions with a decreasing probability.

The detailed algorithm for Simulated Annealing can be found in Algorithm 3.

**Algorithm 3** Simulated Annealing (SA)

```
1:  **Input**: Initial solution $s^{(0)}$, initial temperature $T^{(0)}$, cooling rate $\rho$, number of iterations $N$.
2:  Set best solution $s^* \leftarrow s^{(0)}$.
3:  **for** each iteration $n = 1, \dots, N$ **do**
4:      Generate a new solution $s'$ by randomly swap 2 cities in current solution $s^{(n-1)}$
5:      **if** $dist(s') < dist(s^{(n-1)})$ **then**
6:          Update $s^{(n)} \leftarrow s'$
7:      **else**
8:          Update $s^{(n)} \leftarrow s'$ with acceptance probability which equals $\frac{1}{T^{(n-1)}}\exp(dist(s') - dist(s^{(n-1)}))$
9:      **end if**
10:     If $dist(s^{(n)}) < dist(s^*)$, then we update best solution as $s^* \leftarrow s^{(n)}$
11:     Cool down temperature with $T^{(n)} \leftarrow \rho T^{(n-1)}$
12: **end for**
```

### 2.5 Genetic Algorithm

Genetic Algorithm (GA) [Sto18] is a swarm intelligence method that simulates the social behavior of real animals through a population. It is a nature-inspired metaheuristic approach used to solve the TSP problem by mimicking the principle of natural selection.

With an initial population of candidate solutions, GA selects solutions with short total distances from the population to serve as parents for the next generation. The child solutions are generated by combining information from the parents. To introduce more diversity into the population, certain children can undergo mutation or small changes. The details of the algorithm are outlined in Algorithm 4.

### 2.6 Ant Colony Optimization

Ant Colony Optimization (ACO) [WH21] is another swarm intelligence optimization algorithm inspired by the foraging behavior of ants. In ACO, each ant $a$ represents a solution $s_a$, and the pheromone $\gamma_{xy}$ represents the attractiveness of the connection between any two cities $x$ and $y$. Each ant, currently at city $i$, constructs a solution by iteratively selecting the next city $j$ to visit based on a probabilistic decision rule that depends on $\gamma_{ij}$ (the attractiveness of the connection between city $i$ and $j$) and $d(i, j)$ (the distance between city $i$ and $j$). After all ants have constructed their tours, the pheromone levels $\gamma_{xy}$ for all cities $x$ and $y$ are updated based on the distances $dist(s_a)$ from all ants $a$ that consecutively visited cities $x$ and $y$.

As the algorithm progresses, pheromone levels converge towards the edges of optimal solutions, biasing subsequent ants to explore promising areas of the search space. The algorithm is described in Algorithm 5.

**Algorithm 4** Genetic Algorithm (GA)

```
1: Input: Initial populations $\mathcal{P}^{(0)}$, population size $\lambda$, tournament size $\gamma$, mutation rate $r$, number of iterations $N$.
2: Set best solution $s^* \leftarrow \arg\min_{s\in\mathcal{P}_0} dist(s)$.
3: for each generation $n = 1, \ldots, N$ do
4:   Initialize an empty current population set $\mathcal{P}^{(n)} \leftarrow \emptyset$.
5:   for each child $i = 1, \ldots, \lambda$ do
6:     for each parent $j = 1, 2$ do
7:       Randomly select $\gamma$ solutions $\{s_i\}_{i=1}^{\gamma}$ from $\mathcal{P}^{(n-1)}$.
8:       $s_{\text{parent},j} \leftarrow \arg\min_{i=1,\ldots,\gamma} dist(s_i)$
9:     end for
10:    Randomly select a portion of the tour in $s_{\text{parent},1}$ and fill them into $s_{\text{child},i}$.
11:    Fill the remainder of the tour with the cities from the cities in $s_{\text{parent},2}$ that are not inside $s_{\text{child},i}$, in the
       order in which they appear.
12:    Randomly swap two cities in $s_{\text{child},i}$ with probability $r$
13:    If $dist(s_{\text{child},i}) < dist(s^*)$, then update $s^* \leftarrow s_{\text{child},i}$
14:    Add child into the current population: $\mathcal{P}^{(n)} \leftarrow \mathcal{P}^{(n)} \cup \{s_{\text{child},i}\}$
15:  end for
16: end for
```

**Algorithm 5** Ant Colony Optimization (ACO)

```
1: Input: Number of ants $\lambda$, pheromone factor $\alpha$, heuristic factor $\beta$, pheromone evaporation rate $\rho$, pheromone
   deposit factor $Q$, initial pheromone $\gamma^{(0)}$, total number of cities $C$, number of iterations $N$.
2: for each iteration $n = 1, \ldots, N$ do
3:   for each ant $a = 1, \ldots, \lambda$ do
4:     Start the solution $s_a$ at a random city.
5:     for For number of cities visited $k = 2, \ldots, C$ do
6:       Choose next unvisited city $j \notin s_a$ with probability $\frac{(\gamma_{ij}^{(n-1)})^\alpha (1/d(i,j))^\beta}{\sum_{u\notin s_a} (\gamma_{iu}^{(n-1)})^\alpha (1/d(i,u))^\beta}$, where $d(i,j)$ denotes the
         distance between city $i$ and $j$ and $i$ denotes the current city ant $a$ is at.
7:       Update current city $i \leftarrow j$ and set $s_a[k] \leftarrow i$
8:     end for
9:     If $dist(s_a) < dist(s^*)$, then update $s^* \leftarrow s_a$
10:  end for
11:  Update the pheromone between any two city $i, j$ with $\gamma_{ij}^{(n)} \leftarrow (1-\rho)\gamma_{ij}^{(n-1)} +$
     $\sum_{a=1}^{\lambda} \frac{Q}{dist(s_a)} I\{$There exists $k \in \{1, ..., C-1\}$ such that $s_a[k] = i$ and $s_a[k+1] = j$ or $s_a[k] =$
     $j$ and $s_a[k+1] = i\}$, where $I$ denotes the indicator function.
12: end for
```

### 2.7 Cross Entropy (CE)

Unlike SA, GA, and ACO, which are driven by meta-heuristics and inspired by laws of nature, Cross Entropy (CE) is an optimization technique derived from a precise and rigorous mathematical framework. CE operates in two steps: generating a data sample according to a certain distribution and updating the parameters of the distribution to produce a better sample in the next iteration.

In the context of the TSP, the solution tours are generated using a Markov process with a city-city transition matrix. The transition matrix is updated with the empirical transition counts based on better solutions. The update rules for the transition matrix are derived in [DBKMR05] by formulating TSP as a minimization problem and using Lagrange multipliers and Monte Carlo estimation. We have adapted the code from https://github.com/v-iashin/CrossEntropyTSP, and the detailed procedure is outlined in Algorithm 6.

**Algorithm 6** Cross Entropy (CE)

```
1: Input: Number of cities $C$, number of samples $\lambda = C^2$, smoothing factor $\alpha$, quantile $\rho$, and number of iterations $N$.
2: Initialize transition matrix as $P_{xy}^{(0)} \leftarrow \frac{1}{C-1}$ for any city $x$ and city $y$.
3: for each iteration $n = 1, \dots, N$ do
4:   for each sample $i = 1, \dots, \lambda$ do
5:     Start the solution $s_i$ at the first city.
6:     Make a copy of current transition matrix $Q^{(1)} \leftarrow P^{(n-1)}$
7:     for number of cities visited $k = 2, \dots, C$ do
8:       Obtain $Q^{(k)}$ from $Q^{(k-1)}$ by setting the $s_i[k-1]$-th column of $Q^{(k-1)}$ to 0 and normalizing the rows to sum up to 1.
9:       Choose next unvisited city $x$ from the distribution formed by the $s_i[k-1]$-th row of $Q^{(k)}$, and set set $s_i[k] \leftarrow x$
10:    end for
11:    If $dist(s_i) < dist(s^*)$, then update $s^* \leftarrow s_i$
12:  end for
13:  Select the Top $\rho\lambda$ samples in $\{s_i\}_{i=1}^{\lambda}$ with the lowest $dist(s_i)$.
14:  Compute the empirical transition frequency matrix with
     $F_{xy} \leftarrow |\{s_i | \text{There exists } k \in \{1, ..., C-1\} \text{ such that } s_i[k] = x \text{ and } s_i[k+1] = y\}| / \lambda$, which equals the fraction of sample tours in which transition from city $x$ to $y$ is made.
15:  Smoothly update the transition matrix as $P^{(n)} \leftarrow \alpha F + (1-\alpha) P^{(n-1)}$
16: end for
```

## 3 Experiments

### 3.1 Experiment settings

We conducted experiments on the berlin52 and att48 datasets[1]. The berlin52 dataset consists of 52 cities, with an optimal solution that has a total distance of 7,544. The att48 dataset contains 48 cities, having an optimal solution with a total distance of 33,523.

For each algorithm, we executed 5 repetitions on each cities dataset, using different initial solutions and random seeds. The number of total iterations was adjusted to ensure convergence of the best distances. Detailed settings and distance loss plots can be found in the Python notebook.

In the case of Simulated Annealing (SA), we set the initial temperature $T^{(0)}$ to 1,000 and the cooling rate $\rho$ to 0.99. For Tabu Search (TS), we set the tabu size $\lambda$ to 40. In the case of Genetic Algorithm (GA), we set $\lambda$ to 400, the tournament size $\gamma$ to 10, and the mutation rate $r$ to 0.1. For Ant Colony Optimization (ACO), we set the number of ants $\lambda$ to 100, the pheromone factor $\alpha$ to 1, the heuristic factor $\beta$ to 2, the pheromone evaporation rate $\rho$ to 0.1, the

[1]The cities data and the optimal tour can be found at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

pheromone deposit factor $Q$ to 100, and the initial pheromone $\gamma^{(0)}$ to 1. For Coss Entropy (CE), we set smoothing factor $\alpha = 0.5$ and quantile $\rho = 0.1$.

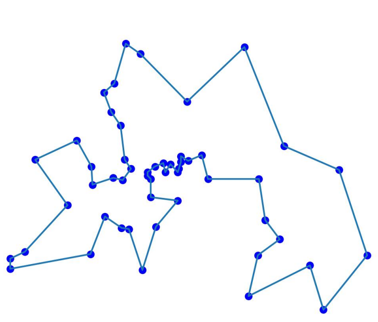
## 3.2 Results

Table 1 compares the best, worst, and average final total distances achieved ($dist(s^*)$) among all 5 repetitions. The error is calculated as $\frac{\text{average}-\text{optimal}}{\text{optimal}}$. It is evident that ACO achieves the shortest distance for both berlin52 and att48 datasets. CE obtains slightly longer distances. Additionally, the 3-opt, a heuristic local search approach, also obtains solutions that are relatively close to the optimal solution.

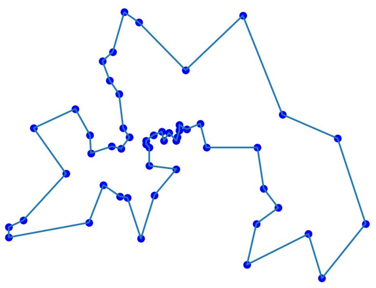
| | optimal | best | worst | average | error |
|---|---|---|---|---|---|
| | | berlin52 | | | |
| Greedy | 7,544 | 9,396 | 9,792 | 9,635 | 0.27 |
| 2-opt | 7,544 | 8,056 | 9,384 | 8,979 | 0.19 |
| 3-opt | 7,544 | 7,880 | 8,944 | 8,641 | 0.14 |
| TS | 7,544 | 9,164 | 10,679 | 9,908 | 0.31 |
| SA | 7,544 | 9,831 | 12,424 | 10,643 | 0.41 |
| GA | 7,544 | 8,839 | 11,313 | 9,672 | 0.28 |
| ACO | 7,544 | 7,776 | 7,847 | 7,812 | 0.035 |
| CE | 7,544 | 7,773 | 8,287 | 8,025 | 0.064 |
| 3 opt with ACO initialization | 7,544 | 7,596 | 7,863 | 7,762 | 0.028 |
| GA with ACO initialization | 7,544 | 7,544 | 7,804 | 7,642 | **0.013** |
| | | att48 | | | |
| Greedy | 33,523 | 40,880 | 42,444 | 41,455 | 0.23 |
| 2-opt | 33,523 | 35,708 | 40,887 | 38,628 | 0.15 |
| 3-opt | 33,523 | 36,356 | 37,705 | 36,938 | 0.10 |
| TS | 33,523 | 41,047 | 46,159 | 43,354 | 0.29 |
| SA | 33,523 | 40,646 | 59,162 | 47,499 | 0.41 |
| GA | 33,523 | 35,667 | 45,243 | 39,639 | 0.18 |
| ACO | 33,523 | 34,945 | 35,341 | 35,142 | 0.048 |
| CE | 33,523 | 34,779 | 36,142 | 35,387 | 0.055 |
| 3 opt with ACO initialization | 33,523 | 34,562 | 35,303 | 35,005 | 0.044 |
| GA with ACO initialization | 33,523 | 34,084 | 34,477 | 34,308 | **0.023** |

Table 1: The comparison the lengths of the solutions derived from each algorithm on berlin52 and att48.

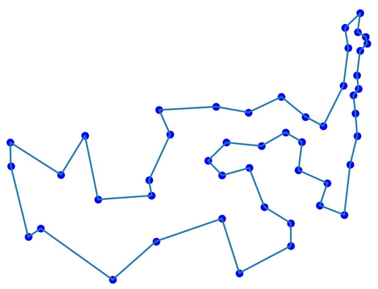
Finally, we utilize the solution obtained from ACO to initialize the 3-opt and GA algorithms. Specifically, for 3-opt, the initial solution is set as the solution derived from ACO. For GA, any solution in the initial population $\mathcal{P}^{(0)}$ is set to be the solution derived from ACO. The results are reported in Table 1. It is evident that GA with ACO initialization yields better solutions compared to 3-opt with ACO initialization. This observation suggests that nature-inspired metaheuristic methods are more effective than local search methods in further refining a solution that is close to the optimal solution. In the case of berlin52, GA with ACO initialization provides the best solution with a total distance of 7,544, matching the optimal solution. We visualize this solution in Figure 2b, which is identical to the optimal solution shown in Figure 2a. For att48, GA with ACO initialization yields the best solution with a total distance of 33,988. We illustrate this solution in Figure 2d, which is slightly worse than the optimal solution shown in Figure 2c. To achieve the optimal solution, we can explore other parameter settings, such as increasing the number of repetitions and iterations of ACO and GA.



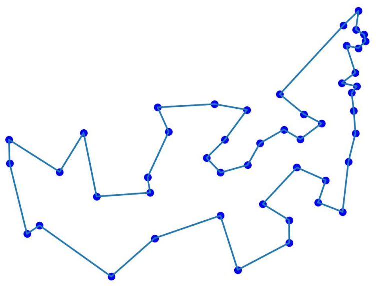
(a) Optimal solution of berlin52 (length=7,544)



(b) Our solution of berlin52 obtained by GA with ACO initialization (length=7,544)



(c) Optimal solution of att48 (length=33,523)



(d) Our solution of att48 obtained by GA with ACO initialization (length=34,084)

Figure 2: Visual comparison of optimal solutions and our solutions on berlin52 and att48.

## 4 Conclusion

In this report, we have utilized a variety of heuristic and metaheuristic methods to solve the Traveling Salesman Problem (TSP). Our observations show that local search heuristics (3-opt), a population-based method driven by statistical rigor (Cross Entropy), and nature-inspired swarm intelligence metaheuristic methods (Ant Colony Optimization) excel in obtaining good solutions for the TSP. Additionally, our experimental results indicate that nature-inspired metaheuristic algorithms are more effective in refining the solution further, ultimately achieving the optimal total distance compared to local search.

There are limitations to this project. For instance, we only conducted experiments with a relatively small number of cities. The scale of the problem, or the number of cities, is likely to affect the convergence of the algorithms. One potential solution is proposed in [DCH+12], which combines GA and ACO to achieve faster and better convergence as the scale of the problem increases.

## A Code availability

The datasets and the Python notebook can be found at https://github.com/maggie980000/Traveling-Salesman-Problem.

## References

[DBKMR05] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

[DCH+12] Wu Deng, Rong Chen, Bing He, Yaqing Liu, Lifeng Yin, and Jinghuan Guo. A novel two-stage hybrid swarm intelligence optimization algorithm and application. *Soft Computing*, 16:1707–1722, 2012.

[SDL+21] Jingyan Sui, Shizhe Ding, Ruizhi Liu, Liming Xu, and Dongbo Bu. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 1301–1316. PMLR, 2021.

[Sim] Simulated annealing. Simulated annealing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Simulated_annealing. Accessed: 2024-04-16.

[Sto18] Eric Stoltz. Evolution of a salesman: A complete genetic algorithm tutorial for python. https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35, 2018. Accessed: 2024-04-16.

[Tab] Tabu search. Tabu search — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Tabu_search. Accessed: 2024-04-16.

[WH21] Yong Wang and Zunpu Han. Ant colony optimization for traveling salesman problem based on parameters optimization. *Applied Soft Computing*, 107:107439, 2021.