

# **Investigation on best algorithm use Traveling Salesman problem**

How can genetic algorithms and 2-opt algorithms efficiently solve the optimized routes when tourists are traveling in Tokyo?

**Extended Essay:** Computer Science

**Word count:** 3997

# Content

<b>1 Introduction</b>	<b>3</b>
1.1 What is a traveling salesman problem?	3
1.2 What is a Genetic algorithm?	4
1.3 What is a 2-opt algorithm?	5
1.4 Research Question	5
<b>2 Exploration</b>	<b>6</b>
2.1 Scope and Hypothesis	6
2.3 Methodology	7
2.3.1 Experiment I - Genetic algorithm with 5 variables	10
2.3.2 Experiment II - Genetic algorithm with 10 variables	12
2.3.3 Experiment III - 2-opt algorithm with 5 variables	12
2.3.4 Experiment IV - 2-opt algorithm with 10 variables	14
<b>3 Analysis</b>	<b>16</b>
3.1 Experiment I	16
3.2 Experiment II	18
3.3 Experiment III	20
3.4 Experiment IV	22
<b>4 Discussion</b>	<b>24</b>
4.1 Major Findings	24
<b>5 Conclusion</b>	<b>25</b>
5.1 Real-life Approaches	25
5.2 Strengths and Limitations	25
5.3 Future Suggestions	26
<b>Reference</b>	<b>28</b>
<b>Appendix</b>	<b>29</b>

# 1 Introduction

## 1.1 What is a traveling salesman problem?

The traveling salesman problem was first introduced to find the shortest route for visiting all the destinations in the given value without repeatedly passing the same location and returning to the starting destinations while keeping at a lower cost and shortest distance. It is a form of optimization problem where the result should be found logically and optimally. This study is now often used in the traveling market, map applications, etc. For example, when planning a tour trip or developing software for clients for route research.

The traveling salesman problem was first introduced to find the shortest route for visiting all the destinations in the given value without repeatedly passing the same location and returning to the starting destinations while keeping at a lower cost. It is a form of an optimization problem where the result should be found logically and optimally.

Similarly, when planning for a trip or a route, data calculation and transition are relevant while the calculation process is complicated. Therefore, an implementation of the algorithm in the solving process is relevant. This concept is later adapted in computer science to solve problems on data transformation between various nodes and also other problems related to optimal traveling. Through several studies from the past, various solutions have been developed. Such as dynamic programming is often used as a solution, as well as other optimization algorithms. For example, genetic algorithms, 2-opt algorithms, and dynamic programming. While there are various approaches for TSP, most findings can only be proximate calculated but not the real optimized route.

## **1.2 What is a Genetic algorithm?**

The genetic algorithm is a heuristic method that is well-known for solving optimization problems. It applies the concept of biology by representing each location as a gene. The algorithm goes through a process from the initial population to selection, crossover, and mutation, and outputs the result if the optimal solution is found, if not the same process will be rerun until the best solution is found. The solutions are present in arrays called “chromosomes”. In genetic algorithms, one generation has a population which is a set of all the solutions(chromosomes) in the particular generation. While each chromosome is formed by a set of genes(nodes).

In the classical genetic algorithms, there are five different solving processes. They are applied together to solve the optimal result for the problem: Initial population, Fitness function, Selection, Crossover, and Mutation. Fitness score is used in the selection phase to determine which chromosome is the best fit for the next generation. In most cases, the higher the fitness score the higher the chance to be selected. During the process, the crossover is the most relevant one, as it will exchange the gene from two selective chromosomes within the crossover point. When the genes in both individuals exchange to form their new offspring (a new set of chromosomes). In order to keep the diversity in each population, a mutation method is used to flip the gene in a chromosome to prevent premature outcomes. Lastly, it outputs the chromosome (solution) represented as an array, while each one of the numbers is a gene(node).

### **1.3 What is a 2-opt algorithm?**

The 2-opt algorithm is another heuristic method that is commonly used to solve optimization problems, such as TSP by improving the initial value through swapping pairs of edges. While offering an efficient way to solve the approximate result. However, it only limits the solution locally, not globally (Davendra, Donald, et al. 1).

There are 5 major steps during the process:

1. The process begins with an initial value from the given data that connects all the given nodes.
2. The algorithm will swap the chosen edges to improve its initial route. In each iteration, 2 edges in the current route will be selected.
3. The selected edges will first be removed and then reconnected to the route by connecting the endpoints of one edge to the other.
4. The algorithm will calculate the distance of this new route and compare the total distance found with the previous route.
5. The algorithm will output the new route if the distance found is less than the previous route, if not, the algorithm will again run from the edge swapping to distance comparison.

### **1.4 Research Question**

The above is a brief background of the topic and concept of the algorithms that will be explored later on. The topic is focused on Tokyo because Tokyo is a popular tourist place where public transportation is one of the most common ways to travel around and commute.

Additionally, it is one of the most developed train systems in the world. Due to the many connections between different places in Tokyo, the TSP can be applied to route optimization for the tourist to find the best route by finding the optimal distance and time between different locations. However, planning and designing a traveling route is complicated and time-consuming. Therefore, I will investigate “To what extent can genetic algorithms and 2-opt algorithms efficiently solve the optimized routes when tourists are traveling in Tokyo?” My aim is to explore the possibilities of shortcut solutions in solving TSP, particularly in finding transportation routes.

## **2 Exploration**

### **2.1 Scope and Hypothesis**

To find the result for the question, I decided to utilize 4 experiments on solving the TSP problem in combination with mathematical and computational knowledge focusing on the distance of the route that is found by the two algorithms. I will compare:

1. The efficiency of genetic algorithms in TSP with 5 variables
2. The efficiency of genetic algorithms in TSP with 10 variables.
3. The efficiency of the 2-opt algorithm in TSP with 5 variables.
4. The efficiency of the 2-opt algorithm in TSP with 10 variables.

Consider the usage of two algorithms specifically in the Tokyo area. The 2-opt could be a more straightforward way to find the best routes, as it is in a local area. On the other hand, the genetic algorithm may seem to be a more complex way to find the solution because there are more steps included to find the final solution. However, TSP is an intractable problem, meaning that no efficient algorithm exists to solve the problem (University of Canterbury Computer Science Education Research Group). Therefore, my hypothesis is that the two algorithms are most likely to have similar results in terms of the efficiency of solving the optimal route.

### **2.3 Methodology**

To make an effective experiment, I will explore whether a shortcut solution exists in planning transportation systems, by comparing the distance of the route found and the solving process of the two algorithms. The experiments will be completed through the fundamental concept of both algorithms through Python. The program of the genetic algorithm will generate the route through selecting, mutation, and comparison of the fitness value. Each generation will be improved as it goes, as the fittest chromosome will be selected in each generation to mutate the next generation. It will output the fittest chromosome and the total distance can be calculated with the data in Table 1. Additionally, the program of the 2-opt algorithm will generate the route by removing 2 coordinates and reconnecting back to the list to reform a new order of the route. It will also calculate the distance of the initial route and the optimal route.

I gathered data about the traveling distance and time between 10 stations in Tokyo. The location: Shinjuku - Asakusa - Sky Tree - Tokyo Tower - Shibuya - Harajuku - Ginza - Jiyugaoka - Mita - Shimokitazawa, as the set to determine the result. I will begin by comparing the solving

process with 2 data sets. Firstly, a comparison between a 5-variable data set and a 10-variable set using a genetic algorithm. Secondly, a comparison between a 5-variable data set and a 10-variable set using a 2-opt algorithm. With the given results, I will evaluate the efficiency of each algorithm through the visual presentation of the routes.

The distance and time data are shown in the table below.

A = Shinjuku

B = Asakusa

C = Oshiage(Sky Tree)

D = Akabanebashi (Tokyo Tower)

E = Shibuya

F = Harajuku

G = Ginza

H = Jiyugaoka

I = Mitaka

J = Shimokitazawa

*Table 1: The distance between 10 destinations*

distance/km	A	B	C	D	E	F	G	H	I	J
A	0	10.4	11.9	7.1	3.4	2.2	6.8	10.4	13.8	4.9
B	10.4	0	1.5	8.6	14.9	12.1	7.1	18.6	27.2	17.3
C	11.9	1.5	0	10.1	16.9	13.6	7.1	20.1	26.6	19.7
D	7.1	8.6	10.1	0	6.1	7.4	3.1	10.9	20.3	8.9
E	3.4	14.9	16.9	6.1	0	1.2	7.2	7	14.3	3
F	2.2	12.1	13.6	7.4	1.2	0	6.5	8	16	3.6
G	6.8	7.1	7.1	3.1	7.2	6.5	0	13.4	21.4	9.5
H	10.4	18.6	20.1	10.9	7	8	13.4	0	21.3	10
I	13.8	27.2	26.6	20.3	14.3	16	21.4	21.3	0	11.3
J	4.9	17.3	19.7	8.9	3	3.6	9.5	10	11.3	0

(Yahoo 乗換案内、時刻表、運行情報)

The data above includes 10 stations that are often visited by tourists as well as local Japanese. Even though 10 stations might seem small compared to the number of stations Tokyo has. Using 10 variables can give out an approximation to determine the efficiency of the two algorithms.

### **2.3.1 Experiment I - Genetic algorithm with 5 variables**

This experiment aims to compare the efficiency of using TSP through two different optimization algorithms. Distance will be the key element to consider. Through genetic algorithms, the route result can be found by comparing each chromosome's fitness score and forming a new generation by selecting a better gene, the same process will go on till the required iterations have been finished. During the experiment process, we will consider factors including, what are the necessary processes in the algorithm to solve TSP, what are the possible shortcut ways, and the given result of an optimized solution.

The procedure of the experiment is:

1. Import random implementation for the program to have functions that randomly select destinations.
2. Identify the population size, the number of locations to travel, and the genes in this population.
3. Create a class of the structure of chromosomes which is determined by fitness value.
4. Create a function to randomly generate numbers from start to end within the given population.
5. Create a function to verify if the gene has already been used in the string, if found it returns false.
6. Create a mutation function that randomly interchanges two characters within the genome and makes sure both characters are different, returning a mutated chromosome.
7. Create a function that generates the initial population of the given information, using the looping until the required length of the chromosome is made.

8. Create a fitness function to determine the fitness value for the chromosome through a distance matrix of all the destinations. If there is a valid connection between the destination, the fitness value is returned, else, the initial maximum is returned.

The procedure above illustrates the classical model of how a genetic algorithm is being used to find the solution. Next, it is how the genetic algorithm solves TSP.

9. Create a TSP problem function with the process of solving the required populations.
10. Implement the algorithm into the function, beginning with determining the population, first creating the chromosomes, and calculating its fitness.
11. Create a while loop for the generating new population to iterate within the required time.  
The new generation is found by mutation function.
12. The fitness function is used to determine the fitness value of each chromosome in the generation.
13. To output the new generation, the fitness value needs to be checked by  
`new_chromosomes <= previous`. If smaller, output a new generation, else repeat the loop.

The experiential hypothesis is that as the destination adds on the complexity of the solving process will increase because there are more variables in a chromosome, meaning that more possible outcomes exist to compare for finding the best route.

### **2.3.2 Experiment II - Genetic algorithm with 10 variables**

To make a deeper comparison, I want to explore further the difference with more variables in the problem. The procedure is the same as experiment I while increasing the number

of genes. Doing so, will show the complexity between the two experiments and further analyze the efficiency and shortcut solution of solving TSP.

Mathematically, the 10 variables problem will take more time to generate the solution, due to an increase in variables, if skipping any part of the step is likely to reduce the accuracy of the problem, as it can lead to a lack of possible solutions. Therefore, the outcome may not be the most optimal solution for the problem.

In experiment, I, only 5 generations are being generated because of the smaller numbers of variables. However, the complexity of the solving process increases as the number of variables increases. Therefore, for the 10 variables experiment when there are only 5 generations, it might not be possible for the program to find the best solution within a limited time of trial as with 10 variables more possible chromosomes exist.

### **2.3.3 Experiment III - 2-opt algorithm with 5 variables**

This experiment aims to explore if there is an efficient yet shortcut way to solve TSP problems by comparing different algorithms that are often being used to solve optimization problems. In order to solve TSP through a 2-opt algorithm, finding a coordinate for each destination is essential as this is how the distance is approximately calculated. The 2-opt algorithm solves problems through changes in cutsets, therefore, it is easier to input the coordinates and generate the solution. However, the data I collected does not have information about the coordinates. To solve for the coordinates, I plot out all the destinations on the map in Tokyo. Then scale the map with x and axis, so the coordinates can be made. However, these

coordinates do not determine the real distances between each location. The distances found are in approximation because the real-world conditions the roads are not straight. These values are only being used for experimental purposes.

To find the coordinates, I applied the distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1. I put the map and the given points into the xy-plan.
2. Find the coordinates for each point
3. Put 2 coordinates into the distance formula and compare the result with the real distance found in Table 1.
4. A valuable x is found as the difference, which is used as a scale to magnify the other coordinates.
5. Multiply all the coordinates with x to find a more accurate scaling.

Coordinate:

A (16, 11.4)

B (26, 14.2)

C(27.7, 14.1)

D(20.5, 6.9)

E(16, 7.4)

These coordinates will be input into the 2-opt algorithm. The procedure for this experiment is:

1. Import random implementation for the program to randomly select destinations.

2. Import math implementation for the program to calculate the total distance.
3. Create a function that generates the distance formula to calculate the distance between 2 destinations.
4. Create a function that calculates the total distances of the route.
5. Create a 2-opt algorithm function that improves a given route by moving the cutset.
6. Create the iteration of the solution 5 times to check the possible result the algorithm gives and the cities will be represented as coordinates.
7. Output all the results, the initial route, and the optimized route with both of the distances.

### **2.3.4 Experiment IV - 2-opt algorithm with 10 variables**

Mathematically the complexity increases exponentially as the number of variables increases, so I will also explore the 2-opt with an increase in variables to see the change in solving efficiency. The coordinates used for this experiment are listed below calculated through the same method in experiment III.

Coordinate:

A (16, 11.4)

B (26, 14.2)

C(27.7, 14.1)

D(20.5, 6.9)

E(16, 7.4)

F(16.1, 9.1)

G(22.5, 9.1)

H(12.5, 0.8)

I(1.3, 13.1)

J(12.4, 7.8)

The 10 variables experiment for 2-opt and genetic algorithms will show a bigger difference than the one with 5 variables because the bigger number of variables can illustrate the complexity of the solving process clearly. Where the steps taken and the number of outcomes can be compared.

### 3 Analysis

#### 3.1 Experiment I

**Result:**

<b>Initial population:</b> <b>CHROMO FITNESS VALUE</b>	<b>Generation 1</b> <b>CHROMO FITNESS VALUE</b>	<b>Generation 2</b> <b>GNAME FITNESS VALUE</b>
013420 200000	043210 31.5	043210 31.5
043210 31.5	043210 31.5	043210 31.5
032410 200000	014230 200000	031240 37.49999999999999
014230 200000	042310 49.4	043210 31.5
043210 31.5	034210 42.0	014230 200000

<b>Generation 3</b> <b>GNAME FITNESS VALUE</b>	<b>Generation 4</b> <b>GNAME FITNESS VALUE</b>	<b>Generation 5</b> <b>GNAME FITNESS VALUE</b>
043210 31.5	043210 31.5	043210 31.5
043210 31.5	043210 31.5	043210 31.5
043210 31.5	043210 31.5	043210 31.5
031240 37.49999999999999	031240 37.49999999999999	031240 37.49999999999999
034210 42.0	034210 42.0	031240 37.49999999999999

Figure 1. The result of the genetic algorithm for solving 5 variables

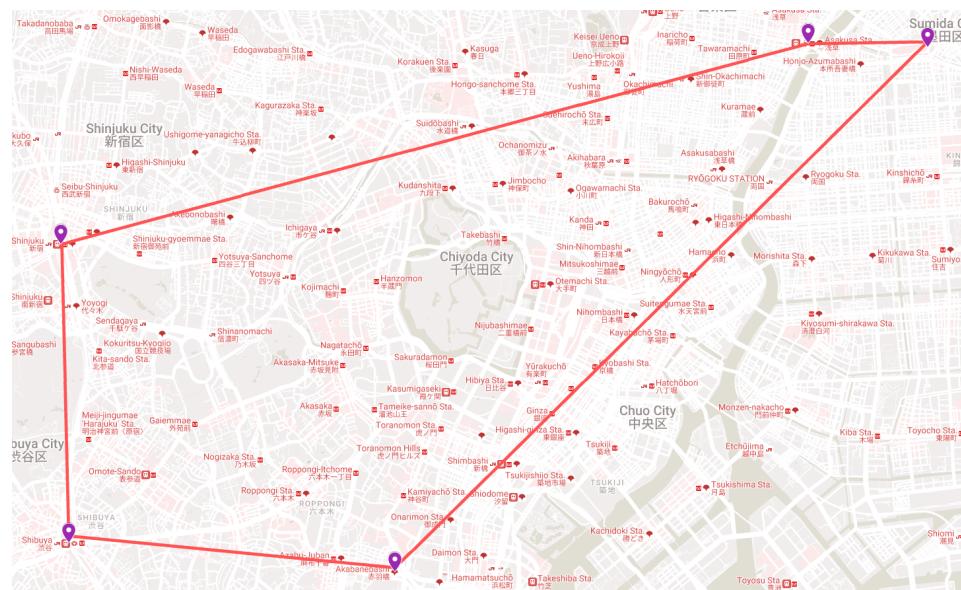


Figure 2. The optimized route of the genetic algorithm with 5 variables

The 2 figures above show the result of the optimized route solved by the genetic algorithm. Figure 1. shows the results that are generated by a genetic algorithm. The generation is getting better and better, as the fitness value gets smaller as the generation passes. Figure 2 shows the optimized route that is being generated by selecting the fittest chromosomes from each generation and forming a better version of the next generation.

Through that, I can understand the importance of going through all the solving steps because if one step is missing, may lead to a reduction in the effectiveness of the program. For example, if either fitness value, crossover, selection, or mutation are not processed for all generations. It could be less accurate and unreliable because when the program doesn't fully process the step. Therefore, increases the chance of the result being incorrect. For example, if a crossover is not done properly, it could lead to the initial generation being the same as the new generation, as the genes are not swapped correctly. Hence, when determining the optimal route through the genetic algorithm, there is no shortcut or more efficient way to determine the result.

### 3.2 Experiment II

<b>Initial population:</b>	<b>Generation 1</b>	<b>Generation 2</b>
<b>CHROMO FITNESS VALUE</b>	<b>CHROMO FITNESS VALUE</b>	<b>CHROMO FITNESS VALUE</b>
06742958130 126.29999999999998	07659123480 97.0	01345798620 96.0
06279183450 118.3	08236197450 98.40000000000002	07659123480 97.0
07146329850 200000	01679852340 101.3999999999999	08236197450 98.40000000000002
02683594710 200000	03521976480 105.60000000000001	01376248950 98.6999999999999
07428619350 125.2	01376248950 98.6999999999999	01679852340 101.3999999999999
03921746850 109.60000000000001	03921746850 109.60000000000001	08325479610 103.0
06432918570 128.8	07842391650 115.0	03521796480 103.0
03274681950 123.2	06279183450 118.3	03921746850 109.60000000000001
07842391650 115.0	06957132480 110.1999999999999	06957132480 110.1999999999999
06345798120 200000	03274681950 123.2	07328619450 110.2
08236197450 98.40000000000002	07328619450 110.2	06259834170 112.70000000000002
01679852340 101.3999999999999	03742958160 122.29999999999998	07842391650 115.0
07912638540 200000	06432918570 128.8	06278193450 118.2
05839264710 200000	06824791530 132.6	05832964710 121.0
01576248930 109.5	07146329580 200000	03742958160 122.29999999999998
07824691530 135.79999999999998	02687594310 200000	03274681950 123.2
01468953720 200000	01345798620 96.0	06437918520 127.0
06957832410 121.80000000000001	07412638590 200000	06854791230 98.39999999999999
01369427580 200000	05832964710 121.0	07142369580 200000
07659123480 97.0	01428953760 200000	02687495310 200000
06253894170 200000	01369827540 200000	07412698530 200000
02759143680 200000	06259834170 112.70000000000002	01425983760 200000
04125769380 200000	02751943680 200000	01396827540 200000
08321479650 200000	04521769380 200000	01752943680 200000
03521978460 106.5	08325479610 103.0	04521789360 89.69999999999999
<b>Generation 3</b>	<b>Generation 4</b>	<b>Generation 5</b>
<b>CHROMO FITNESS VALUE</b>	<b>CHROMO FITNESS VALUE</b>	<b>CHROMO FITNESS VALUE</b>
04521789360 89.69999999999999	07459123680 89.39999999999999	07459123680 89.39999999999999
01345798620 96.0	04521789360 89.69999999999999	04521789360 89.69999999999999
07459123680 89.39999999999999	01345798620 96.0	01345798620 96.0
06857491230 98.19999999999999	06857491230 98.19999999999999	01376245980 97.19999999999999
08236197450 98.40000000000002	08236197450 98.40000000000002	06857491230 98.19999999999999
01376248950 98.69999999999999	01376245980 97.19999999999999	08236197450 98.40000000000002
01679852340 101.3999999999999	07325489610 98.8	07325489610 98.8
07325489610 98.8	01679852340 101.3999999999999	01679852340 101.3999999999999
03521796480 103.0	03521796480 103.0	03674982150 87.3
03921746850 109.60000000000001	03921745860 108.2	06521793480 100.1
06957132480 110.19999999999999	06957132480 110.19999999999999	01523984760 107.9
07328619450 110.2	07328619450 110.2	03921745860 108.2
06259834170 112.70000000000002	03742598160 111.5	08957132460 104.9
07842391650 115.0	06259834170 112.70000000000002	07328619450 110.2
06278193450 118.2	07842391650 115.0	03742598160 111.5
05832964710 121.0	07415698230 115.5	06279834150 110.89999999999999
03742598160 111.5	03274986150 101.3999999999999	05834162790 108.69999999999999
03274689150 115.8	06278193450 118.2	07842391650 115.0
06437918520 127.0	05834962710 113.30000000000001	07495618230 108.6
07142365980 200000	06437819520 125.89999999999999	06278193450 118.2
02387495610 200000	07142563980 200000	01936728540 120.40000000000002
07415698230 115.5	02987435610 200000	06437819520 125.89999999999999
01423985760 200000	01523984760 107.9	07142583960 200000
01936827540 200000	01936728540 120.40000000000002	02187435690 200000
01753942680 200000	01743952680 200000	01743259680 114.09999999999999

Figure 3. The result of the genetic algorithm for solving 10 variables



Figure 4. The optimized route of the genetic algorithm with 10 variables

As the result shown in Figure 3, the best route is being found through the same process, however, with more variables in the problem, more possible chromosomes exist to determine the optimal route. This experiment was done with 10 variables, while the number of chromosomes is the exponent of experiment I. The optimal route is found by looking at Figure 3 where the smallest fitness value of 89.4 repeatedly shows a newer generation and is not replaced by a fitter chromosome, suggesting this chromosome is the optimal solution.

However, for the experiment to be more accurate, it could be better if more generations were made, as there are 10 variables. Mathematically  $1024$  possible routes can be determined. This is because to find the combination of the routes that exist the formula  $2^n$  is used, where n

represents the number of the variable. While the selection and mutation process can eliminate some outcomes in a faster way. However, for it to be more accurate, having 41 generations could be a more accurate test. Through this, it demonstrates that the complexity of the solution increases exponentially as the number of variables increases.

### 3.3 Experiment III

#### Solution 1:

Initial Route:  $[(12.3, 7.5), (0.66, 4.8), (10.7, 7.5), (0.7, 0.9), (5.1, 0.5)]$   
 Optimized Route:  $[(12.3, 7.5), (10.7, 7.5), (0.66, 4.8), (0.7, 0.9), (5.1, 0.5)]$   
 Initial Total Distance: 48.787462036964  
 Optimized Total Distance: 30.35697271658654

#### Solution 2:

Initial Route:  $[(0.7, 0.9), (5.1, 0.5), (0.66, 4.8), (12.3, 7.5), (10.7, 7.5)]$   
 Optimized Route:  $[(0.7, 0.9), (0.66, 4.8), (5.1, 0.5), (12.3, 7.5), (10.7, 7.5)]$   
 Initial Total Distance: 36.12974493250777  
 Optimized Total Distance: 33.70467601436208

#### Solution 3:

Initial Route:  $[(12.3, 7.5), (0.66, 4.8), (0.7, 0.9), (10.7, 7.5), (5.1, 0.5)]$   
 Optimized Route:  $[(12.3, 7.5), (10.7, 7.5), (0.66, 4.8), (0.7, 0.9), (5.1, 0.5)]$   
 Initial Total Distance: 46.837185666911715  
 Optimized Total Distance: 30.35697271658654

#### Solution 4:

Initial Route:  $[(10.7, 7.5), (0.66, 4.8), (12.3, 7.5), (5.1, 0.5), (0.7, 0.9)]$   
 Optimized Route:  $[(10.7, 7.5), (12.3, 7.5), (5.1, 0.5), (0.66, 4.8), (0.7, 0.9)]$   
 Initial Total Distance: 48.787462036964  
 Optimized Total Distance: 33.70467601436208

#### Solution 5:

Initial Route:  $[(0.66, 4.8), (0.7, 0.9), (12.3, 7.5), (10.7, 7.5), (5.1, 0.5)]$   
 Optimized Route:  $[(0.66, 4.8), (0.7, 0.9), (10.7, 7.5), (12.3, 7.5), (5.1, 0.5)]$   
 Initial Total Distance: 33.99164563422368  
 Optimized Total Distance: 33.70467601436208

Figure 5. The result of the 2-opt algorithm for solving 5 variables



Figure 6. The optimized route of the 2-opt algorithm with 5 variables

In Figure 5, the result of the 2-opt algorithm is shown differently from the genetic algorithm. Based on the result, shows that the initial route is found by randomly selecting and the optimized route is discovered by swapping the two points in the initial route. Therefore, when a 2-opt algorithm is applied to solve TSP problems or other optimization, generating multiple solutions is relevant because there are many possible outcomes, the given one solution is not necessarily the best route. The algorithm only forms an improved version of the initial chosen route. While both algorithms are for solving optimization, the 2-opt algorithm could be slightly less efficient than the genetic algorithm, because the 2-opt solves the route one by one by beginning with only one initial route. However, the genetic algorithm begins with a generation where many possible routes are found together so that more inefficient routes can be eliminated at once. It could eliminate the less effective ones by a generation. Therefore, the 2-opt might need more time to find the best route because it solves a route one by one in each solution.

## 3.4 Experiment IV

### Solution 1:

Initial Route: [(16, 7.4), (22.5, 9.1), (27.7, 14.1), (16, 11.4), (12.4, 7.8), (20.5, 6.9), (1.3, 13.1), (16.1, 9.1), (12.5, 0.8), (26, 14.2)]

Optimized Route: [(16, 7.4), (16.1, 9.1), (16, 11.4), (12.4, 7.8), (1.3, 13.1), (12.5, 0.8), (20.5, 6.9), (22.5, 9.1), (27.7, 14.1), (26, 14.2)]

Initial Total Distance: 114.84963338117643

Optimized Total Distance: 72.07520963828395

### Solution 2:

Initial Route: [(16.1, 9.1), (27.7, 14.1), (1.3, 13.1), (20.5, 6.9), (16, 7.4), (12.5, 0.8), (26, 14.2), (12.4, 7.8), (22.5, 9.1), (16, 11.4)]

Optimized Route: [(16.1, 9.1), (12.4, 7.8), (1.3, 13.1), (12.5, 0.8), (16, 7.4), (20.5, 6.9), (22.5, 9.1), (27.7, 14.1), (26, 14.2), (16, 11.4)]

Initial Total Distance: 124.6575215029581

Optimized Total Distance: 69.43245087060075

### Solution 3:

Initial Route: [(16, 11.4), (16.1, 9.1), (20.5, 6.9), (12.4, 7.8), (16, 7.4), (26, 14.2), (12.5, 0.8), (22.5, 9.1), (1.3, 13.1), (27.7, 14.1)]

Optimized Route: [(16, 11.4), (12.4, 7.8), (1.3, 13.1), (12.5, 0.8), (16, 7.4), (16.1, 9.1), (20.5, 6.9), (22.5, 9.1), (26, 14.2), (27.7, 14.1)]

Initial Total Distance: 123.10405743025333

Optimized Total Distance: 70.98879310434182

### Solution 4:

Initial Route: [(1.3, 13.1), (20.5, 6.9), (27.7, 14.1), (12.5, 0.8), (22.5, 9.1), (16.1, 9.1), (16, 7.4), (26, 14.2), (12.4, 7.8), (16, 11.4)]

Optimized Route: [(1.3, 13.1), (12.4, 7.8), (12.5, 0.8), (16, 7.4), (20.5, 6.9), (22.5, 9.1), (27.7, 14.1), (26, 14.2), (16.1, 9.1), (16, 11.4)]

Initial Total Distance: 118.66729571017429

Optimized Total Distance: 71.42602273549281

### Solution 5:

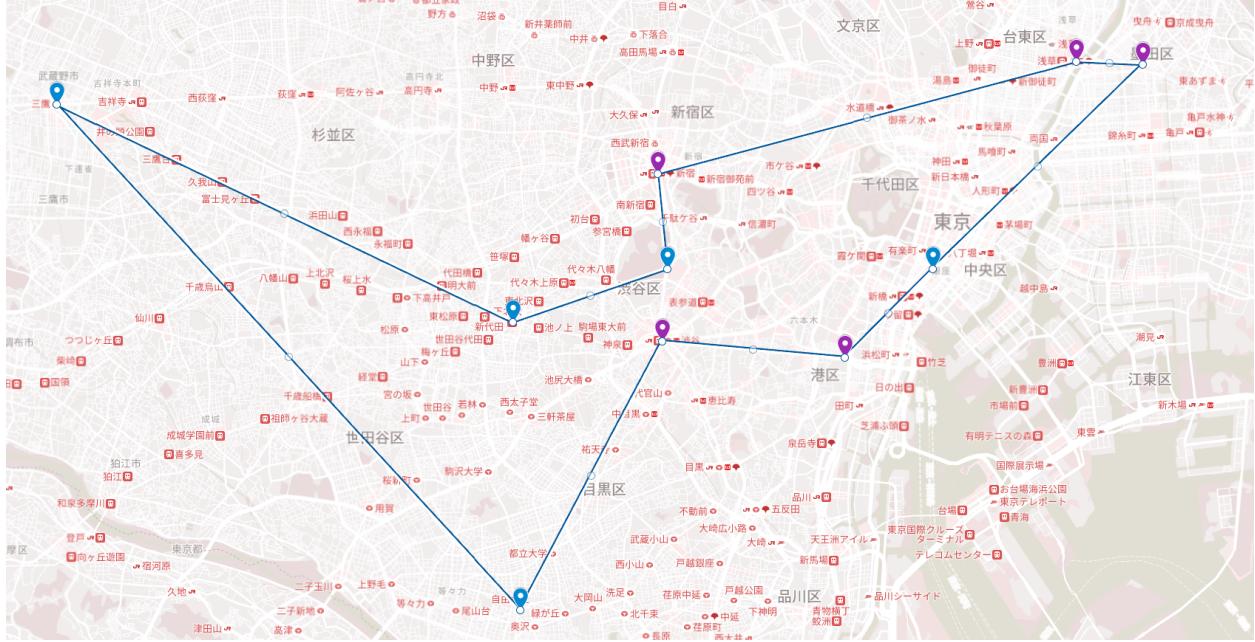
Initial Route: [(20.5, 6.9), (1.3, 13.1), (16.1, 9.1), (16, 11.4), (12.4, 7.8), (26, 14.2), (12.5, 0.8), (27.7, 14.1), (16, 7.4), (22.5, 9.1)]

Optimized Route: [(20.5, 6.9), (16.1, 9.1), (16, 7.4), (12.5, 0.8), (1.3, 13.1), (12.4, 7.8), (16, 11.4), (26, 14.2), (27.7, 14.1), (22.5, 9.1)]

Initial Total Distance: 120.32422184954275

Optimized Total Distance: 70.39430811629757

Figure 7. The result of the 2-opt algorithm for solving 10 variables



*Figure 8. The optimized route of the 2-opt algorithm with 10 variables*

Figure 8 illustrates the result of the application of a 2-opt algorithm to solve TSP with 10 destinations. Even though the algorithm randomly selects a route and then improves it, there is a possibility for the optimal route in the initial solution. However, for accurate results, evaluating all the possible outcomes is needed, making sure all the outcomes are compared. Which increases the complexity of the solving process. This is particularly relevant when the problem is associated with real-life approaches. Therefore, a missing solution could lead to inaccurate results, suggesting the absence of an efficient solution for the TSP.

## 4 Discussion

### 4.1 Major Findings

By exploring the two algorithms in solving traveling salesman problems, I concluded that to accurately solve for the optimal route, all the steps in the solving process cannot be missed. In terms of efficiency or shortcut solution, there was no solution, as solving complexity increases factorially (Rahul). This is evident in the exploration of the algorithms with different numbers of variables, the complexity of the process increases as more variables are added. By using both algorithms to do experiments, a big difference in their solving efficiency can not be found. This is concluded based on exploring 5 and 10 variables but there could potentially be deviation with a larger variable set.

Based on the major findings, there needs to be an efficient algorithm to solve traveling salesman problems. With the factors, I discovered during the experiment.

1. There is no polynomial-time solution (Rubin, S. H.).
2. It is intractable, there is no efficient/shortcut algorithm to solve the problem
3. The solution time grows exponentially as the number of variables increases

These conditions are seen in nondeterministic polynomial (NP) problems. My experiment in solving the optimal route evident that TSP fulfills all of the conditions listed. Resulting in the discovery of TSP as an NP problem. However, there are many other conditions and factors in an NP problem. The experiments I have done can only give the idea that TSP is likely to be an NP problem but more experiment and expiration is required when proofing.

The most efficient result can only be achieved approximately because it requires a long computational process and large resources. Therefore, it is currently difficult to conclude the efficiency of different algorithms in solving TSP. The study by Vladimir Deineko and Alexander Tiskin stated that TSP is a classical NP-hard problem where it is a type of problem that is unsolvable due to its complexity (Deineko and Tiskin).

## 5 Conclusion

### 5.1 Real-life Approaches

TSP is one of the most relevant problems for planning and designing transportation. For example, it can be implemented into the traveling business when planning for the best-traveling route for their customers. In addition, each local metropolitan must determine its local transportation development plan. Crucially, nowadays with the growing awareness of global warming, it focuses on causing minimal damage and harm to the environment. Such an algorithm would be able to be implemented in airplane flights, and car routes, reducing carbon emissions to our atmosphere.

### 5.2 Strengths and Limitations

The experiment and research show the usage of optimization algorithms under the Tokyo train system to determine the optimal route. The strength of this experiment is learning the application of computer knowledge in real-life situations. Also possible to discover the possible solution that can be later used in solving optimization problems.

However, several limitations need to be considered for further development by more complex involvement. Due to the complexity of the algorithm and polynomial-time factor, there are many ways to approach the problems more accurately. Such as experimenting with experts working in transportation systems.

First, the complexity of the actual transportation in Tokyo, how the railway was developed, and the various lines people can use. Transfer time and waiting times could be some of the possible elements that need to be considered. Even though the optimized route is found from the two algorithms, when the path is implemented in real life it might not be the best route, due to the transfer, express train, and cost that needs to be considered.

Second, in solving the 2-opt algorithm, the coordinates are needed, however, I could only find the coordinates by looking at the graph and the distance is being calculated through the distance formula. Meaning that it is a straight line distance but most railways are not built in a straight line.

### **5.3 Future Suggestions**

Based on the research, when applying TSP in a train system, it takes a long time for the programmer and designer to get the result, as a shortcut way or more efficient solutions in TSP or other NP-hard problems have not yet been found. It is currently an area mathematicians and computer scientists have been exploring experimenting and investigating.

The introduction of quantum computing could be the possibility of solving TSP/NP-hard problems in the future. Quantum computing is a new introduction to solving complex problems more efficiently as it processes the data differently compared to the traditional approach. It has been applied to solve some NP problems in several studies that can be solved in a polynomial time. However, the solution to the problems that cannot be solved in polynomial time cannot have been discovered. In the future, when more experiments and studies on quantum computing have been done, it could be the possible solution for solving NP-hard problems. A study from Cornell University experienced solving the possible largest size of partition problem, reformulated by several NP-hard problems and optimization problems. It is solved by using a quantum computer, resulting in “the first time a size of 256 Maximum Cut, 64 Minimum Partition, and 128 Maximum Weighted Independent Set have been executed on real universal gate-based quantum computers.” Suggests that the introduction of quantum computing could be a possible area to further investigate in solving NP-hard problems in the future (Chatterjee et al. 10 ).

## Reference

- "2-opt Algorithm: Solving the Traveling Salesman Problem in Python." *Saturn Cloud Blog*, 25 August 2023,  
<https://saturncloud.io/blog/2opt-algorithm-solving-the-travelling-salesman-problem-in-python/>.
- "An Introduction to Genetic Algorithms - Whitman College." *Whitman College*, n.d.,  
<https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf>.
- Chatterjee, Yagnik, et al. "Solving Various NP-Hard Problems Using Exponentially Fewer Qubits on a Quantum Computer." *ArXiv*, 2023, /abs/2301.06978. Accessed 30 Sept. 2023.
- Chaudhary, S. "Genetic Algorithm Applications in Machine Learning." *Genetic Algorithm Applications in Machine Learning*, 18 November 2022,  
<https://www.turing.com/kb/genetic-algorithm-applications-in-ml>.
- Contributor, T. "What Is the Traveling Salesman Problem (TSP)?" *WhatIs.com*, 26 June 2020,  
<https://www.techtarget.com/whatis/definition/traveling-salesman-problem>.
- Davendra, Donald, et al. 'CUDA Accelerated 2-OPT Local Search for the Traveling Salesman Problem'. *Novel Trends in the Traveling Salesman Problem*, IntechOpen, 9 Dec. 2020. Crossref, doi:10.5772/intechopen.93125.
- Deineko, Vladimir; and Alexander Tiskin. "(PDF) One-Sided Monge TSP Is NP-Hard - Researchgate." *One-Sided Monge TSP Is NP-Hard*, Nov. 2006,  
[www.researchgate.net/publication/225151677\\_One-sided\\_monge\\_TSP\\_is\\_NP-Hard](http://www.researchgate.net/publication/225151677_One-sided_monge_TSP_is_NP-Hard).
- Davis, A. "Traveling Salesman Problem with the 2-opt Algorithm." *Medium*, 19 May 2022,  
<https://slowandsteadybrain.medium.com/traveling-salesman-problem-ce78187cf1f3>.
- "Difference Between NP-Hard and NP-Complete Problem." *GeeksforGeeks*, 5 January 2023,  
<https://www.geeksforgeeks.org/difference-between-np-hard-and-np-complete-problem/>.
- "Genetic Algorithms - Crossover." *Online Courses and eBooks Library*, n.d.,  
[https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_crossover.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm).
- "Genetic Algorithms - Mutation." *Online Courses and eBooks Library*, n.d.,  
[https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_mutation.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm).
- Hilton, Rod. "Traveling Salesperson: The Most Misunderstood Problem." n.d.,  
<https://www.rodhilton.com/2012/09/14/traveling-salesman-the-most-misunderstood-problem/>.
- Li, W. "How to Solve the Traveling Salesman Problem." *IntechOpen*, 9 February 2021,  
<https://www.intechopen.com/chapters/75156>.
- Mallawaarachchi, V. "Introduction to Genetic Algorithms - Including Example Code." *Medium*, 1 March 2020,  
<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>.

"NP-Complete Problem." *Encyclopædia Britannica, Inc.*, 28 July 2023, <https://www.britannica.com/science/NP-complete-problem>.

"NP-Hard and NP-Complete Classes." *Online Courses and eBooks Library*, n.d., [https://www.tutorialspoint.com/design\\_and\\_analysis\\_of\\_algorithms/design\\_and\\_analysis\\_of\\_algorithms\\_np\\_hard\\_complete\\_classes.htm](https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_np_hard_complete_classes.htm).

*Qiskit.* "What Can a Quantum Computer Do?" Medium, 9 February 2022, <https://medium.com/qiskit/what-can-a-quantum-computer-actually-do-4daed0691f6b>.

"Pygad Module - PyGAD 3.2.0 Documentation." Pygad Module, n.d., <https://pygad.readthedocs.io/en/latest/pygad.html#cal-pop-fitness>.

*Qiskit.* "What Can a Quantum Computer Do?" Medium, 9 February 2022, <https://medium.com/qiskit/what-can-a-quantum-computer-actually-do-4daed0691f6b>.

*Rahul.* "What Is The Traveling Salesman Problem (TSP)?" Route Optimization Blog, 26 June 2023, <https://blog.route4me.com/traveling-salesman-problem/>.

*Rubin, S. H., Bouabana-Tebibel, T., Hoadjli, Y., & Ghalem, Z.* "Reusing the NP-Hard Traveling-Salesman Problem to Demonstrate That P~NP (Invited Paper)." In 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI), 574-581. Pittsburgh, PA, USA, 2016. doi: 10.1109/IRI.2016.84.

"Solve the Traveling Salesman Problem (Genetic Algorithm, Ant Colony Optimization)." YouTube, 25 October 2021, <https://www.youtube.com/watch?v=Sk9QQUGMdY8>.

"Traveling Salesman Problem Using Genetic Algorithm." GeeksforGeeks, 21 February 2023, <https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/>.

*University of Canterbury Computer Science Education Research Group.* "Complexity and Tractability 12.4. the Travelling Salesman Problem." *The Travelling Salesman Problem - Complexity and Tractability - Computer Science Field Guide*, [www.csfieldguide.org.nz/en/chapters/complexity-and-tractability/the-travelling-salesman-problem/](http://www.csfieldguide.org.nz/en/chapters/complexity-and-tractability/the-travelling-salesman-problem/). Accessed 26 Oct. 2023.

*Venhuis, M.* "Around the World in 90.414 Kilometers." Medium, 22 April 2019, <https://towardsdatascience.com/around-the-world-in-90-414-kilometers-ce84c03b8552>.

"Yahoo 乗換案内、時刻表、運行情報." Yahoo!路線情報, [transit.yahoo.co.jp/](https://transit.yahoo.co.jp/). Accessed 27 Oct. 2023.

## **Appendix I - Genetic Algorithm:**

```
from random import randint
```

```
INT_MAX = 200000
```

```
V = 10
```

```
GENES = "ABCDEFGHIJ"
```

```
START = 0
```

```
POP_SIZE = 25
```

```
class route:
```

```
    def __init__(self) -> None:
```

```
        self.chromo = ""
```

```
        self.fitness = 0
```

```
    def __lt__(self, other):
```

```
        return self.fitness < other.fitness
```

```
    def __gt__(self, other):
```

```
        return self.fitness > other.fitness
```

```
def random(start, end):
```

```
    return randint(start, end-1)
```

```
def repeat(g, c):  
    for i in range(len(g)):  
        if g[i] == c:  
            return True  
    return False
```

```
def mutatedGene(chromo):  
    chromo = list(chromo)  
    while True:  
        r = random(1, V)  
        r1 = random(1, V)  
        if r1 != r:  
            temp = chromo[r]  
            chromo[r] = chromo[r1]  
            chromo[r1] = temp  
        break  
    return ''.join(chromo)
```

```
def create_chromo():  
    chromo = "0"  
    while True:  
        if len(chromo) == V:  
            chromo += chromo[0]
```

```
break

temp = random(1, V)

if not repeat(chromo, chr(temp + 48)):

    chromo += chr(temp + 48)

return chromo
```

```
def cal_fitness(chromo):

    mp = [

    ]

    f = 0

    for a, b in zip(chromo, chromo[1:]):

        cost = mp[ord(a) - 48][ord(b) - 48]

        if cost == INT_MAX:

            return INT_MAX

        f += cost
```

```
return f
```

```
gen = 1

gen_thres = 5

population = []
```

```
def TSPUtil(mp):
```

```

gen = 1

gen_thres =5

population = []

temp = route()

for i in range(POP_SIZE):

    temp = route()

    temp.chromo = create_chromo()

    temp.fitness = cal_fitness(temp.chromo)

    population.append(temp)

print("\nInitial population:\nCHROMO\tFITNESS VALUE")

for individual in population:

    print(individual.chromo, individual.fitness)

    print()

while gen <= gen_thres:

    population.sort(key=lambda x: x.fitness)

    new_population = []

    for i in range(POP_SIZE):

        p1 = population[i]

        new_g = mutatedGene(p1.chromo)

        new_chromo = route()

        new_population.append(new_chromo)

```

```

new_chromo.chromo = new_g

new_chromo.fitness = cal_fitness(new_chromo.chromo)

if new_chromo.fitness <= p1.fitness:
    new_population.append(new_chromo)
else:
    new_population.append(p1)

population = new_population

print("Generation", gen)
print("GNOME\tFITNESS VALUE")

for individual in population:
    print(individual.chromo, individual.fitness)

gen += 1

if __name__ == "__main__":
    mp = [
        ]
    TSPUtil(mp)

```

## Appendix II - 2-opt Algorithm:

```
import random  
import math  
  
def calculate_distance(city1, city2):  
    x_diff = city1[0] - city2[0]  
    y_diff = city1[1] - city2[1]  
    return math.sqrt(x_diff**2 + y_diff**2)  
  
def calculate_total_distance(route):  
    total_distance = 0  
    for i in range(len(route)):  
        total_distance += calculate_distance(route[i], route[(i + 1) % len(route)])  
    return total_distance  
  
def two_opt(route):  
    best_route = route  
    best_distance = calculate_total_distance(route)  
    improved = True  
  
    while improved:  
        improved = False  
        for i in range(1, len(route) - 1):
```

```

for j in range(i + 1, len(route)):

    new_route = best_route[:i] + best_route[i:j][::-1] + best_route[j:]

    new_distance = calculate_total_distance(new_route)

    if new_distance < best_distance:

        best_route = new_route # Update the best route here

        best_distance = new_distance

        improved = True

        break # Break the inner loop if improvement is found

return best_route


cities = []

num_solutions = 5

solutions = []


for _ in range(num_solutions):

    initial_route = random.sample(cities, len(cities))

    optimized_route = two_opt(initial_route)

    solutions.append((initial_route, optimized_route))

for i, (initial_route, optimized_route) in enumerate(solutions):

    print(f"Solution {i+1}:")

    print("Initial Route:", initial_route)

```

```
print("Optimized Route:", optimized_route)
print("Initial Total Distance:", calculate_total_distance(initial_route))
print("Optimized Total Distance:", calculate_total_distance(optimized_route))
print()
```