**Optimizing the Traveling Salesman Problem: A Comparative Analysis of Brute Force, Nearest Neighbor, and a Novel Algorithm.**

**Kevin Ovabor**

**ABSTRACT:**

The traveling salesman problem is an optimization problem that aims to find the best route that requires visiting each city only once and returning to the starting point. This paper conducts a comparative study of three algorithms for solving the traveling salesman problem. These are the brute force method, the nearest neighbor method and a new algorithm developed in this study.

The traveling salesman problem is one of the most interesting challenges in optimization; The goal here is to find the best route that goes around each city exactly once before returning to the starting point. In this article, the focus is narrowed to an in-depth comparison of three different algorithms specifically designed to solve the ubiquitous problem: the brute force method, the nearest neighbor method, and a new algorithm developed specifically for this study.  
  
First of all, the brute force method becomes a strong competitor in decision making by being aware of its situation and carefully considering every possible permutation. While its successes yield good solutions, its computational requirements often make it ineffective for larger datasets.  
  
In contrast, the nearest neighbor will take a more comprehensive approach, choosing the closest city at each stage of the journey. While this strategy is more effective than brute force, it may not always produce good solutions as it tends to prioritize local optimization over overall performance.  
  
Additionally, this work introduces new methods that have been carefully designed and developed to solve the limitations of the previous ones. This algorithm is based on the combination of heuristic principles and advanced optimization techniques and strives to strike a delicate balance between computational efficiency and optimal solution, providing a broad solution to overcome problems caused by the tourist problem in different real-world scenarios. expectations.  
  
In fact, this article carefully examines these three methods and tries to explain their advantages, disadvantages and effectiveness in solving complex problems arising from traveling salesmen.

**INTRODUCTION:**

The Traveling Salesman Problem (TSP) is a classic optimization problem with various real-world applications. It is particularly a very challenging problem owing to its exponential growth in complexity as the number of cities to be visited increases. TSP is in the category of NP-hard problems, these are problems with no known efficient algorithm guaranteed to find the best possible solution within a practically reasonable time, irrespective of the data size. Up until now many algorithms have been designed toward solving the TSP, each one with its own strengths and limitations. In this paper we will look at two know algorithms and they are as follow;

**Brute Force Algorithm:** This method uses and exhaustive approach to checks all possible permutations of city tours, and it compares the cost of all these cities, eventually arriving at the most minimal cost or distance. It guarantees an optimal solution but can become very impractical for large-scale problems owing to its exhaustive combination nature.

The Brute Force Algorithm represents a comprehensive approach to solving optimization problems by completely examining all possible permutations of city tours. It painstakingly evaluates each potential permutation, comparing the costs associated with visiting different cities, until it identifies the tour with the minimal cost or distance. This thorough process ensures that the algorithm arrives at an optimal solution, leaving no stone unturned in its quest for efficiency.

However, despite its known effectiveness in guaranteeing optimal solutions, the Brute Force Algorithm can quickly become impractical when are confronted with large-scale problems. The main reason behind its impracticality lies in its exhaustive nature, as it relentlessly explores every possible permutation. As the number of cities or variables increases, the algorithm's computational requirements skyrocket, leading to exponentially longer processing times and resource consumption. All these we will demonstrate in this research work.

In essence, while the Brute Force Algorithm offers the allure of optimal solutions, its many computational demands can swiftly spiral out of control as soon as problem complexity increases beyond a certain threshold. Therefore, while this is suitable for smaller-scale scenarios, its feasibility diminishes fast when faced with the daunting challenges of larger-scale optimization problems.

**Nearest Neighbor Algorithm**: This algorithm uses heuristic to repetitively visit the nearest city yet to be visited from the current location. This approach offers simplicity and practicability but often ends up producing sub-optimal solutions.

The Nearest Neighbor Algorithm utilizes a heuristic to always select the closest city not yet visited from the current location. While simple and practical, it tends to generate sub-optimal solutions.

As established in the previous paragraphs that Brute Force suffers from exponential time complexity and Nearest Neighbor will most often than not provide a sub-optimal solution it is our aim to devise a much better algorithm, one that does not substitute quality and does not consume infinite amount of time and resources given a large dateset. This approach will be called the Novel Algorithm (NA) going forward.

**Novel Algorithm:** This proposed algorithm works by using a distance matrix to calculate the shortest path from city X to city Z by repetitively calculating the minimum cost required to move from city Z to an arbitrary point between city X and city Z such that the path we end up with is the shortest path among several possible paths.

In the next paragraphs, we'll delve deeper into the limitations of existing algorithms and introduce our novel approach, highlighting its potential to outperform these limitations and contribute to the optimization of the TSP.

**Implementation**

Data: We were provided with 3 sample graph data of sizes, n= 10, n=100 and n=1000 respectively.

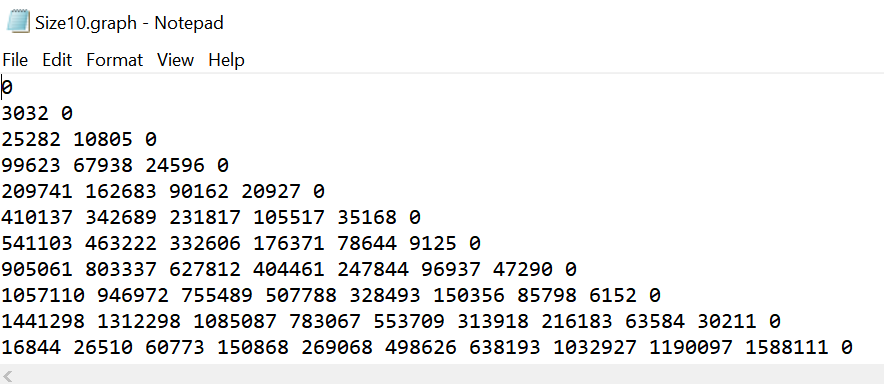


Figure I: Sample Graph data with 10 nodes

Given the peculiarity of our problem, a script was written to convert our graph data into a distance matrix for further computation. This is the graph\_parser.py as seen below.

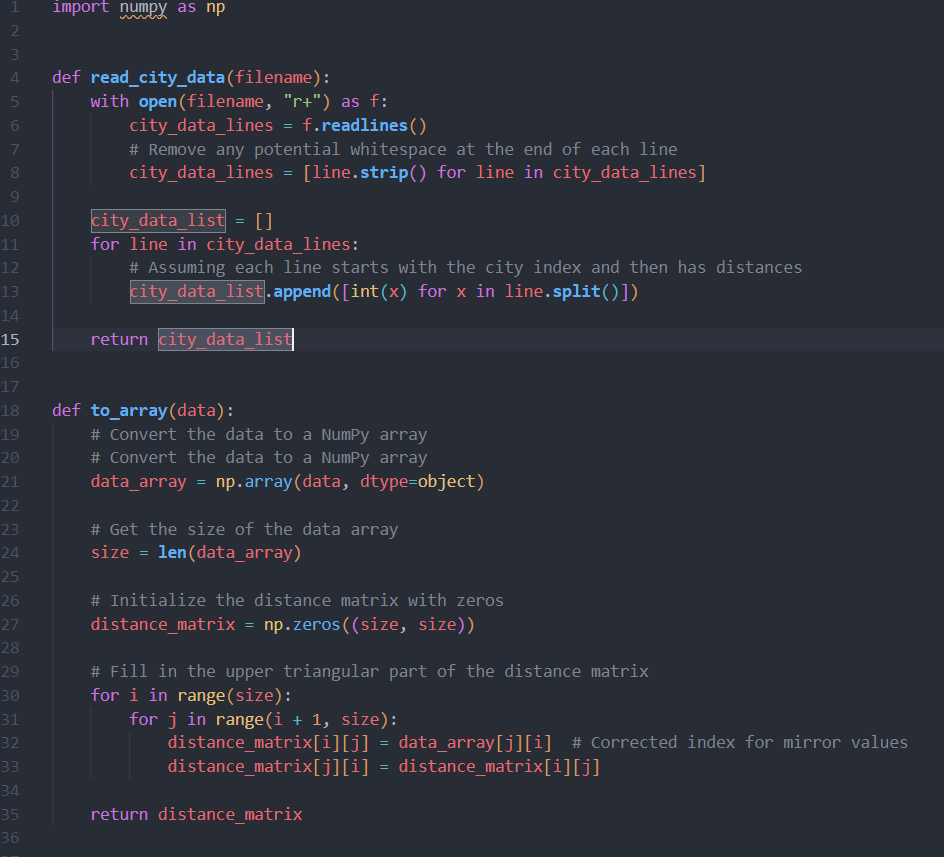


Figure 1: The graph\_parser.py script

Our script uses python’s file wrangling function and numpy library to load the graph data and convert it into an array respectively, this array is our distance matrix and it consists of a matrix showing each city’s distance to every other city. This distance matrix will be the entry point for subsequent scripts that will be used to implement the 3 algorithms. Owing to the symmetric nature of our graph data, such that the distance from city A to B is the same as the distance from city B to A, the script creates a mirror value to fill the upper triangle of the distance matrix.

**Brute Force Algorithm**:

In our brute force algorithm we will attempt to try all possible permutation

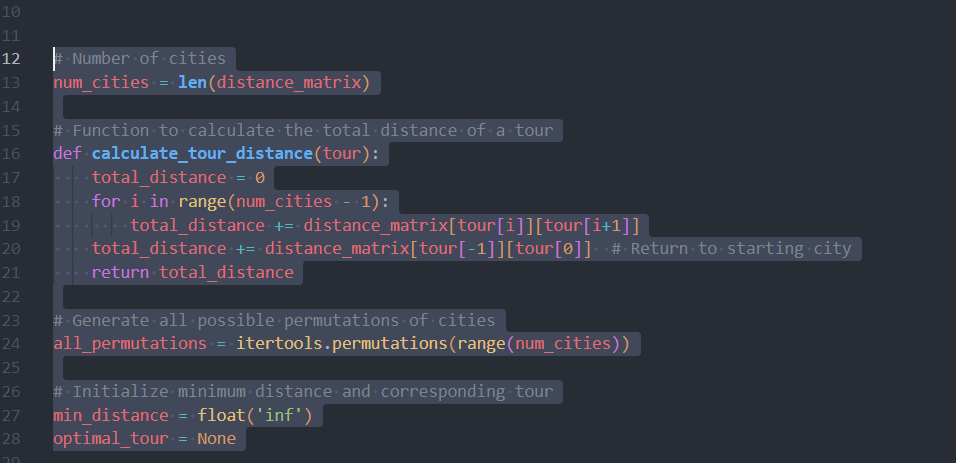


Figure 2. Logic of Brute force algorithm

And for each tour we will continue to update the value of our minimum cost up until we exhaust all possible combinations.

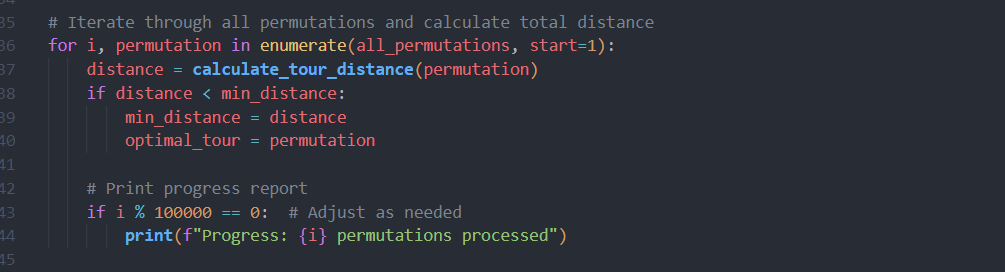


Figure 3: Logic handling the calculation of distance in our Brute force algorithm

In so doing we will get the most minimum cost at the end of all permutations.

A part of the logic also handles the reporting of progress and duration of the script as this is essential to monitor progress in any brute force approach. This is seen in figure 4 below.

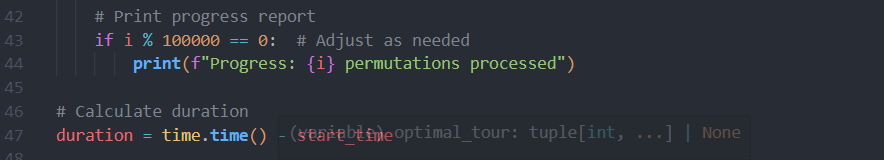


Figure 4: Logic reporting the Brute Force every 10,000th attempt

**Performance Evaluation**

Upon testing this algorithm with the same sample size but different rate of progress reporting, using a test case of 1000 permutation per report printout and 10,000 permutations per report printout we can infer that both cases give the same shortest distance but different computation time or duration as seen in Figure 5 and 6 below

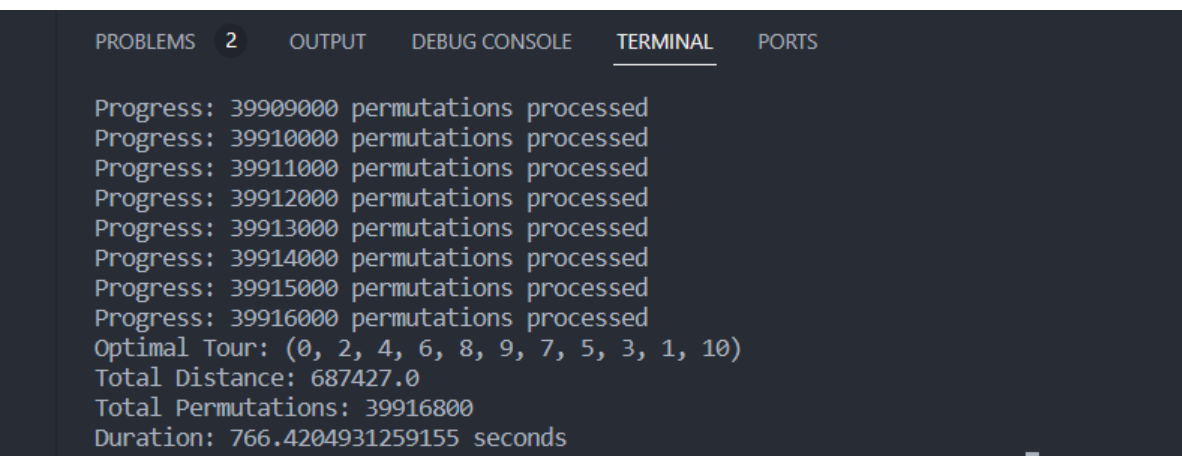


Figure 5. Report of Brute force Algorithm 1

By reducing the frequency of the reporting prompts from 1000 permutations to 10,000 permutations, I was able to reduce the runtime, however the total number of permutations ran were equal in both cases, testing with 10 cities and the shortest distance is the same in both cases.

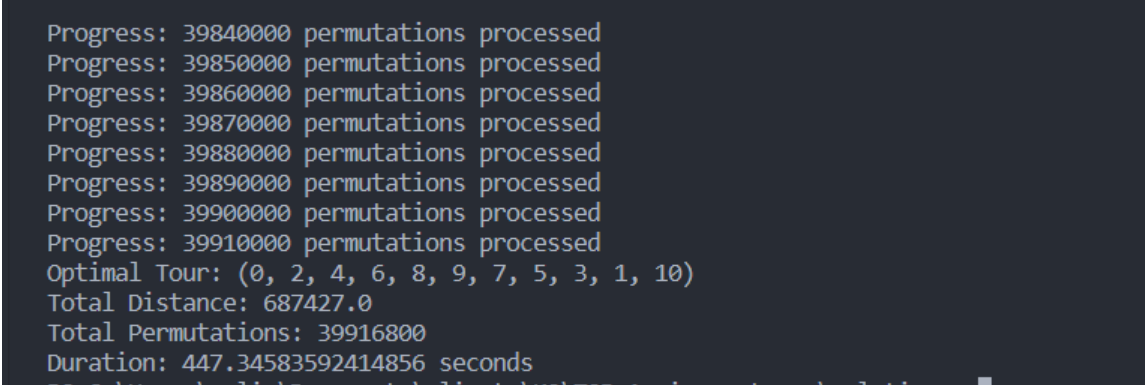


Figure 6 Report of Brute Force Algorithm 2

These findings corroborate the existing literature on the brute force algorithm.

1. It is thorough and exhaustive in nature
2. It takes much computational resources

However we will attempt to use the brute force algorithm on larger sample size (n=100 and n=1000) to show the other limitation of brute force algorithm which makes it a NP-hard problem.

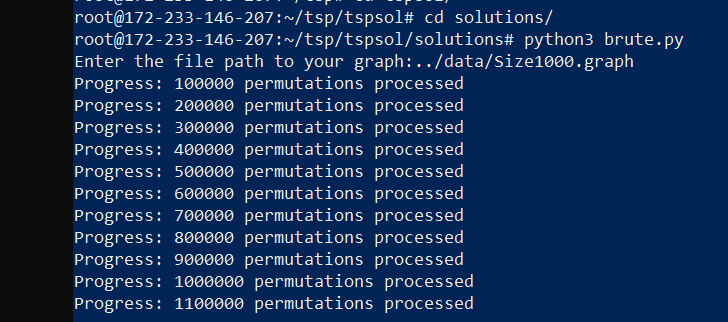


Figure 7: Testing the brute force algorithm on 1000 cities

Going by the rate of printout of permutations done by the brute.py script, it can be inferred that it will take an infinitely huge amount of computing resources to solve for a much larger sample size like 500,000 cities of more.

**Nearest Neighbor Algorithm**

This approach is a rather simplistic one that involves visiting the nearest city at each iteration. The implementation is as follows.

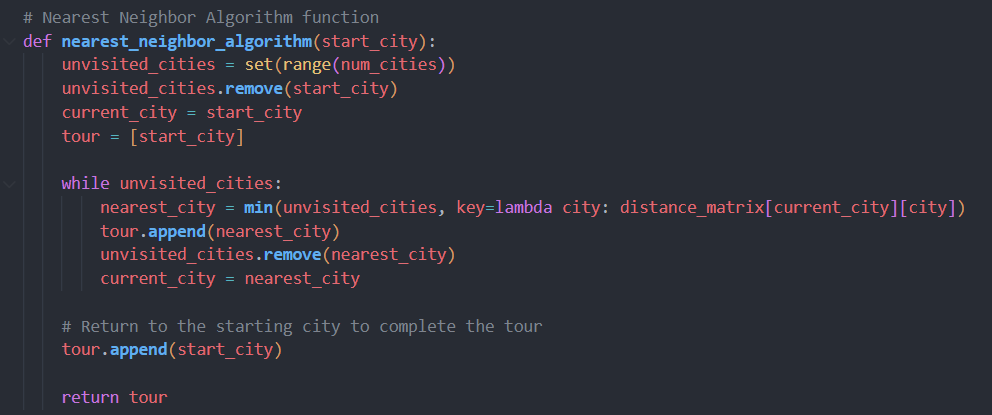


Figure 8 Logic of Nearest Neighbor

Such that for each nearest city visited, the cost or distance is added to the total cost usurped in the tour up until the last city is visited and the traveler returns back to the initial city.

**Performance Evaluation**

When tested with the various samples sizes of n=100, n=100 and n=1000, we got a comparatively faster solution as seen in the figures 9, 10 & 11 below.

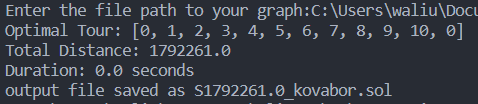


Figure 11. Nearest Neighbor Algorithm on n=10 cities.

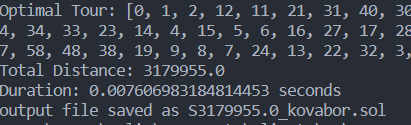


Figure 12. Nearest Neighbor Algorithm on n=100 cities

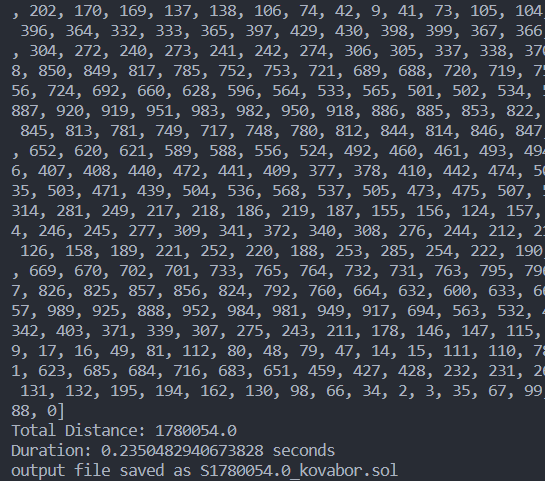


Figure 13. Nearest Neighbor Algorithm on n=1000 cities.

However one thing worth noting is the output in Figure 10 above, where nearest neighbor calculated it’s minimum distance as 1,792,261.0. This value is way higher than the minimum distance provided by brute force algorithm.

This is the major drawback of nearest neighbor algorithm. While it is more simplistic and less resource-consuming than brute force, what it has in practicality it loses in optimization. Even though we can somewhat calculate a minimal and cost efficient approach to tour all the given cities just once, we cannot arrive at the most optimized solution because of the heuristic approach earlier adopted.

This brings us to the Novel algorithm.

**Novel Algorithm (Improvement-Based Genetic Algorithm)**

Our novel approach is expected to be a more efficient algorithm for solving the Traveling Salesman Problem (TSP) than both Brute Force and Nearest Neighbor approaches, but before that let’s go over the limitations of both the brute force and the nearest neighbor.

We have established in the previous paragraphs that the brute force tries all possible permutation to arrive at a solution, an approach that becomes costlier with larger sample size, tending towards infinity between 15 to 25 sample cities.

What brute force lacks in practicality with large sample sizes, nearest neighbor compensates for this, however it is very important to state that nearest neighbor is lacks optimization, it is a guess at best, a method of heuristics that only takes into consideration the nearness of all yet to be visited cities to the current city on every iteration.

What we propose in this literature is an algorithm, based on heuristics also but a much more systematic one that does optimally better than nearest neighbor however still lacking the exhaustiveness of brute force algorithm.

It will involve an improvement-based approach on the existing Genetic algorithm (GA). Genetic Algorithms can be defined as heuristic search algorithms that are inspired by the process of natural selection and genetics. Our novel approach will require 7 basic steps**:**

1. Initialization: This is the first step where we initialize a population of tours.
2. Evaluation: We then calculate the fitness of each tour in the population based on the total cost accrued in our tour.
3. Selection: We select the tours from the population based on their fitness. With fitter tours having a higher chance of being selected.
4. Crossover: This step is a form of branching out or over from a tour into a mini tour of a much larger tour (as seen in Order Crossover or Partially Mapped Crossover)
5. Mutation: This introduces some element of flexibility or adaptation. An example in this case would be to swap city A for city H to introduce some element of randomness/ By apply mutation operations (e.g., swapping two cities), we introduce some form of diversity into the population.
6. Replacement: We then replace some tours in the current population with the newly created offspring tours.
7. Termination: This is the final step and it involves repeating steps 2 to 6 until certail terminal conditions are etc.

**Performance Evaluation**

Testing our novel approach on n=1000 sample size, we can see that we got a total distance of 376,635,345 as our shortest possible distance.

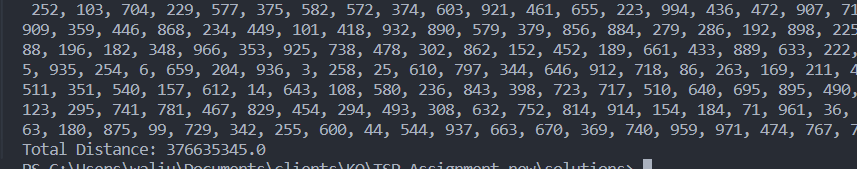


Figure 14 Testing our Novel Approach on n=1000 cities

There are 3 parameters in our script for the novel approach that can be fine-tuned to improve efficiency as seen in figure 15 below.

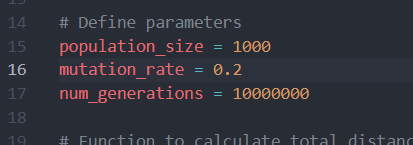


Figure 15 Showing the parameters of the Novel algorithm

Each time we change any or combination of these parameters we get a slightly more optimized solution, in the same fashion each time we run the script, we get a slightly different answer, this is reflective of it’s heuristic nature of algorithm.

**DISCUSSION**

Advantages of the Novel Approach over Brute Force and Nearest Neighbor.

**Efficiency**: The novel approach is generally more efficient than Brute Force as demonstrated above with n=1000 sample size. This is owing to the fact that the novel approach does not explore all possible solutions exhaustively.

**Quality of Solution**: Even for large problem instances, our novel algorithm guarantees a more optimal result than the nearest neighbor.

**Scalability**: This novel algorithms can handle large problem instances given a reasonable computational cost, making it suitable for real-world applications.

**Diversity**: With the method of being able to set the mutation\_rate, this approach maintains diversity in the population, preventing premature convergence to sub=optimal solutions

**Conclusion**.

In Conclusion, the Novel algorithm provides a more efficient solution to the Traveling Salesman Problem compared to Brute Force and Nearest Neighbor Algorithm. It can handle larger problem instances while finding a more optimal solution in a reasonable amount of time. Additionally, this Novel Algorithm can be further optimized and parallelized to improve its performance.

**Reference**

Traveling Salesman Problem (TSP): https://en.wikipedia.org/wiki/Travelling\_salesman\_problem

Dynamic Programming: https://en.wikipedia.org/wiki/Dynamic\_programming

Branch and Bound Algorithm:

https://en.wikipedia.org/wiki/Branch\_and\_bound

Genetic Algorithm:: https://en.wikipedia.org/wiki/Genetic\_algorithm