Project Report

Visualisation and Analysis of Meta-heuristic search algorithms for the Asymmetric Travelling Salesman Problem

Jay Patel - ***300288159***

Siddhant Tiwari - ***300294512***

Table of Contents

1. Introduction

Travelling Salesman Problem is a problem which tries to find the shortest cost of travelling to all the cities from any chosen origin point when given a set of cities and the distances between every pair in the set. The asymmetric Travelling Salesman (ATSP) problem is a variant of the TSP where the cost function utilised in the TSP has different costs for travelling one way and the inverse travel for the path. It is classified as an NP-Hard problem which means that the cost of computing the solution will go higher as the sample space of the set of cities/points goes up.

ATSP is a part of the larger subset of Combinatorial Optimization problems. It aims to provide the best path to be taken for travelling between different points. Such optimization can be associated with generating the best path for a consumer for travel, deliveries and so on by companies like Google, Apple, Uber etc. The main significance of the ATSP can be seen in this statistic relating to last-mile delivery. Last-mile delivery refers to the transportation of goods from depots/warehouses to the customer's requested point of delivery and it is the leading cost driver in the supply chain. Companies spend an average of $10.1 on the last mile delivery whereas customers pay only $8.08 for this [1]. Thus this statistic establishes the relative importance of the ATSP problem.

ATSP can be solved using a variety of metaheuristic search techniques. Metaheuristic search techniques like tabu search and genetic search themselves have found wide applications in software testing and test automation. ATSP can be converted to TSP and then basic TSP search algorithms can be applied directly. Branch and Cut techniques - which converge from Branch and Bound and cutting planes have been applied in the recent past to solve ATSP [2].

2. Related Work

In this paper [3], we can see that there are modifications to the ATSP problem like the online ATSP with different algorithms and provisional solutions to solve the problem in polynomial time. In the online ATSP problem considered here, we only receive the order and sites to visit over time and the salesman has to decide the order of servicing these requests. This paper is the first research on competitive analysis of algorithms for online ATSP.

In this study [4], the author has taken a Machine Learning approach to the TSP problem. They propose RL as it performs better in use cases where an approximate solution is of more importance than a perfect solution and RL also holds up nicely in partially observed environments.

The experimental paper [2] which suggests the transformation approaches, Integer Linear Programming Models, and branch and cut methods. The transformation approach is borrowed from [5].

3. Research Questions

RQ1: What is the best algorithm to solve the ATSP?

RQ2: Can we transform TSP problems into ATSP?

RQ3: Does the best algorithm change depending on the problem?

4. Methodology

The way this project has been organised is as follows:

1. Preparation of Data
2. Transformation of Dataset
3. Calling the required search algorithm according to user input
4. Giving input regarding path length and number of steps
5. Generating a Graph for the Algorithm over a set of 50 points

We prepare the dataset for the problem by first processing it in vectors and creating a 2D matrix containing the edges between the data. These 2D matrices are then passed to the function ***transformATSPToTSP*** where 2-node transformation is performed. This transformed matrix is then passed on to whichever algorithm is selected by the user. These algorithms have also been embedded with the feature of random restarts so that we can try to avoid local optima. If the user requests the graph, then a graph is generated for the requested algorithm. We have answered *RQ2* here.

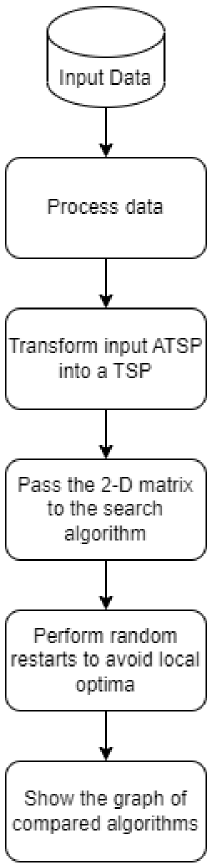

Input Data
Process data
Transform input ATSP into a TSP
Pass the 2-D matrix to the search algorithm
Perform random restarts to avoid local optima
Show the graph of compared algorithms

Figure 1: Project Flow Chart

4.1 Transformation from ATSP to TSP

According to [2] we can observe that the transformation helps solve occasional ATSP with the algorithm available for TSP and partly symmetric TSPs can be solved without the need to rely on special algorithms using assignment relaxation like Balas and Christofides for ATSP. These relaxation algorithms perform poorly on symmetric and almost symmetric problems.

TSPs with only a few weight asymmetries benefit the most from this transformation. Although the problem size is doubled due to the transformation, the lack of good algorithms for ATSP makes transformations the most feasible approach.

4.1.1 2 – Node Transformation

In the 2-node transformation from V to V' we replace every node with 2 nodes, and out of these one acts like an ingoing node and the other one acts like an outgoing node. For partially asymmetric TSPs we can use this transformation for doubling only the nodes which are a part of the asymmetric measure of V.

V is the set of given two points

No_of_total_vertices → 2n

The Updated costs of the edges were:

- Cost_of_edge (i,n+i) → 0 (i in V)
- Cost_of_edge (n+i,j) → c_{ij} [i,j in V]
- Rest of the edges → ∞

4.1.2 3 – Node Transformation

Just like 2 node transformation, we have also tried 3 node transformation as proposed by Karp and Richard M. [6].

No_of_total_vertices → 3n

Here, again as we consider for two node transformation, V is the set of the given two points

The updated costs of the edges were:

- Cost_of_edges (i,n+i) and (n+i,2n+i) → 0 (i in V)
- Cost_of_edge (2n+i,j) → c_{ij} (i,j in V)
- Rest of the edges → ∞

4.2 Representation and Fitness Function:

The given dataset of the adjacency matrix is processed as a 2-D matrix of ATSP which is then transformed to make a TSP matrix that is utilised as input data. The fitness of each solution we get is computed by finding the total distance travelled by the salesman and it is used to compare two different solutions.

4.3 Algorithms Implemented

Now, to make a comparative analysis of the best algorithm for the Asymmetric Travelling Salesman Problem, we planned to implement four meta-heuristic algorithms, namely: First Ascent Hill Climbing, Steepest Ascent Hill Climbing, Simulated Annealing, and Genetic Search algorithm.

In the very first step, we preprocessed our dataset by considering two steps given below:

- Firstly, converted the '.atsp' file to the 2D array in Java which represents the distance from one node to the other node.
- Now, as we are dealing with asymmetricity in the problem, we transform the 2D matrix to make it a symmetric travelling salesman problem. The above method states how a 2-node and 3-node transformation is done on ATSP.

The remainder of this section provides the pseudo code and explanation for each of the implemented algorithms.

4.3.1 First Ascent Hill Climbing Algorithm

Description:

First Ascent Hill Climbing is a basic algorithm where a solution of the total distances will be randomly generated and then the algorithm will keep generating other random solutions until a better one is found. The first better option is declared as the final solution.

Advantages:

The main advantage is that the hill climbing algorithm is that it doesn't use a lot of computation power and it doesn't consume a lot of time either.

Disadvantages:

The biggest drawback of this algorithm is that the algorithm is bound to find the local minimum. In most cases, it is highly probable that the algorithm will find a better solution and declare it as the best one.

Implementation:

Following are the steps implemented by us in "FirstAscentHillClimbing.java" file for the first ascent hill climbing algorithm:

1. currentAnswer → Calling the generateRandom() method where we can get a permutation (from 0 to n - 1, n = total number of cities). It is in the form of a Vector (datatype: integer).
2. currentDistance → Finding the path length of the currentAnswer using getTotalDistance() definition.
    a. getTotalDistance() → It uses the adjacency matrix (ATSP) to find the total distance which is travelled for currentAnswer path (ATSP). (here, ATSP is the transformed matrix by 2 - node transformation to TSP).
3. neighboursOfCurrentAnswer → Here we store all the possible neighbours of the randomised solution (currentAnwer) which we assumed using the generateAllNeighbours() function.
    a. generateAllNeighbours() → This function takes O(n^2) time complexity, n = number of cities. Because it swaps all possible pairs of indices to make it a neighbouring solution.
4. bestNeighbour, bestNeighbourDistance → It makes a call to findBetterAnswer() method for getting the best possible solution. It returns the path for the best solution from all neighbours and by calling getTotalDistance() method, we can get the distance travelled.
    a. findBetterAnswer() → It iterates to all the neighbours of our currentSolution and returns the best answer (which has the smallest distance for ATSP).
    b. Once we get a better neighbour than the currentSolution, then we simply return that neighbour by terminating the loop.
5. Now, we run an infinite loop until bestCurrentDistance is less than currentDistance. In this loop, we:

    a. Reiterate steps 3 and 4.
6. Lastly, to get the final answer.
    a. Now, since there is a probability for it to stick to local optima, we then optimise our implemented algorithm. We then use an advanced mutation operator, i.e., the usage of random restarts. We iterate our solution for n number of times (n can be any positive integer). So, in every iteration, we start from a new random solution to get the optimised solution.

4.3.2 Steepest Ascent Hill Climbing Algorithm

Description:

Steepest Ascent Hill Climbing is an optimization of first ascent hill climbing where instead of searching for one better solution, the algorithm explores all the neighbours of the current state and then it finds the best solution among them all.

Advantages:

Steepest Ascent Hill Climbing is better than the base hill climbing algorithm as it considers a larger solution space (all the neighbouring nodes) and explores it for the best solution instead of selecting the next best solution and terming it as the final answer.

Drawbacks:

Although this algorithm is an upgrade from the first ascent, it does again suffer from the problem of local optimums. It has a lesser probability of getting stuck on the local minimum as it searches for multiple neighbours yet in the case of a local minimum the algorithm is unable to escape it. We have implemented a modification to avoid this case which is provided at a later stage in this report.

Implementation:

Following is the implementation of the Steepest Ascent Hill Climbing Algorithm:

1. As the implementation of the steepest ascent hill climbing is very similar to the first ascent hill climbing we have just listed the changes required. In the first ascent algorithm, we just have to tweak the 4th step to achieve the steepest ascent algorithm as follows:
    a. Every time we get a better solution, our answer is updated to that solution. That is, unlike the first ascent hill climbing algorithm, here we do not break the loop once a better solution is found.

4.3.3 Simulated Annealing Algorithm

Description:

Simulated Annealing is an optimization of steepest ascent hill climbing where instead of searching for just a better solution, this algorithm also explores some of the solutions (with a condition explained in implementation) whose fitness function is less than the best solution so far.

Advantages:

In general, Simulated Annealing is better than the first ascent and steepest ascent hill climbing algorithms because this minimizes the chances of getting stuck into local optima. Because we are exploring different routes also which may reach to global optima.

Drawbacks:

The biggest issue in this algorithm is the cooling factor. That is, the cooling must be very slow in order to explore as many different solutions as possible. However, it can take a long run time. This algorithm requires a lot of parameters to be tuned.

Implementation:

All the steps are almost similar to the steepest ascent hill climbing to implement this algorithm, but there are a few changes in the findBetterAnswer() method which are:

1. Pre-define the cooling temperature T.
2. Re-iterate all the possible neighbouring solutions with the following steps:
3. If the currentDistance is better than the bestCurrentDistance we have so far, then update the bestCurrentDistance. (This is the same thing as done in the steepest ascent hill climbing algorithm).
4. Else, create a variable differenceDelta which represents the difference between the fitness function of the current and the fitness function of the best solution. This is then divided with the cooling temperature to return its exponential.
5. Now, the exponential output is compared with the random(0, 1). So, if the exponential output is greater than random(0, 1), then the currentBest is updated (this step can try to reduce the chances of getting stuck in local optima.
6. Then, the temperature T is cooled.

4.3.4 Genetic Algorithm

Description:

A genetic algorithm is a search algorithm that is based on Darwin's theory of evolution. In this algorithm, firstly natural selection is done from the fittest parents. Then, the offspring are created to generate mutation and the next generation [7].

Advantages:

The method may tackle issues that are hard for other search algorithms to address since it is based on natural selection and evolution. As a result, it produces global optima or better outcomes and is less prone to become locked in local optima.

Drawbacks:

Particularly when compared to other optimization techniques, the genetic algorithm may be rather sluggish. They can also be challenging to comprehend and analyse, which makes it challenging to grasp why a specific solution was discovered. Finally, they may be resistant to the original circumstances and need help to identify the global optima.

Implementation:

In this algorithm, the following steps are involved:

1. Generate the initial population to start the search process. Here, we generate n populations called generationSize.
2. Now, select the first random population and start the process which is called globalBest.
3. We do the following steps now for the maximum iteration times.
4. Selection: Here, we determine and is used to identify and pass on the positive traits from one generation to the following.
    a. We are doing tournament selection. In tournament selection, the winner is chosen from among k randomly chosen genomes that will take part in the competition. The likelihood that a genome will prevail increases with fitness.
    b. We took the reproduction size as 200 which we call the tournament selection function.
5. Generate offspring: A list of generation size is created by the implementation of the below steps:
    a. Firstly, a list of parents is generated by picking N random elements.
    b. Crossover: In the crossover, we produce a number of child genomes from two-parent genomes that were selected via selection. The process for combining the two genomes can vary significantly, but in general, we take some genes from one parent and some from the other.
    c. Mutation: In order to reduce the chances of getting stuck in early convergence, we mutate genomes with a specific probability to select new ones. Here, we call the mutation rate. If the mutation rate is very high, then it will be the same as a random search. In case it’s lower, early convergence may come. So, we have to select the mutation rate in that manner.
6. After generating the population, we select the best genome.

5. Data Set

We have used “MP-TESTDATA - The TSPLIB Asymmetric Travelling Salesman Problem Instances” as our data set [8]. It is available on TSPLIB. TSPLIB is a library where we can find the dataset for all the problems which are related to the Travelling Salesman Problem. There are about 20 files in the dataset with optimal answers. For example, the optimal known answer for the file ‘br17.atsp’ is 39 units.

For simplicity, we have used the ‘br17.atsp’ file as input to compare the results of the implemented algorithm. The dataset has a few lines at the beginning like:

- NAME: br17
- TYPE: ATSP
- COMMENT: 17 city problem (Repetto)
- DIMENSION: 2
- EDGE_WEIGHT_TYPE: EXPLICIT
- EDGE_WEIGHT_FORMAT: FULL_MATRIX
- EDGE_WEIGHT_SECTION
- Then, the 2 – dimensional matrix shows the distance among all the nodes.

So, we preprocessed the input file to a 2D array in our Java code. This array will be used as a distance matrix to get the results of implemented algorithms. That means the user has to provide the input data in the same format to get the output.

6. Technologies used

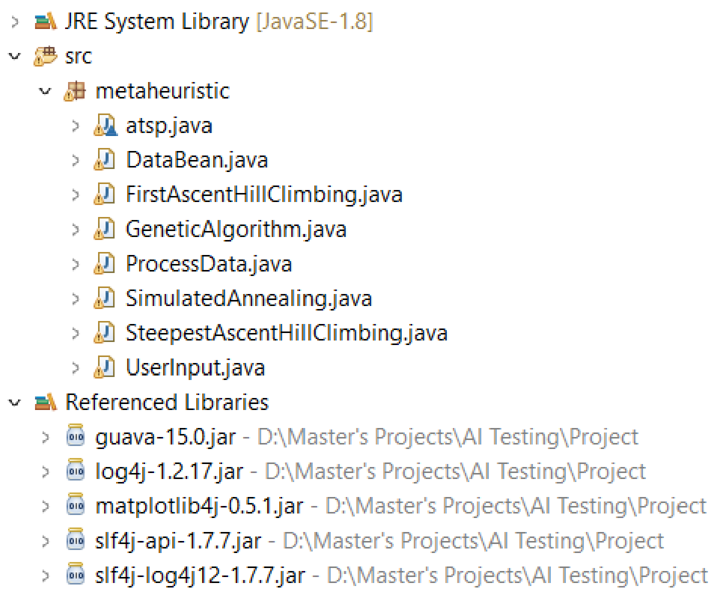
The technologies used to implement the project are stated below:

- Java: We have used Java programming language to implement the algorithms and process the input file.
- Swing: This is used to make the User Interface (UI) for the project.
- matplotlib4j: It is needed in our project to represent the comparison graph.
- Eclipse: We wrote our code in Eclipse IDE and imported the required '.jar' files for using matplotlib4j here.

7. How to Run?

The following points represent the steps involved in running the implemented code:

- Download JDK 8
- Download Eclipse (the latest version)
- Open Eclipse → File → Open Projects from File System
- Browse to the submitted code folder
- Press the command 'Ctrl + F11' to run the User Interface, or
- Go to the Navigation bar → Run the atsp.java file. Fig. 1 shows the project structure.
- If the above steps do not work, then one can run the ATSP.jar file which is a runnable jar file.
- For that, open the command prompt
- Write the command java -jar ATSP.jar and press Enter.



```
> JRE System Library [JavaSE-1.8]
v src
  v metaheuristic
    > atsp.java
    > DataBean.java
    > FirstAscentHillClimbing.java
    > GeneticAlgorithm.java
    > ProcessData.java
    > SimulatedAnnealing.java
    > SteepestAscentHillClimbing.java
    > UserInput.java
v Referenced Libraries
  > guava-15.0.jar - D:\Master's Projects\AI Testing\Project
  > log4j-1.2.17.jar - D:\Master's Projects\AI Testing\Project
  > matplotlib4j-0.5.1.jar - D:\Master's Projects\AI Testing\Project
  > slf4j-api-1.7.7.jar - D:\Master's Projects\AI Testing\Project
  > slf4j-log4j12-1.7.7.jar - D:\Master's Projects\AI Testing\Project
```

Figure 2: Project Structure

- This will open the UI wherein the algorithms are listed with a checkbox
- Checkmark the algorithms you want to compare and then select the input file
- Press the 'Run' button and wait for the results
- The output frame will have the results (or one can see the output in the command prompt) with an option to see the comparison graph

8. Results

In our comparisons between the different algorithms, we found that genetic search performs the best out of the lot. The following graph shows a detailed comparison between different algorithms and the shortest path lengths they found over a period of 50 iterations.

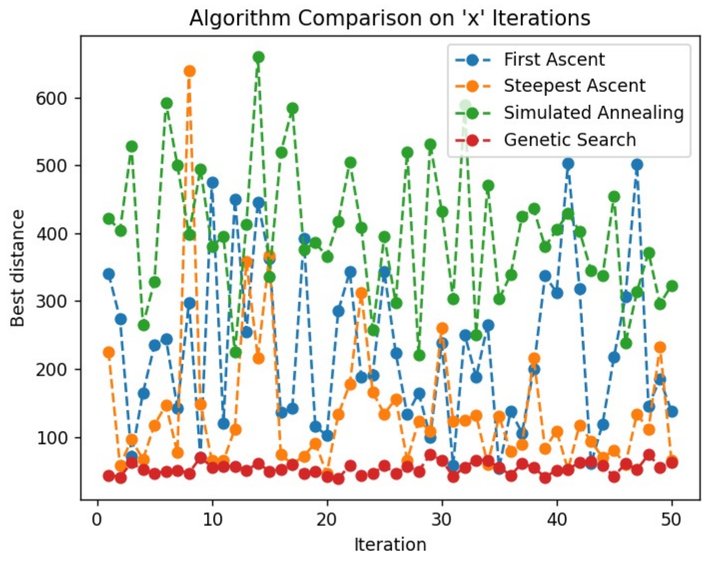
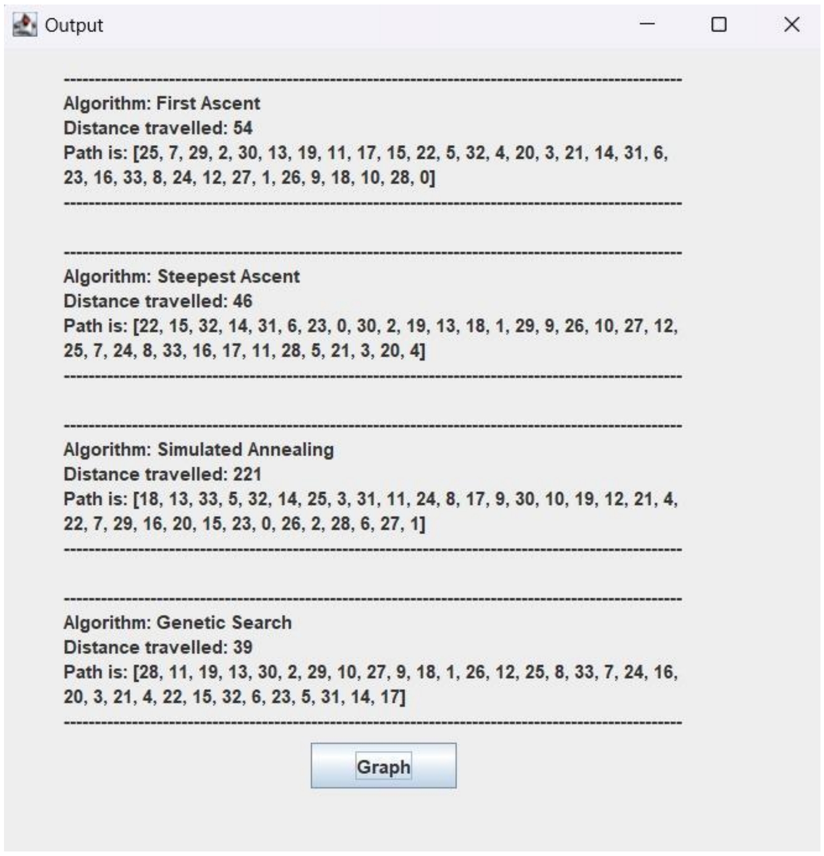


Figure 3: Comparison of results obtained from Heuristic Search Algorithms over 50 iterations

On the dataset br17.atsp, genetic search gives the shortest distance of 39 and that too consistently remains under the shortest value of 50.

Also, the steepest ascent is found to be the next best algorithm which again performs considerably better than the other two and reaches a minimum path value of 46. First ascent and Simulated annealing are not as efficient as the other two models listed above.

Thus we find the answer to *RQ1* by selecting Genetic Search as our optimal search algorithm here. The genetic algorithm also reaches the optimal answer 39 that is provided for the dataset in [8].



```
------------------------------------------------------------------------------------------------
Algorithm: First Ascent
Distance travelled: 54
Path is: [25, 7, 29, 2, 30, 13, 19, 11, 17, 15, 22, 5, 32, 4, 20, 3, 21, 14, 31, 6,
23, 16, 33, 8, 24, 12, 27, 1, 26, 9, 18, 10, 28, 0]
------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------
Algorithm: Steepest Ascent
Distance travelled: 46
Path is: [22, 15, 32, 14, 31, 6, 23, 0, 30, 2, 19, 13, 18, 1, 29, 9, 26, 10, 27, 12,
25, 7, 24, 8, 33, 16, 17, 11, 28, 5, 21, 3, 20, 4]
------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------
Algorithm: Simulated Annealing
Distance travelled: 221
Path is: [18, 13, 33, 5, 32, 14, 25, 3, 31, 11, 24, 8, 17, 9, 30, 10, 19, 12, 21, 4,
22, 7, 29, 16, 20, 15, 23, 0, 26, 2, 28, 6, 27, 1]
------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------
Algorithm: Genetic Search
Distance travelled: 39
Path is: [28, 11, 19, 13, 30, 2, 29, 10, 27, 9, 18, 1, 26, 12, 25, 8, 33, 7, 24, 16,
20, 3, 21, 4, 22, 15, 32, 6, 23, 5, 31, 14, 17]
------------------------------------------------------------------------------------------------
```

Figure 4: Output path travelled for each search algorithm

3 node transformation was also tried in our implementation but it yielded impractical results tending to infinity and also took a lot of time to run.

9. Conclusion

Our project revolves around the transformation of ATSP to TSP. Although the problem set is doubled, we can still observe that transformation is one of the best approaches to solving ATSP. Transformation of ATSP into TSP is plausible as TSP problems can get only as complex as ATSPs.

Another important conclusion we can make is that selecting the most optimal heuristic search algorithms depends a lot on the problem at hand. Simulated annealing algorithm which takes into fact additional parameters like temperature reduction to overcome local optima. But yet we see that the first ascent and steepest ascent both perform better than simulated annealing algorithms.

10. Project Demo Video

One can see the demo video of our project by visiting the following URL:

https://jay-patel-07.github.io/portfolio/atsp.html

References

1. S. Ma, “Solving the travelling salesman problem for deliveries,” *1_tPDQSBTaq5LPFXkdh9QlFA@2x*. [Online]. Available: https://blog.routific.com/travelling-salesman-problem.
2. Roberti, R., Toth, P. Models and algorithms for the Asymmetric Travelling Salesman Problem: an experimental comparison. *EURO J Transp Logist* 1, 113–133 (2012). https://doi.org/10.1007/s13676-012-0010-0.
3. G. Ausiello, V. Bonifaci, and L. Laura, “The on-line asymmetric traveling salesman problem,” *Journal of Discrete Algorithms*, 09-Apr-2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570866707000172.
4. “Traveling salesman problem with reinforcement learning,” *Traveling Salesman Problem with Reinforcement Learning - Amazon SageMaker Examples 1.0.0 documentation*. [Online]. Available: https://sagemaker-examples.readthedocs.io/en/latest/reinforcement_learning/rl_traveling_salesman_vehicle_routing_coach/rl_traveling_salesman_vehicle_routing_coach.html.
5. Jonker, R., & Volgenant, T. (1983). Transforming asymmetric into symmetric travelling salesman problems. *Operations Research Letters*, *2*(4), 161–163. https://doi.org/10.1016/0167-6377(83)90048-2.
6. Karp, Richard M. “Reducibility Among Combinatorial Problems.” 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, 2009, pp. 219–41, https://doi.org/10.1007/978-3-540-68279-0_8.
7. Traveling Salesman Problem with Genetic Algorithms in Java. (2019, July 24). Stack Abuse. https://stackabuse.com/traveling-salesman-problem-with-genetic-algorithms-in-java/.
8. MP-TESTDATA - The TSPLIB Asymmetric Travelling Salesman Problem Instances: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html