

# Open Vehicle Routing Problem by Genetic Algorithm



**GROUP 5:** Omkar R-191070059    Rahul B-191070018    Akash N-191070050

# Open Vehicle Routing Problem

The Open vehicle routing problem (OVRP) has received in the literature relatively less attention than the VRP. The problem is first described in a paper by Schrage (1981) without any solution attempt. Bodin et al. (1983) address the express airmail distribution in the USA and solve two separate routing problems, one for the delivery and another one for the pick up using a modified Clarke-Wright saving algorithm. The open vehicle routing problem (OVRP) describes efficient routes with minimum total distance and cost for a fleet of vehicles that serve some commodity to a given number of customers. The OVRP differs from the well-known vehicle routing problem (VRP) in that the vehicles do not necessarily return to their original locations after serving to the customers; if they do, they must follow the same path in the reverse order.

The major difference in theory between the OVRP and the VRP is that the routes in the OVRP consist of Hamiltonian paths originating at the depot and ending at one of the customer side, while the routes in the VRP are Hamiltonian cycles. In other words, the best Hamiltonian path is NP-hard, since the Hamiltonian path problem is equivalent to the traveling salesman problem, which is known to be NP-hard.

# Introduction

Amazon.com, Inc. is an American multinational technology company that focuses on e-commerce, cloud computing, digital streaming, and artificial intelligence. The year was 94' and Bezos was working diligently on Wall Street. At 30 years old, he began to see the internet revolution take place, and made the decision to quit his job and start an internet company.

“The wake-up call was finding this startling statistic that web usage in the spring of 1994 was growing at 2,300 percent a year. You know, things just don't grow that fast. It's highly unusual, and that started me thinking, “What kind of business plan might make sense in the context of that growth?”

After making a list of the 'top 20' products that he could potentially sell on the internet, he decided on books because of their low cost and universal demand. It turns out, it was just the beginning.....


# The Logistics

Apart from putting every product on a single website, the main business of amazon is its logistics. Amazon aims for the day when users will get their placed orders within minutes. Currently, they offer 1-day delivery to few cities. Amazon is one of the new companies that is competing against decades-old logistic companies like DHL, FedEx.

A lot of things happen between placing an order and its delivery. If any step goes wrong, there will be a delay in delivery. Amazon stores nearly all of its products in Amazon Fulfillment Centers. They are large warehouses present in different states.

## Amazon Logistic Network





Whenever a user places an order, it's get packed and get ready to be shipped. After this, the order travels to its next stop "Amazon Sortation Centers".

Depending on the distance, either air cargo planes or lorry is used. After which the package gets transported to a delivery station near the customer location. Now, for last-mile connectivity small delivery vans are used which travel to different locations within a city to deliver packages.

So if delivery routes for all these operations are not planned, it will cost millions to Amazon.



# Search Space

Let's say you are currently in Hyderabad, and you have to go to Delhi, Lucknow, and Kolkata. What route you would take?

How many possible routes are there? There are 6 possible routes that you can take. This is the search space of this problem.

The space of all feasible solutions is called search space. Within search space, some solutions will be optimal and some will be non optimal.

For this problem, the search space is very small, thus we could find the solution very easily. But will you be able to find the best solution if there are a total of 30 destinations? The search space for this problem will have  $30!$  solutions, or 265,252,859,812,191,058,636,308,480,000,000 solutions.

It's not possible to compare each of those solutions for deliveries of just 30 packages. So we need an algorithm that will search this search space for a suitable or best solution.

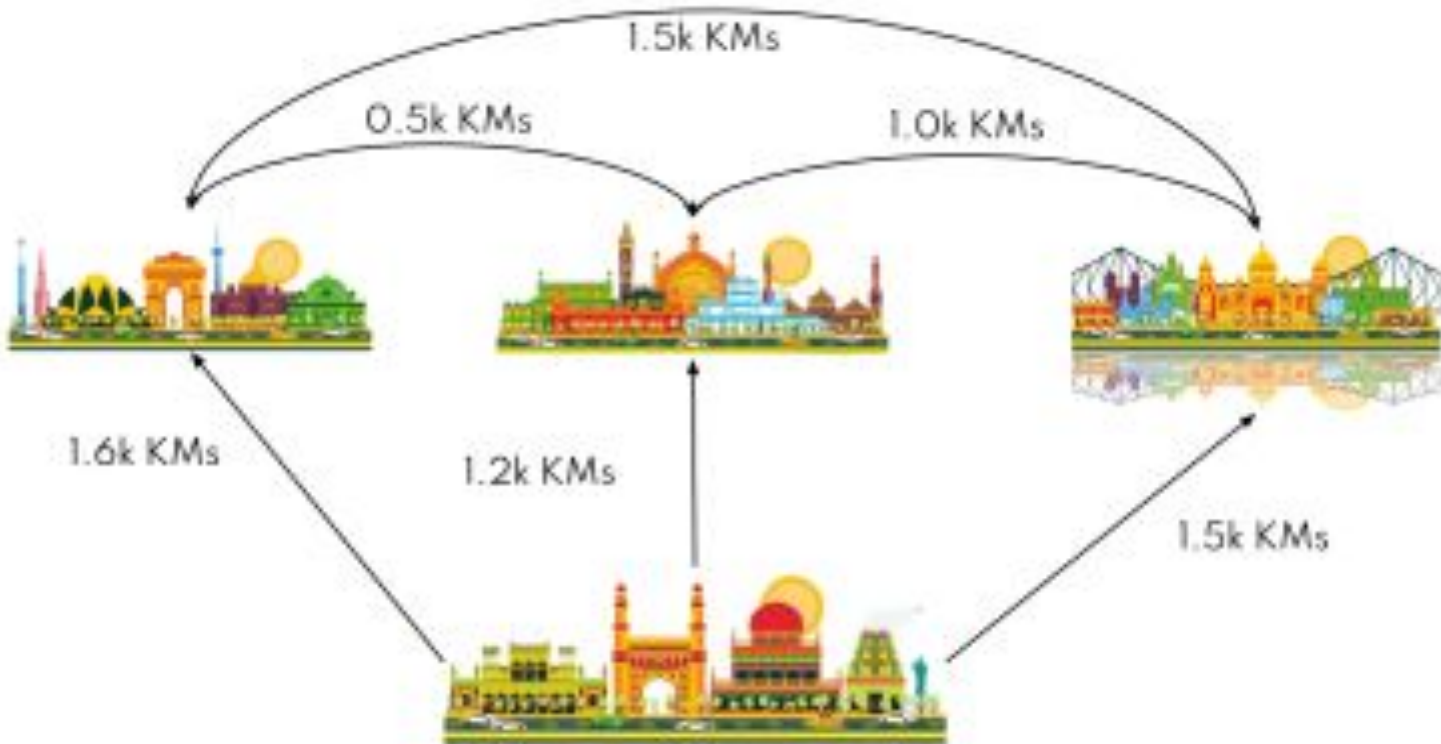


Figure : Search space



# Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. Heuristic means that the algorithm applies shortcuts, and judgments efficiently. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

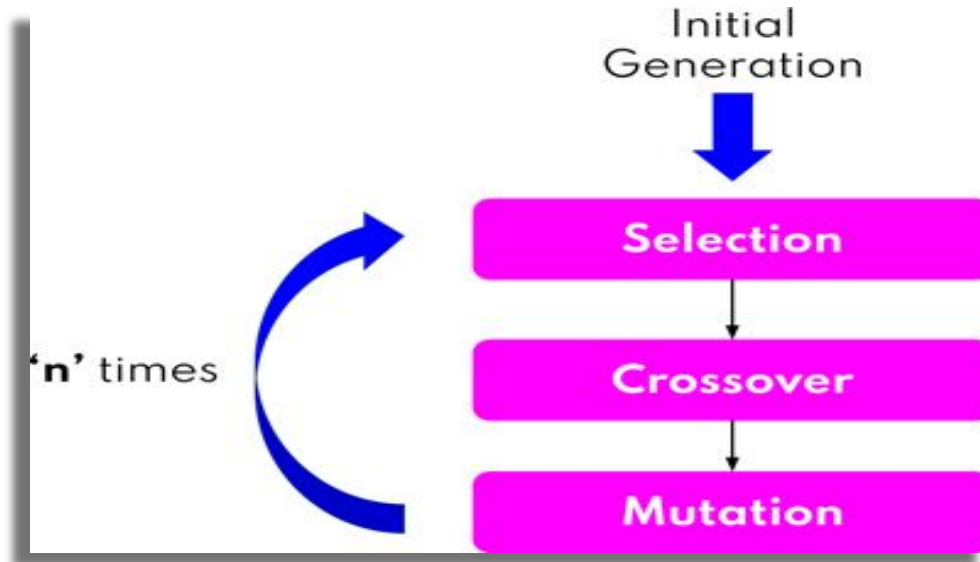
A search space can have three types of solutions, worst solution, suitable solution, and best solution. The genetic Algorithm will give us a suitable solution or best solution from the search space.





**As the Genetic algorithm is inspired by nature, it consists of the following three steps:**

- 1. Selection of the fittest through the tournament**
- 2. Crossover**
- 3. Mutation**



# Introduction to Colab

Nearly all machine learning and deep learning algorithms require good hardware. What if ... you don't have good hardware? Should you drop your dream to be a data scientist? No, there's an alternative Let us introduce you to Colab.

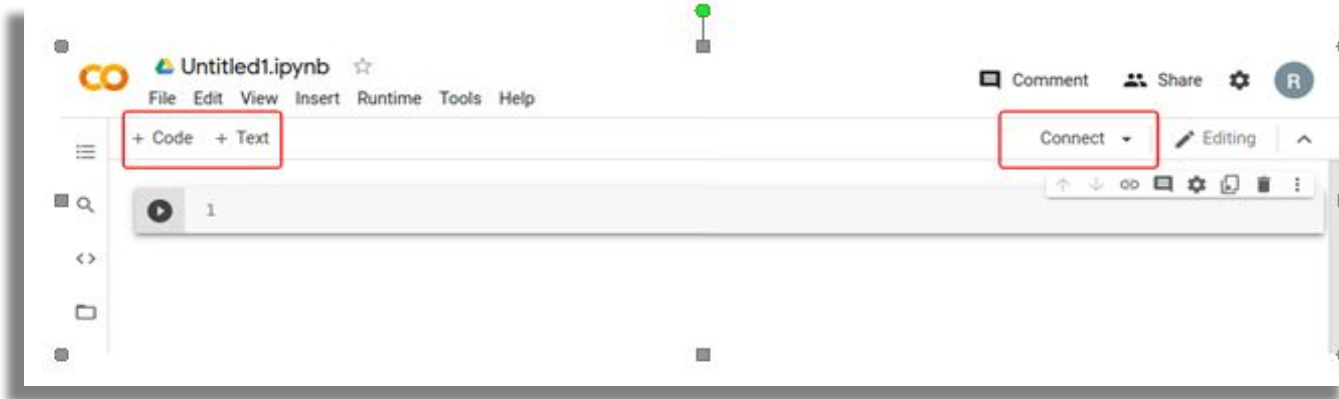
Colab is a service provided by Google which lets you access a virtual machine hosted on google servers. These virtual machines have dual core Xeon processors, with 12GB of RAM. You can even use GPU for your neural networks. Colab is an interactive python notebook (ipynb), which means that with writing python code, you can also write normal text, include images.

To create a new colab notebook, just go to "<https://colab.research.google.com>", and create a new notebook. You will get something like below

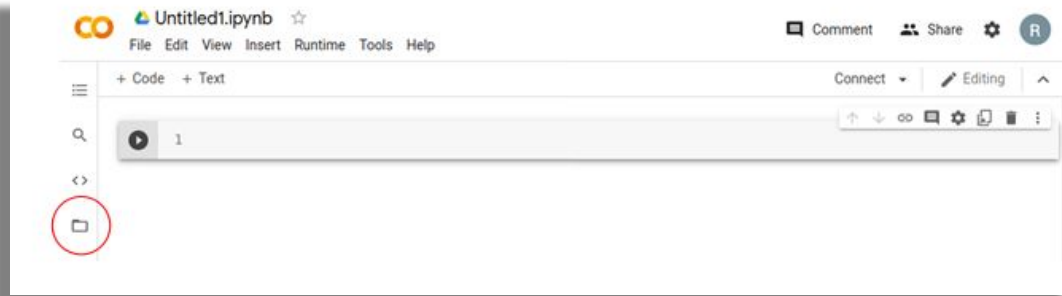
You can connect to runtime by clicking the "Connect" button in the right corner. You can add a new Code cell, or text cell using respective buttons in the toolbar.

## Uploading files

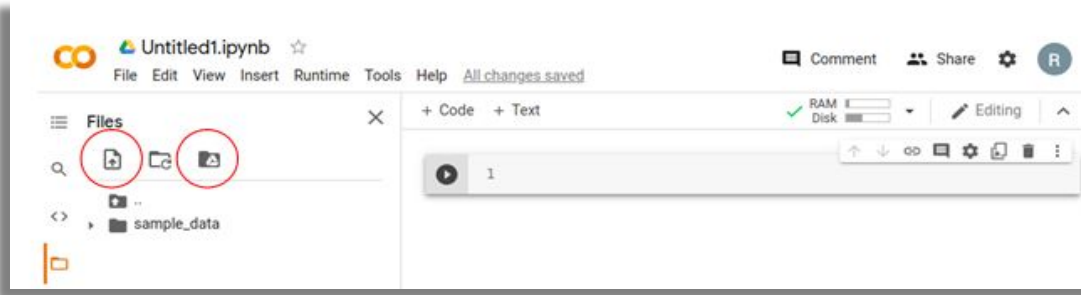
Sometimes you need to use a file from your PC. For that, you can upload the required files to google colab. Colab provides 100 GB in a collab session. If you want to access some files from your drive, you can even do so by connecting the drive to collab.



To upload something, open file pane from left toolbar.



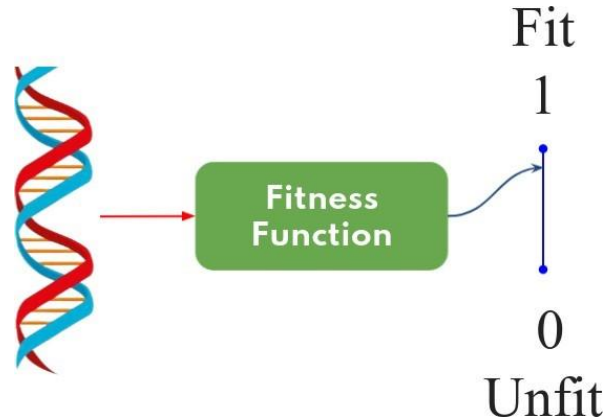
Now, just upload the file using first button, and you can also mount google drive using last button



# Fitness Function

The fitness function is a function that takes a gene as input and produces as output how “fit” or how “good” the solution is with respect to the problem in consideration. The fitness of a gene ranges from 0 to 1. 1 means that the gene is 100% fit.

For our problem of finding the best route, we need the route to have a minimum distance. So our fitness function will take a route, and calculate fitness as reciprocal of the total distance.



## Finding best route in a generation

```
def get_best_route(generation):  
    best_fitness = -1  
    best_route = None  
def get_best_route(generation):  
    best_fitness = -1  
    best_route = None  
    for route in generation:  
        if best_fitness < route.fitness:  
            best_fitness = route.fitness  
            best_route = route
```

# Natural Selection

It is a process through which the best traits in genes are passed to offspring. It ensures that only the best genes will get transmitted to the next generation. There are multiple ways we can code natural selection, we will be selecting the best genes through a tournament.

The tournament will take place between random  $k$  (we can decide as 2, 3, ...) genes will compete with each other, and the fittest among them will survive. There will be multiple tournaments to select the fittest genes and allow them to reproduce.



# Elitism

Like in real life, some space is reserved in the next population for elites. Elites fill these spaces without any competition with others. We can set this reservation in advance.

```
ELITE_SIZE = 50
TOURNAMENT_SIZE = 5

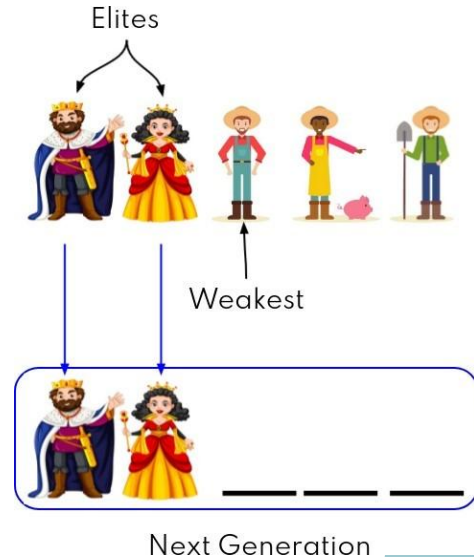
def select_generation(ranked_generation):

    selected = [ranked_generation[i] for i in
                range(ELITE_SIZE)]

    for i in range(len(ranked_generation) - ELITE_SIZE):

        random_k = random.sample(ranked_generation,
                                TOURNAMENT_SIZE) won = max(random_k,
                                                            key=operator.attrgetter('TfitnessT'))
        selected.append(won)

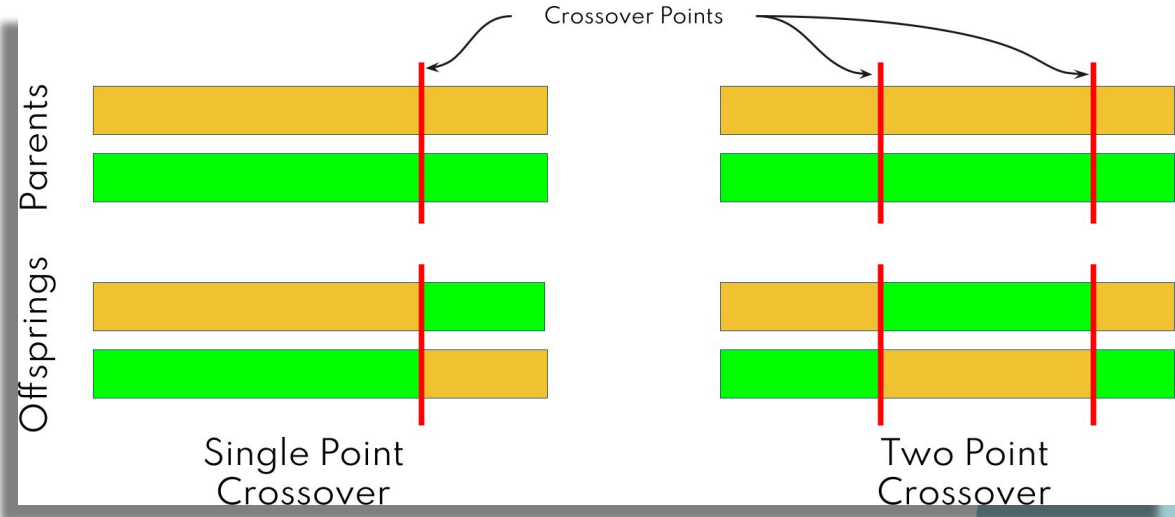
    return selected
```



# Crossover

It is a step in which two offspring are generated from two parents after their natural selection. The offspring will carry some genes of the first parent and some genes of the second parent. There are multiple ways to generate offspring:

1. Single point crossover
2. Two-point crossover

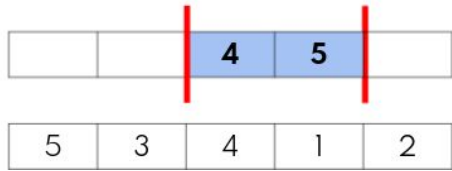


## pseudocode:

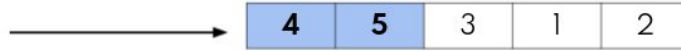
```
def mate(parent1, parent2):  
  
    i = random.randint(0, len(parent1)-1)  j = random.randint(0,  
    len(parent1)-1) min_index = min([i, j])  
    max_index = max([i, j])  
    parent1_gene = parent1[min_index:max_index] parent2_gene =  
    parent1[min_index:max_index]  
  
    child_one = [] child_one.extend(parent2_gene)  
    child_one.extend([city for city in parent1 if city not in parent2_gene])  
    child_two = [] child_two.extend(parent1_gene)  
    child_two.extend([city for city in parent2 if city not in parent1_gene])  
    child_one = Route(child_one) child_one.calculate_fitness()  
    child_two = Route(child_two) child_two.calculate_fitness() return  
    [child_one, child_two]  
  
def crossover(selection): random.shuffle(selection) new_generation = []  
    for i in range(0, GENERATION_SIZE, 2):  
        children = mate(selection[i], selection[i+1]) new_generation.extend(children)  
    return new_generation
```

As we are dealing with the route, so we can't just swap cities from two different routes. Instead of using simple two-point crossover we use ordered crossover.

Parents



First Offspring



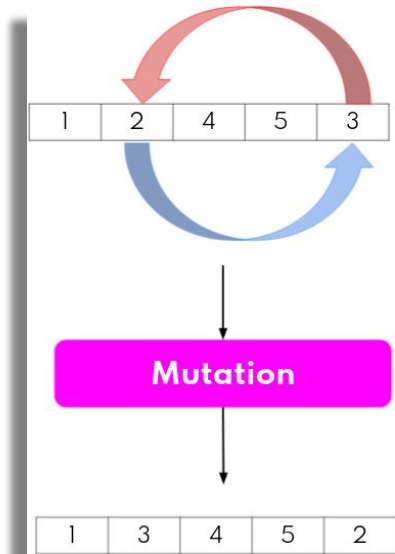
The above picture explains how ordered crossover works. We can observe that the crossover gene from the first parent is inserted in the beginning of the offspring, and the rest of gene is taken from the second parent.

# Mutation

A mutation is a change that occurs in our DNA sequence, either due to mistakes when the DNA is copied or as the result of environmental factors such as UV light and cigarette smoke. Mutation can result in many different types of change in sequences. Mutations in genes can have no effect, alter the product of a gene, or prevent the gene from functioning properly or completely.

With mutation, gene fitness can increase or decrease. If its fitness increases, then it will be transferred to the next generation. Mutation probability is very small, that's why we see differences among the same species of different generations.

In our problem, we will just swap random cities if want to mutate an offspring.



## pseudocode:

```
MUTATION_RATE = 0.01
def mutate_individual(individual):
    for i in range(len(individual)):
        if random.random() < MUTATION_RATE:
            random_j = random.randint(0, len(individual)-1) # swap cities in route
            city = individual[i]
            individual[i] = individual[random_j]
            individual[random_j] = city
    return individual

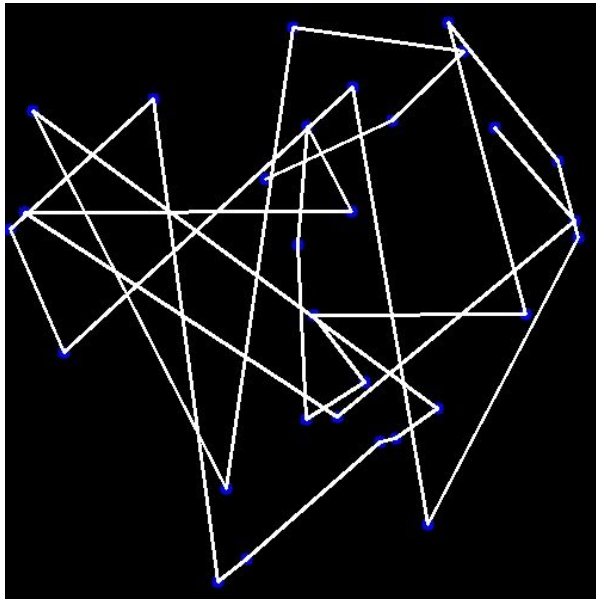
def mutation(crossovered):
    mutated = []
    for individual in crossovered:
        mutated.append(mutate_individual(individual))
    return mutated
```

```
def next_generation(current_generation):  
    ranked_generation = rank_generation(current_generation) selection =  
    select_generation(ranked_generation) crossovered = crossover(selection)  
    mutated = mutation(crossovered) return mutated  
  
cities = generate_cities() new_generation =  
first_generation(cities)  
current_best_route = get_best_route(new_generation)  
  
best_route = current_best_route for _ in  
range(GENERATIONS):  
    new_generation = next_generation(new_generation) current_best_route =  
    get_best_route(new_generation)  
  
    if current_best_route.fitness > best_route.fitness: best_route =  
        current_best_route
```



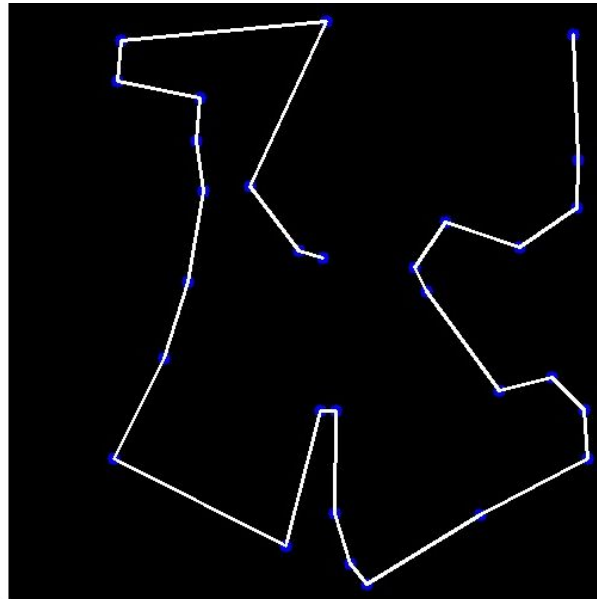
# Result:

Initial route



Distance: 5445.6

Suitable route



Distance: 2055.5



**THANK YOU !!!!**

