COS 314
Artificial Intelligence
Assignment One

23 March 2025

Dewald Colesky u23536030

Table of contents:

Introduction

The Traveling Salesman Problem (TSP) is a classic optimization problem in the field of computer science and operations research. The problem involves finding the shortest possible route that visits each city exactly once and returns to the starting point. This problem has significant real-world applications in logistics, transportation, and network design. As the number of cities increases, solving the TSP optimally becomes computationally infeasible due to its NP-hard nature. Therefore, heuristic methods such as Simulated Annealing (SA) and Tabu Search (TS) are commonly employed to find near-optimal solutions in a reasonable amount of time.

This assignment aims to evaluate and compare the performance of two local search algorithms, Simulated Annealing (SA) and Tabu Search (TS), in solving instances of the Traveling Salesman Problem (TSP). We will analyze the efficiency and effectiveness of both algorithms by considering their ability to find near-optimal solutions while also exploring their computational costs.

Background

The Traveling Salesman Problem (TSP) is a well-known NP-hard problem, meaning that there is no known polynomial-time algorithm to solve it optimally for large instances. Exact algorithms, such as the brute force method or dynamic programming, guarantee the optimal solution but are computationally expensive for large inputs. As a result, heuristic and metaheuristic methods have become essential for tackling TSP in practical applications where large-scale instances are common.

Two such algorithms are Simulated Annealing (SA) and Tabu Search (TS).

- Simulated Annealing (SA) is inspired by the physical process of annealing, where a material is slowly cooled to reach a state of minimal energy. In the context of TSP, SA explores the solution space by probabilistically accepting worse solutions in the hope of escaping local optima, gradually reducing the probability of accepting worse solutions as the algorithm progresses.

- Tabu Search (TS) is a local search algorithm that employs memory structures to avoid revisiting previously explored solutions. It uses a "tabu list" to keep track of recently visited solutions, preventing the search from cycling back to those solutions. This memory helps TS escape local minima by diversifying the search.

Both algorithms offer trade-offs between computational cost and solution quality. SA is generally faster and easier to implement but may converge to suboptimal solutions. TS, on the other hand, can often find better solutions but may require more computational resources. In this assignment, we will compare the performance of SA and TS using a set of TSP instances to assess which algorithm provides a better balance between speed and solution quality.

Initial solution generation method.

Randomness in Search Algorithms

Both Simulated Annealing (SA) and Tabu Search (TS) are metaheuristic algorithms used to solve combinatorial optimization problems, like the Travelling Salesman Problem (TSP). Both SA and TS are based on exploration and exploitation, where randomness plays a key role in navigating the solution space.

Randomness in Simulated Annealing (SA)

The initial solution in SA must start and end with city 1. The rest of the path is a random permutation of the remaining cities. This starting solution is the initial configuration for the algorithm to explore the solution space.

Randomness in Tabu Search (TS)

Similar to SA, TS must start and end with city 1. The rest of the path is a random permutation of the remaining cities. This starting solution is the based for the search to explore neighbouring solutions.

Perturbation method.

Perturbation in Simulated Annealing

The perturbation method in Simulated Annealing involves making a small, random change to the current solution to generate a neighboring solution. This can be achieved by swapping two cities in the route. The perturbation is random and aims to explore new areas of the solution space.

Perturbation in Tabu Search

In Tabu Search, the perturbation method is also based on making small changes to the current solution. A typical perturbation is a swap or inversion of two cities. The perturbation is controlled to avoid revisiting solutions recently explored, which is handled by the Tabu list.

Neighbourhood definition.

Neighbourhood definition of Simulated Annealing
The neighborhood of a solution in Simulated Annealing consists of all solutions that can be generated by making a small random change to the current solution. For the TSP, this may include swapping two cities or reversing a subset of the tour.

Neighbourhood definition of Simulated Annealing
The neighborhood in Tabu Search is similar in that it consists of solutions generated by applying a small change to the current solution. However, Tabu Search uses a memory structure (Tabu list) to track recently visited solutions or moves, ensuring that the algorithm avoids revisiting solutions that could lead to cycling and premature convergence.

Acceptance criterion.

Acceptance criterion for Simulated Annealing

The acceptance criterion in Simulated Annealing is based on both the quality of the neighboring solution and a probabilistic function. If the neighboring solution is better than the current solution, it is accepted. If it is worse, it may still be accepted with a probability that decreases over time according to the Boltzmann distribution. This allows the algorithm to escape local optima early in the search.

Acceptance criterion for Tabu Search

The acceptance criterion in Tabu Search typically requires that the neighboring solution be better than the current one. In addition, Tabu Search employs a memory structure that records the moves made (Tabu list) to ensure that recently visited solutions are avoided, further guiding the search towards unexplored areas.

Stopping criteria.

Stopping criteria Simulated Annealing

The stopping criterion for Simulated Annealing is typically based on the cooling schedule or a predetermined number of iterations. The algorithm will stop either when the temperature reaches a predefined minimum value or when a specific number of iterations have been completed without an improvement in the solution.

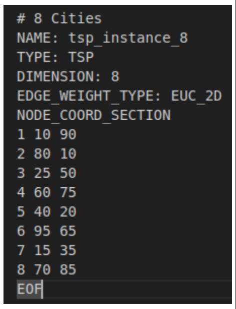
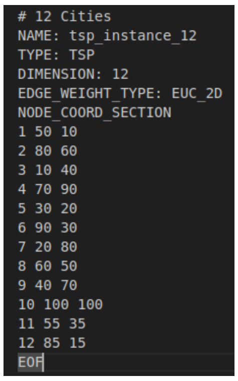
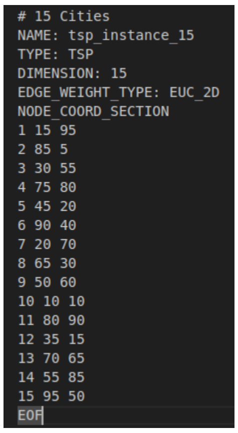
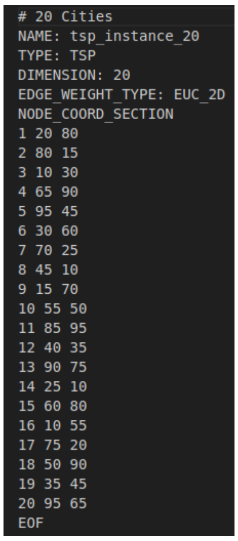
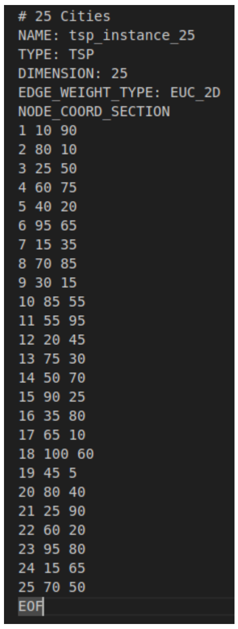
Stopping criteria Tabu Search

The stopping criterion for Tabu Search is usually based on either a fixed number of iterations or when a predefined stopping condition (maximum iterations without improvements) is met. Additionally, if the algorithm reaches a solution that cannot be improved further within the allowed number of iterations, the search will halt.

Experimental setup.

Problem Definition

The problem being solved is the Traveling Salesman Problem (TSP), where the objective is to find the shortest possible route that visits all cities exactly once and returns to the origin city. The following data sets are used, representing the number of cities to be visited:

8 cities	12 cities	15 cities	20 cities	25 cities
				

```
# 8 Cities
NAME: tsp_instance_8
TYPE: TSP
DIMENSION: 8
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 10 90
2 80 10
3 25 50
4 60 75
5 40 20
6 95 65
7 15 35
8 70 85
EOF
```

```
# 12 Cities
NAME: tsp_instance_12
TYPE: TSP
DIMENSION: 12
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 50 10
2 80 60
3 10 40
4 70 90
5 30 20
6 90 30
7 20 80
8 60 50
9 40 70
10 100 100
11 55 35
12 85 15
EOF
```

```
# 15 Cities
NAME: tsp_instance_15
TYPE: TSP
DIMENSION: 15
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 15 95
2 85 5
3 30 55
4 75 80
5 45 20
6 90 40
7 20 70
8 65 30
9 50 60
10 10 10
11 80 90
12 35 15
13 70 65
14 55 85
15 95 50
EOF
```

```
# 20 Cities
NAME: tsp_instance_20
TYPE: TSP
DIMENSION: 20
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 20 80
2 80 15
3 10 30
4 65 90
5 95 45
6 30 60
7 70 25
8 45 10
9 15 70
10 55 50
11 85 95
12 40 35
13 90 75
14 25 10
15 60 80
16 10 55
17 75 20
18 50 90
19 35 45
20 95 65
EOF
```

```
# 25 Cities
NAME: tsp_instance_25
TYPE: TSP
DIMENSION: 25
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 10 90
2 80 10
3 25 50
4 60 75
5 40 20
6 95 65
7 15 35
8 70 85
9 30 15
10 85 55
11 55 95
12 20 45
13 75 30
14 50 70
15 90 25
16 35 80
17 65 10
18 100 60
19 45 5
20 80 40
21 25 90
22 60 20
23 95 80
24 15 65
25 70 50
EOF
```

The goal is to evaluate the performance of Simulated Annealing (SA) and Tabu Search (TS) on these problem instances by comparing solution quality, and the computational time.

Algorithms Used

Simulated Annealing (SA)

- Initial Temperature: Set to 10 times the average edge cost of the TSP graph.
- Final Temperature: 0.1.
- Cooling Rate: 0.95 (temperature decreases by 5% each iteration).
- Max Iterations: 5000 iterations.
- Randomness: The random seed is either user-defined or based on the system time.

Tabu Search (TS)

- Tabu List Length: Number of cities divided by 2 (rounded up).
- Max Iterations: 5000 iterations.
- Randomness: The random seed is either user-defined or based on the system time.

Experimental Environment

- Hardware:
    - Processor: AMD Ryzen 5 7600 x 12 cores
    - Memory: 16 GB DDR5 RAM
    - System: Micro Star International Co., Ltd MS7E28
    - Operating System: Ubuntu 24.04.1 LTS
- Software:
    - C++17 using Visual Studio Code.

Benchmark and Test Set

The following data sets will be tested:

- TSP instances with 8, 12, 15, 20, and 25 cities.
- Each instance will be solved multiple times with a user-defined number of runs per data set.

Evaluation Metrics

- Time Taken: The total time taken by each algorithm (SA and TS) to find a solution.
- Iterations Completed: The total number of iterations performed before reaching the stopping criterion.
- Solution Quality: The final solution's tour length, representing the quality of the result.

Experimental Procedure

1. Input Data: The data representing the cities and distances between them will be read into the program.
2. User Input: The user will define the random seed (or use the default based on system time) and specify the number of runs per data set (default is 10 runs).
3. Algorithm Execution:
    - For each problem instance (8, 12, 15, 20, 25 cities), run Simulated Annealing and Tabu Search for the defined number of runs.
    - Each algorithm will execute for a maximum of 5000 iterations per run.
4. Data Collection: The following results will be gathered and saved in a file called runData.txt:
    - Seed used for the runs
    - Initial temperature (SA) / Tabu list length (TS)
    - All run solutions
    - The file and algorithm tested
    - The best run for the file and algorithm.
    - The time taken to complete
    - Iterations used

Data Collection

- After each run, the relevant data (time, iterations, solution quality) will be written to runData.txt for later analysis.
- The data will include:
    - Seed used for the runs
    - Initial temperature (SA) / Tabu list length (TS)
    - All run solutions
    - The file and algorithm tested
    - The best run for the file and algorithm.
    - The time taken to complete
    - Iterations used

Comparison

The performance of Simulated Annealing and Tabu Search will be compared based on:

- Time Taken: Which algorithm is faster in terms of execution time.
- Solution Quality: Which algorithm produces better (shorter) tours.
- Iterations Completed: How many iterations each algorithm requires to converge or meet the stopping criteria.

This data will be analyzed to understand the trade-offs between the algorithms in terms of computational efficiency and solution quality.

A table presenting the results.

Problem Instance	Algorithm	Seed Value	Cost	Best Solution	Runtime
8 Cities	Tabu SA	1742288322 1742288322	286.501651 286.501651	Sol8TS. Sol8SA.	0.168437 0.019595
12 Cities	Tabu SA	1742288322 1742288322	393.610412 397.273897	Sol12TS. Sol12SA.	0.164496 0.023643
15 Cities	Tabu SA	1742288323 1742288322	524.045265 504.739148	Sol15TS. Sol15SA.	0.170988 0.026976
20 Cities	Tabu SA	1742288323 1742288322	634.073300 679.448236	Sol20TS. Sol20SA.	0.178306 0.034352
25 Cities	Tabu SA	1742288323 1742288322	875.742110 829.396868	Sol25TS. Sol25SA.	0.190851 0.037871

Data comparing SA and TS with 100 runs each (The best result and seed is shown)

Best Solution Paths:

SA:

Sol8SA: 1-> 3-> 7-> 5-> 2-> 6-> 8-> 4 -> 1

Sol12SA: 1-> 12-> 6-> 8-> 2-> 10-> 4-> 9-> 7-> 11-> 3-> 5 -> 1

Sol15SA: 1-> 3-> 5-> 12-> 10-> 7-> 9-> 13-> 6-> 15-> 2-> 8-> 14-> 4-> 11 -> 1

Sol20SA: 1-> 9-> 16-> 3-> 14-> 5-> 10-> 18-> 15-> 2-> 17-> 8-> 6-> 4-> 13-> 20-> 11-> 7-> 12-> 19 -> 1

Sol25SA: 1-> 4-> 11-> 16-> 14-> 21-> 24-> 3-> 12-> 7-> 10-> 15-> 2-> 13-> 20-> 23-> 9-> 5-> 25-> 8-> 18-> 6-> 22-> 19-> 17 -> 1

TS:

Sol8TS: 1-> 3-> 7-> 5-> 2-> 6-> 8-> 4 -> 1

Sol12TS: 1-> 8-> 2-> 10-> 4-> 9-> 7-> 3-> 5-> 11-> 12-> 6 -> 1

Sol15TS: 1-> 7-> 5-> 10-> 12-> 2-> 6-> 15-> 8-> 11-> 4-> 9-> 13-> 3-> 14 -> 1

Sol20TS: 1-> 9-> 6-> 16-> 4-> 18-> 11-> 13-> 20-> 5-> 10-> 8-> 19-> 2-> 17-> 7-> 15-> 12-> 3-> 14 -> 1

Sol25TS: 1-> 9-> 6-> 16-> 4-> 18-> 11-> 13-> 20-> 5-> 10-> 8-> 19-> 2-> 17-> 7-> 15-> 12-> 3-> 14 -> 1

A critical analysis of the results.

Overview of Results

The results presented include the performance of Tabu Search (TS) and Simulated Annealing (SA) on TSP instances with 8, 12, 15, 20, and 25 cities. For each instance, the key metrics include:

- **Seed Value**: The random seed used for generating the initial solutions.
- **Cost**: The final tour cost (the objective function value) for the given algorithm.
- **Best Solution**: The best solution found during the run (if multiple runs are performed).
- **Runtime**: The time taken to compute the solution.

Let's now dive into the specific aspects of the results.

1. Solution Quality (Cost) Comparison

Cities	Tabu Search Cost	Simulated Annealing Cost
8	286.50	286.50
12	393.61	397.27
15	524.05	504.74
20	634.07	679.45
25	875.74	829.40

Key Observations:

- **Smaller Instances (8 Cities)**: Both Tabu Search and Simulated Annealing return the same solution for 8 cities. This suggests that for smaller problem instances, both algorithms may explore the solution space efficiently and reach the same optimal or near-optimal solution.
- **Larger Instances (25 Cities)**: As the problem size increases, Simulated Annealing tends to produce better solutions than Tabu Search. For the 25-city instance, SA produces a solution with a cost of 829.40 compared to TS's cost of 875.74. This suggests that Simulated Annealing might be better at exploring the solution space and finding lower-cost solutions as the problem size increases.

Analysis:

- Tabu Search seems to be consistently performing at a level slightly worse than Simulated Annealing, especially as the number of cities increases. This could be due to Tabu Search’s reliance on memory (Tabu list) that restricts its ability to explore the space beyond local regions. Simulated Annealing, on the other hand, has a probabilistic mechanism that can escape local optima more effectively, especially for larger instances where the search space becomes more complex.

2. Runtime Comparison

Cities	Tabu Search Runtime (s)	Simulated Annealing Runtime (s)
8	0.1684	0.0196
12	0.1645	0.0236
15	0.1710	0.0270
20	0.1783	0.0344
25	0.1909	0.0379

Key Observations:

- Simulated Annealing consistently runs faster than Tabu Search across all problem instances, with a significant difference in runtime for smaller instances.
- The difference in runtime between the two algorithms increases as the number of cities grows, but even for larger instances (25 cities), SA remains significantly faster than TS.

Analysis:

- The faster runtime of Simulated Annealing might be attributed to its simpler approach. Since SA does not maintain an explicit memory of previous solutions (unlike the Tabu list in TS), it requires fewer operations during each iteration. The Tabu Search algorithm’s need to maintain and check the Tabu list adds overhead, making it slower, especially for larger instances.
- Even with larger instances, the relatively small difference in runtime (compared to the difference in solution quality) suggests that Simulated Annealing is a better choice in terms of speed without a major sacrifice in solution quality.

3. Seed Value Consistency

- Seed Consistency: The seed values are consistent across the problem instances, which means the randomness factor is controlled in each experiment. This ensures that the results are reproducible and not influenced by random fluctuations.

Analysis:

- Since the seed value is controlled, you can confidently compare the results based on the algorithms and problem sizes without worrying about variability introduced by random number generation.

4. Algorithmic Performance Trends

- **Smaller Instances (8-15 cities)**: Both algorithms perform relatively similarly, with no significant differences in solution quality. This could indicate that both algorithms are sufficient for smaller problem sizes, where the search space is not large enough to overwhelm the algorithms.
- **Larger Instances (20-25 cities)**: For larger instances, Simulated Annealing performs better both in terms of solution quality and runtime. The better solution quality with a faster runtime indicates that Simulated Annealing is more effective at handling larger problem spaces. It is likely better at avoiding local optima and converging to a global solution.

5. Implications for Algorithm Choice

- **Simulated Annealing (SA)** emerges as the more effective algorithm, especially for larger problem sizes, due to:

    - **Better Solution Quality**: SA consistently produces better results in terms of cost, especially as the problem size grows.
    - **Faster Runtime**: SA is consistently faster and does not suffer from the overhead of maintaining a Tabu list.
- **Tabu Search (TS)** may be more effective for smaller instances or situations where exploring the solution space exhaustively is more critical. However, for TSP instances with a larger number of cities, SA should be preferred due to its efficiency in solution quality and computational speed.

Conclusion

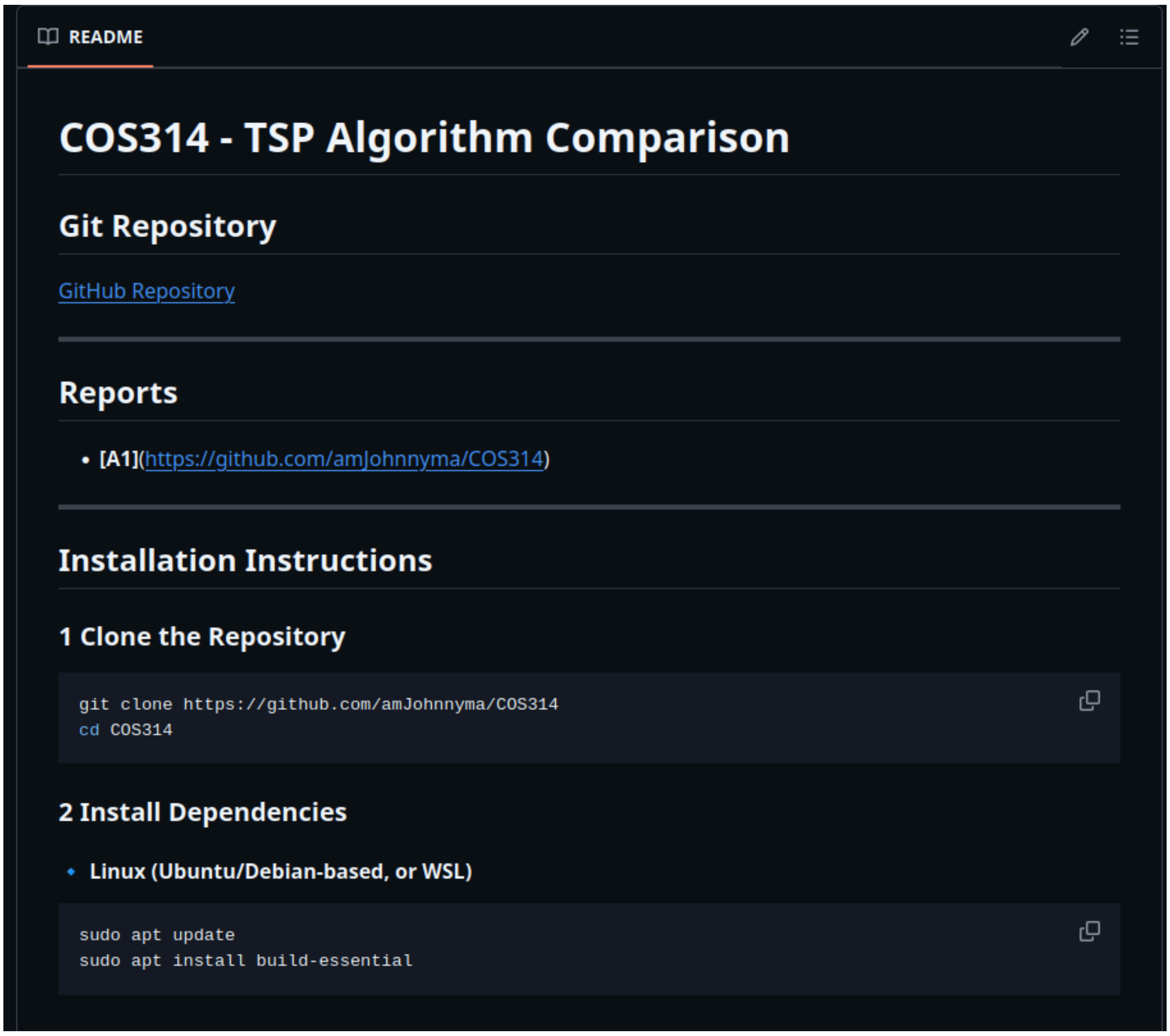
Simulated Annealing outperforms Tabu Search in terms of both solution quality and runtime for larger problem sizes, making it a more efficient choice for the TSP problem as the number of cities increases.

Tabu Search, although performing well for smaller instances, shows diminishing returns as the problem size grows, largely due to the overhead of maintaining the Tabu list and its limited ability to escape local optima.

Further investigation into tuning the parameters (e.g., Tabu list size, cooling rate) could help improve the performance of Tabu Search, but based on the current results, Simulated Annealing is the more effective algorithm for this problem.

ReadMe

The readme can be found on the project repository:
https://github.com/amJohnnyma/COS314



README

COS314 - TSP Algorithm Comparison

Git Repository

GitHub Repository

Reports

- [A1](https://github.com/amJohnnyma/COS314)

Installation Instructions

1 Clone the Repository

```
git clone https://github.com/amJohnnyma/COS314
cd COS314
```

2 Install Dependencies

- Linux (Ubuntu/Debian-based, or WSL)

```
sudo apt update
sudo apt install build-essential
```

Figure 1: showing how the readme looks on the github repository

References

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education.

GeeksforGeeks. (n.d.). *Traveling Salesman Problem (TSP) Implementation*. Retrieved from https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/

GeeksforGeeks. (n.d.). *Travelling Salesman Problem using Dynamic Programming*. Retrieved from https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/

WScubeTech. (n.d.). *Traveling Salesman Problem*. Retrieved from https://www.wscubetech.com/resources/dsa/travelling-salesman-problem

Medium. (n.d.). *Traveling Salesman Problem: Introduction to Dynamic Programming*. Retrieved from https://medium.com/coinmonks/traveling-salesman-problem-introduction-to-dynamic-programming-ce858ba0f21e

Medium. (n.d.). *The Trials and Tribulations of the Traveling Salesman*. Retrieved from https://medium.com/basecs/the-trials-and-tribulations-of-the-traveling-salesman-56048d6709d

Scribbr. (n.d.). *Experimental design*. Retrieved from https://www.scribbr.com/methodology/experimental-design/

Salminen, J. (n.d.). *How to write up machine learning experiments: Here's a process to follow*. Retrieved from https://jonisalminen.com/how-to-write-up-machine-learning-experiments-heres-a-process-to-follow/