School of Computer Science and Engineering

J Component report

Programme : Integrated M.Tech Specialization in Business Analytics

Course Title : Supply Chain Analytics

Course Code : MGT3015

Slot : D2 + TD2

Title

Capacitated Vehicle Routing Problem for Last Mile Delivery

Team Members:

P B S S Jaswanth | 19MIA1039

Vamsidhar S | 19MIA1074

Mansoor Khan Lodi | 19MIA1094

Shashank R | 19MIA1105

Faculty: Stephan Thangaiah. I. S

Sign:

Date:

INDEX

ACKNOWLEDGMENT

Primarily, we would like to thank the almighty for all the blessings he showered over us to complete this project without any flaws.

The success and final outcome of this assignment required a lot of guidance and assistance from many people, and we are extremely fortunate to have got this all along with the completion of our project. Whatever we have done is only due to such guidance and assistance by our faculty, **Stephan Thangaiah I S**, to whom we are really thankful for giving us an opportunity to do this project.

Last but not the least; we are grateful to all our fellow classmates and our friends for the suggestions and support given to us throughout the completion of our project.

ABSTRACT

CVRP (Capacitated Vehicle Routing Problem) is a Vehicle Routing Problem in which vehicles with limited carrying capacity need to pick up or deliver items to various locations. The items have a quantity, such as weight or volume, and each vehicle has a maximum capacity that they can carry. The problem is to pick up or deliver the items with the optimal distance, while never exceeding the capacity of the vehicles.

For our project, we have framed a scenario where we are running a small – scale water can business with a limited number of vehicles. As ours is a small scale business, we would like to reduce the usage of resources and move towards an optimal stand in solving business problems. Our goal here is to reduce the usage of number of vehicles/drivers as well as the distance travelled per each vehicle in the process of delivering the water cans.

Our problem is treated as a graph problem where the places and the distances between them are in terms of nodes and edges. The distance between two places is in terms of straight line distance. In order to solve this problem, we will be using the Google OR Tools package with the help of Python programming

language to retrieve the optimal paths as well as the assignment of those paths to the drivers thus providing a solution to our CVRP.

INTRODUCTION

Last mile delivery, also known as last mile logistics, is the transportation of goods from a distribution hub to the final delivery destination — the door of the customer. The goal of last mile delivery logistics is to deliver the packages as affordably, quickly and accurately as possible

Where does last mile delivery fit in the order process?

Last mile delivery is the final logistics stage in the order process. It takes place after the products have been received, placed in the warehouse, sorted, picked, packed, and shipped to the appropriate distribution centres.

Last-mile delivery is all about shipping the products from delivery hubs directly to the customer's door.

Last mile is usually the most expensive part of the process — often costing more than half of overall shipping costs.

Let us understand what makes last mile delivery such a challenge?

Unlike with large-scale shipping, you're not sending a large number of products to a single location. Instead, your delivery drivers carry a large amount of smaller packages, each with unique destinations.

That is the essence of the last mile problem — **more stops mean more complex routes, more idle time, and more time on the road.** That means you have to maintain a larger fleet of delivery vehicles and drivers to ship a small number of products.

And **one of the factors that make Last Mile Delivery so expensive is** –

Complex routes lead to more out-of-route miles.

With a large number of individual stops, it's a lot easier for drivers to lose track of the route and rack up unnecessary miles.

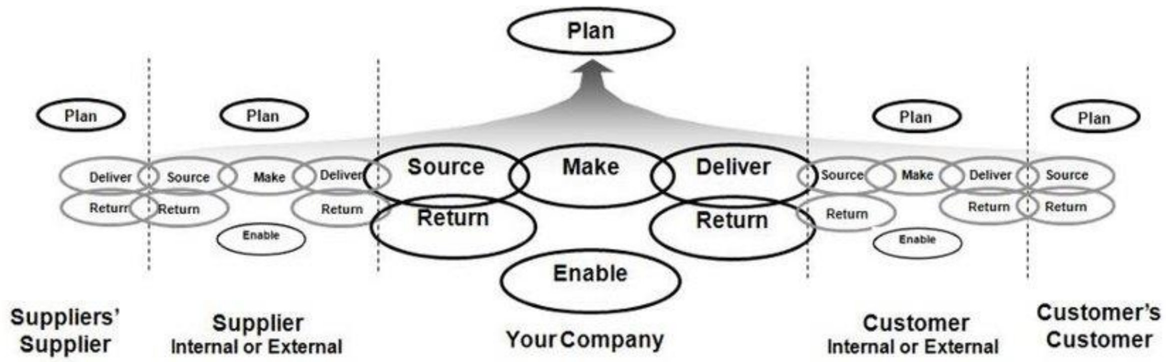
There are many variants of vehicle routing problem. Some of them are –

- **CVRP** (**Capacitated Vehicle Routing Problem**): Vehicles have a limited carrying capacity of the goods that must be delivered.
- **VRPTW** (**Vehicle Routing Problem with Time Windows**), **VR with pickup & delivery** etc.

For our project, we are focusing **on CVRP (Capacitated Vehicle Routing Problem)**

- **CVRP** is Vehicle Routing Problem in which vehicles with limited carrying capacity need to pick up or deliver items to various locations. The items have a quantity, such as weight or volume, and each vehicle has a maximum capacity that they can carry. The problem is to pick up or deliver the items with the optimal distance, while never exceeding the capacity of the vehicles.

SCOR MODEL



Area of focus in SCOR model:

1. Plan, Source, Make, Deliver and return are different areas in SCOR

2. This title focuses mainly on the **Deliver** area.

Reasons:

1. No of orders and destinations are already known.

2. Presence of warehouse/distribution (i.e. one or many)

3. Optimizations involved in delivering the product from warehouse to destinations.

PROBLEM STATEMENT

For our project, we have framed our own scenario.

Let us consider a small-scale Water Cans supply business "**Drink-Pure Water suppliers**" who supplies water cans to their registered users at the doorstep.

The supplier has **6 delivery vans** to deliver to its registered customers where each van can hold upto **15 water cans** at a time.

- **6 delivery drivers**
- **15 Water cans capacity per vehicle**
- **16 destinations** (to be delivered)
- **D0** is the distribution centre
- **1 Route per driver**

The supplier is required to cater to the demand quantity at the customers locations -

First value is zero (because you don't deliver anything in the centre)

- **Example:** The demand at the 1st location is **1 CAN**, similarly

  The demand at the 16th location is **8 CANS**

The following indicates the demand of water cans at all the 16 locations

- **Demand = [0, 1, 1, 2, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]**
- **Vehicle capacities = [15, 15, 15, 15, 15, 15] - i.e., 6 vehicles**

Following some of the constraints to solve this problem -

- Only one visit per vehicle per customer's location
- Depart from depot (i.e., starting from the shop)
- The delivery capacity of each vehicle should not exceed the maximum capacity

Objective:

- **Deliver all water cans with a minimum number of drivers**
- **Optimize the routing to minimize the distance covered per route**

DATASET

For this project, we had created our own dataset with the **Microsoft Excel**. It consists of 17 imaginary individual locations and the imaginary distances between them.

Our dataset contains a matrix of 17 rows and 17 columns.

- The first row and column id indicate our center and remaining 16 rows and columns ids are the customers who are registered under us. The values in the cell indicate the distance between our center to the respective customer house or distance from one customer house to another customer house.
- **For example**, if we look at the cell **d0d6** the value is **5020** (in meters) which means the distance between our center and the customer 4 is 5020 meters, in other case if we look at cell **d5d10** the value is **5820** (in meters) which means the distance between customer 5 and customer 10 is 5820 meters.

Here’s the screenshot of our dataset (Distance Matrix).

	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16
d0	0	5480	7760	6960	5820	2740	5020	1940	3080	1940	5360	5020	3880	3540	4680	7760	6620
d1	5480	0	6840	3080	1940	5020	7300	3540	6960	7420	10840	5940	4800	6740	10160	8680	12100
d2	7760	6840	0	9920	8780	5020	2740	8100	4680	7420	4000	12780	11640	11300	7880	15520	7540
d3	6960	3080	9920	0	1140	6500	8780	5020	8440	8900	12320	5140	6280	8220	11640	5600	13580
d4	5820	1940	8780	1140	0	5360	7640	3880	7300	7760	11180	4000	5140	7080	10500	6740	12440
d5	2740	5020	5020	6500	5360	0	2280	3080	1940	2400	5820	7760	6620	6280	5140	10500	7080
d6	5020	7300	2740	8780	7640	2280	0	5360	1940	4680	3540	10040	8900	8560	5140	12780	4800
d7	1940	3540	8100	5020	3880	3080	5360	0	3420	3880	7300	4680	3540	3200	6620	7420	8560
d8	3080	6960	4680	8440	7300	1940	1940	3420	0	2740	3880	8100	6960	6620	3200	10840	5140
d9	1940	7420	7420	8900	7760	2400	4680	3880	2740	0	3420	5360	4220	3880	2740	8100	4680
d10	5360	10840	4000	12320	11180	5820	3540	7300	3880	3420	0	8780	7640	7300	3880	11520	3540
d11	5020	5940	12780	5140	4000	7760	10040	4680	8100	5360	8780	0	1140	3080	6500	2740	8440
d12	3880	4800	11640	6280	5140	6620	8900	3540	6960	4220	7640	1140	0	1940	5360	3880	7300
d13	3540	6740	11300	8220	7080	6280	8560	3200	6620	3880	7300	3080	1940	0	3420	4220	5360
d14	4680	10160	7880	11640	10500	5140	5140	6620	3200	2740	3880	6500	5360	3420	0	7640	1940
d15	7760	8680	15520	5600	6740	10500	12780	7420	10840	8100	11520	2740	3880	4220	7640	0	7980
d16	6620	12100	7540	13580	12440	7080	4800	8560	5140	4680	3540	8440	7300	5360	1940	7980	0

TOOLS USED

The tools used in this project are:

1. Python (Programming language):

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

2. OR-Tools package:

OR-Tools is an open-source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming. **The different tools available under OR-Tools are:**

• A Constraint Programming solver

• A Linear Programming solvers

• Wrappers around commercial and other open-source solvers, including mixed integer solvers - CBC, CLP, CPLEX, GLPK, Gurobi, SCIP and XPRESS

• Graph algorithms: - shortest paths - min-cost flow - max flow - linear sum assignment

3. Pandas package:

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. This library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

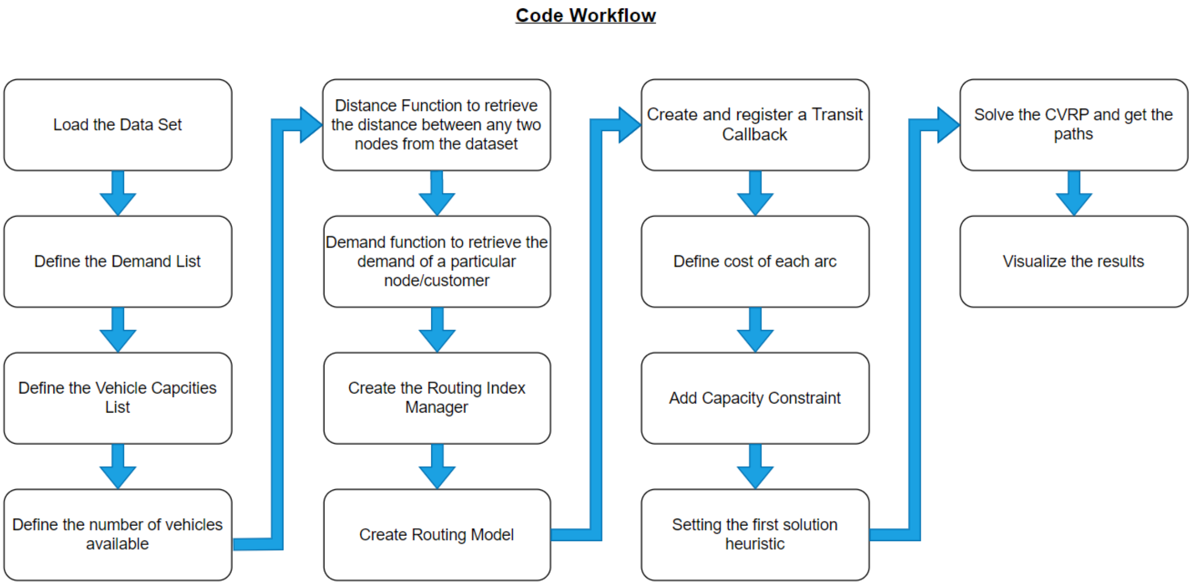
4. Numpy package:

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

WORKFLOW

Here's the workflow of our project mentioning the steps on how our algorithm solves the CVRP.



Now, let's have a look into the steps one by one.

- First, we load our **dataset** as well as we mention the **demand list of the customers (number of water cans required), the number of available vehicles and the capacity of each vehicle (the number of water cans up to which the vehicle can carry).**

- Then, we create two functions namely **distance** and **demand** where the distance function retrieves **the straight line distance between any two nodes/customers present in the dataset based on the indices of the nodes** and the demand function retrieves **the demand of a particular node/customer present in the dataset based on the index of the node.**

- Next comes the part where **Google-OR Tools** comes into the picture. Let's have a detailed look on how this **Google-OR Tools** solves the CVRP.

    - First, we create the **Routing Index Manager** which takes some basic information as parameters such as **the number of nodes (all the available nodes except the 0th node/central depot/shop – 16 in our case), number of available vehicles and the**

location/index of the central depot (which is 0 in our case) and then **creates indices for the nodes such as 0, 1, 2, 3,… and so on.**

- Then, we create an instance of the **Routing Model** which requires further parameters such as the **method to solve the CVRP** etc. It's like deploying a skeleton for now.

- Then, we create the **Register Transit Callback** which maintains a **register of the distances** between all the nodes available in the dataset.

- Then, we call the **SetArcCostEvaluatorOfAllVehicles** function to **define the cost (distance) of each arc/edge/path between all the pairs of nodes** available in the dataset.

- Then, we create the **Register Unary Transit Callback** for the **demands** which works in the same way how we created for the **distances.**

- Then, we define the **capacity constraints** based on the **demands** and **vehicle capacities.**

- Then, we add the **solving method** to the model. The solving method consists of two parts – **First Solution Strategy (Path Cheapest Arc in our case) and Local Search Metaheuristic (Guided Local Search in our case).**

- **Path Cheapest Arc** works in the below manner –
  Starting from a route "start" node, connect it to the node which produces the **cheapest route segment**, then extend the route by iterating on the last node added to the route.

- **Guided Local Search** works in the below manner –
  Uses **guided local search to escape local minima** (cf. http://en.wikipedia.org/wiki/Guided_Local_Search); this is generally the **most efficient metaheuristic** for vehicle routing.

In optimization problems, **escaping local minima is a must. Local minima** often **fools the algorithms** by portraying it as **global minima (the possible least value of the solution)** thus making the algorithms to stop the search further. A problem can have **"n" number of local minima, but, has only one global minima**. So methods such as **Guided Local Search** escape as many local minima as possible and finds out the global minima.

- Then, we make the model solve the CVRP.

- Then, we extract the paths from the solution and finally, we visualize the paths.

CODE SCREENSHOTS

Here are the screenshots of the python code implemented in this project.

In [2]:
```
# Imports
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

In [3]:
```
# Import Distance Matrix
df_distance = pd.read_excel('../input/d/vamsidharsivakumar/dataset/dataset.xlsx', index_col = 0)

# Transform to Numpy Array
distance_matrix = df_distance.to_numpy()
```

In [23]:
```
# Visualization of the distance matrix
G = nx.from_numpy_matrix(distance_matrix)
pos = nx.spring_layout(G)

color_map = []

for node in G:
    if node == 0:
        color_map.append('tomato')
    else:
        color_map.append('orange')

plt.figure(1, figsize = (20, 9))
nx.draw_networkx(G, pos, with_labels = True, node_color = color_map,
                 font_weight = 'normal', node_size = 1000)
plt.show()
```

In [5]:

```python
# Create dictionnary with data
data = {}
data['distance_matrix'] = distance_matrix
print("{:,} destinations".format(len(data['distance_matrix'][0]) - 1))

# Orders quantity (Water Cans)
data['demands'] = [0, 1, 1, 2, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
# Vehicles Capacities (Water Cans)
data['vehicle_capacities'] = [15, 15, 15, 15, 15, 15]
# Fleet informations
# Number of vehicles
data['num_vehicles'] = 6
# Location of the depot
data['depot'] = 0
```

```
16 destinations
```

In [6]:

```python
def distance(from_index, to_index):
    """Returns the distance between the two nodes."""
    # Convert from routing variable Index to distance matrix NodeIndex.
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return data['distance_matrix'][from_node][to_node]
```

In [7]:

```python
def demand(from_index):
    """Returns the demand of the node."""
    # Convert from routing variable Index to demands NodeIndex.
    from_node = manager.IndexToNode(from_index)
    return data['demands'][from_node]
```

In [8]:

```python
# Create the routing index manager.
manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
                                       data['num_vehicles'], data['depot'])

# Create Routing Model
routing = pywrapcp.RoutingModel(manager)

# Create and register a transit callback.
transit_callback_index = routing.RegisterTransitCallback(distance)

# Define cost of each arc.
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

# Add Capacity constraint.
demand_callback_index = routing.RegisterUnaryTransitCallback(demand)
routing.AddDimensionWithVehicleCapacity(demand_callback_index,
    0,  # null capacity slack
    data['vehicle_capacities'],  # vehicle maximum capacities
    True,  # start cumul to zero
    'Capacity')
```

```python
# Setting first solution heuristic.
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
search_parameters.local_search_metaheuristic = (
    routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
search_parameters.time_limit.FromSeconds(1)

# Solve the problem.
solution = routing.SolveWithParameters(search_parameters)
```

```
In [9]:
if solution:
    total_distance = 0
    total_load = 0
    for vehicle_id in range(data['num_vehicles']):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for driver {}:\n'.format(vehicle_id)
        route_distance = 0
        route_load = 0
        while not routing.IsEnd(index):
            node_index = manager.IndexToNode(index)
            route_load += data['demands'][node_index]
            plan_output += ' {0} Water Cans({1}) -> '.format(node_index, route_load)
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(
                previous_index, index, vehicle_id)
        plan_output += ' {0} Water Cans({1})\n'.format(manager.IndexToNode(index),
                                                      route_load)
        plan_output += 'Distance of the route: {} (m)\n'.format(route_distance)
        plan_output += 'Cargo Delivered: {} (Water Cans)\n'.format(route_load)
        print(plan_output)
```

```
        total_distance += route_distance
        total_load += route_load
    print('Total distance of all routes: {:,} (m)'.format(total_distance))
    print('Water Cans Delivered: {:,}/{:,}'.format(total_load, sum(data['demands'])))
else:
    print('No Solution')
```

RESULTS

Upon a successful run of our code, we can be able to see the optimal routes displayed for the available vehicles.

The below image displays the optimal paths representing the available vehicles (6 vehicles) along with the total distance of the routes as well as the quantity of water cans delivered.

```
Route for driver 0:
 0 Water Cans(0) ->  9 Water Cans(1) ->  10 Water Cans(3) ->  16 Water Cans(11) ->  14 Water Cans(15) ->  0 Water Cans(15)
Distance of the route: 15520 (m)
Cargo Delivered: 15 (Water Cans)

Route for driver 1:
 0 Water Cans(0) ->  12 Water Cans(2) ->  11 Water Cans(3) ->  15 Water Cans(11) ->  13 Water Cans(15) ->  0 Water Cans(15)
Distance of the route: 15520 (m)
Cargo Delivered: 15 (Water Cans)

Route for driver 2:
 0 Water Cans(0) ->  0 Water Cans(0)
Distance of the route: 0 (m)
Cargo Delivered: 0 (Water Cans)

Route for driver 3:
 0 Water Cans(0) ->  7 Water Cans(8) ->  1 Water Cans(9) ->  3 Water Cans(11) ->  4 Water Cans(15) ->  0 Water Cans(15)
Distance of the route: 15520 (m)
Cargo Delivered: 15 (Water Cans)

Route for driver 4:
 0 Water Cans(0) ->  0 Water Cans(0)
Distance of the route: 0 (m)
Cargo Delivered: 0 (Water Cans)

Route for driver 5:
 0 Water Cans(0) ->  8 Water Cans(8) ->  2 Water Cans(9) ->  6 Water Cans(13) ->  5 Water Cans(15) ->  0 Water Cans(15)
Distance of the route: 15520 (m)
Cargo Delivered: 15 (Water Cans)

Total distance of all routes: 62,080 (m)
Water Cans Delivered: 60/60
```

In the route for **Driver 0**, we have the path as –

0 Water Cans(0) -> 9 Water Cans(1) -> 10 Water Cans(3) -> 16 Water Cans(11) -> 14 Water Cans(15) -> 0 Water Cans(15)

The path route notations can be interpreted from the above path as –

First, the vehicle starts at Node (customer) **ID - 0** (depot) with 0 Water Cans delivered so far.

2) Then, the vehicle visits **Node ID - 9** with **1 Water Can** to be delivered here.

3) Then, the vehicle visits **Node ID – 10** with **2 Water Cans** to be delivered here.

4) Then, the vehicle visits **Node ID - 16** with **8 Water Cans** to be delivered here.

5) Then, the vehicle visits **Node ID - 14** with **4 Water Cans** to be delivered here.

6) Finally, the vehicle's trip gets terminated at Node ID - 15 as all the Water Cans loaded in the vehicle was delivered (**1 + 2 + 8 + 4 = 15** (Full Vehicle Capacity in this case)).

7) Similarly, the other drivers are also assigned an optimal route to deliver the water cans to meet the remaining demand from the customers.

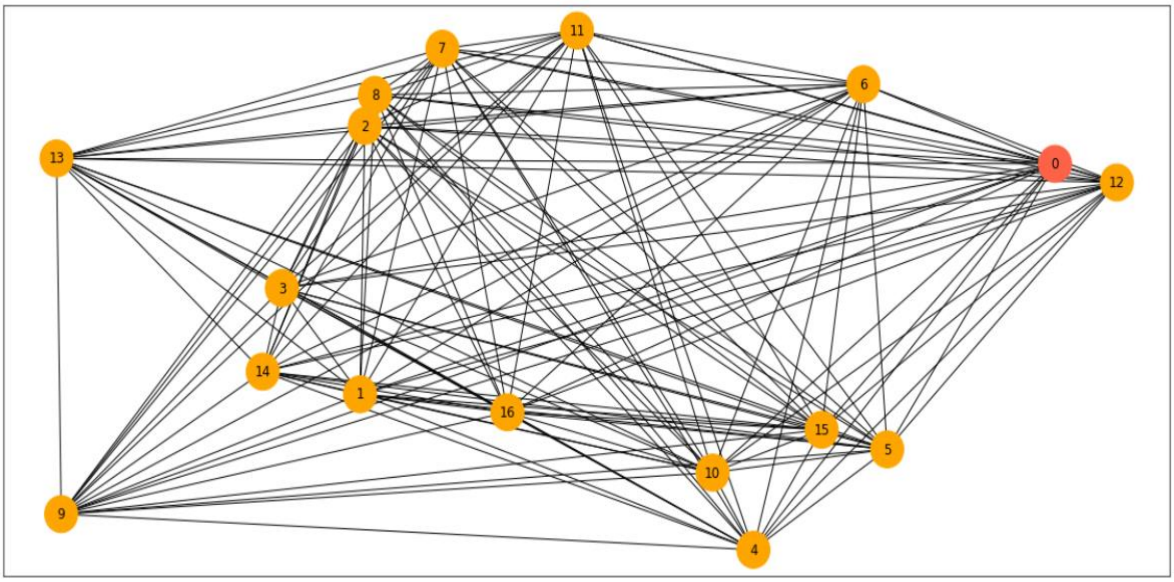
8) The total demand of the water cans from the customers side is **60 water cans** and as this demand value is satisfied with only 4 delivery vans driven by **4 drivers** each having a capacity to hold up to **15 water cans.**

9) So, the remaining 2 **drivers won't be allotted** any route to transport the goods from the distribution center as the demand is already satisfied by the other 4 drivers which helps in minimizing the number of drivers for the process.

Visualizations:

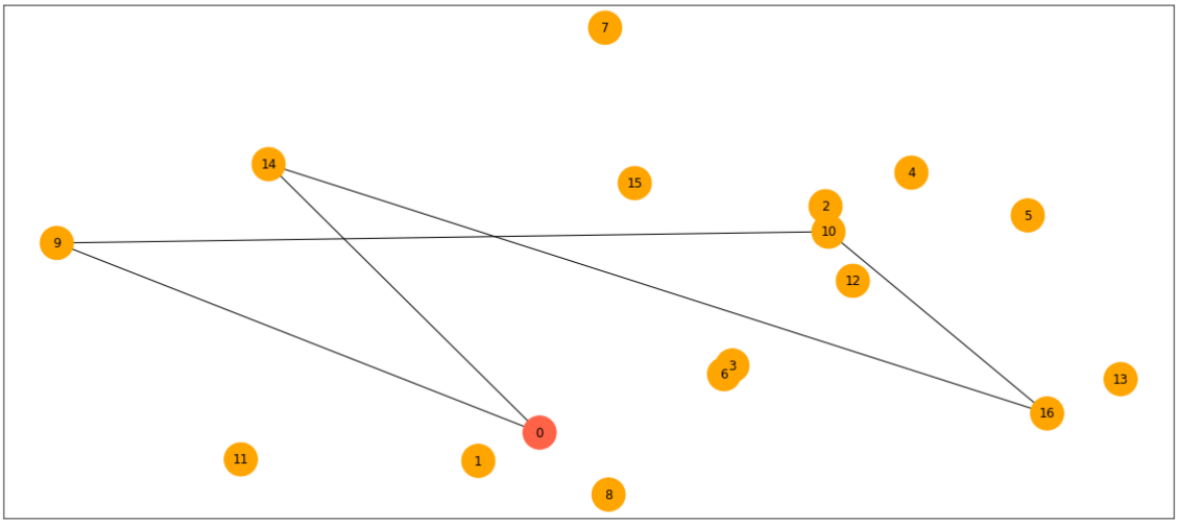
As we are reading the data from a distance matrix where the nodes do not have fixed coordinates, the graph keeps on changing per each run of the code. But, the distance between the nodes remains the same while only the location of the nodes keeps on changing.

1) **Main Network Graph comprising all the nodes**

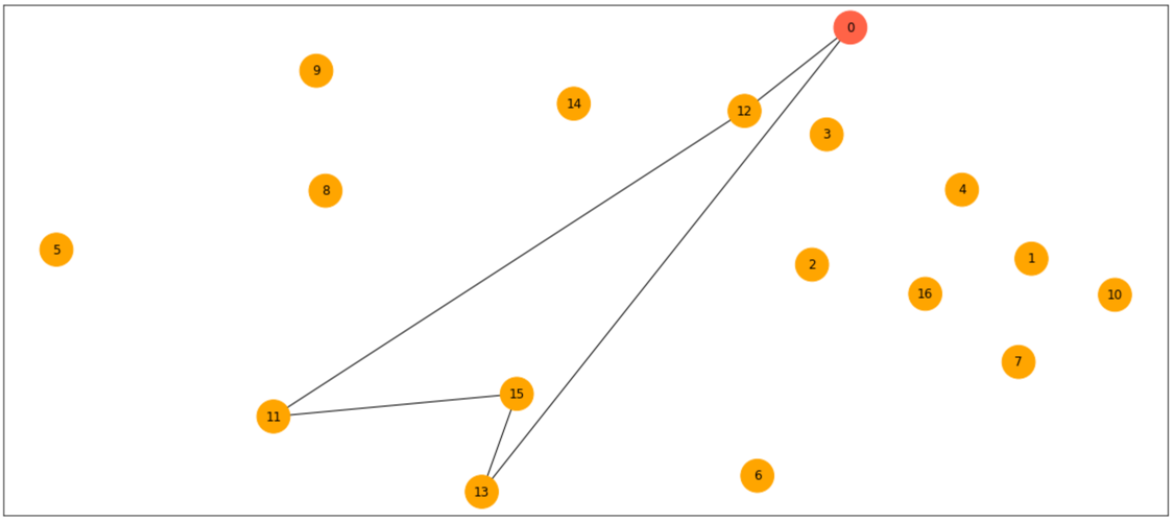


Here, node 0 (reddish colored node) is the central depot/shop and the rest of the nodes are the customers to be served.

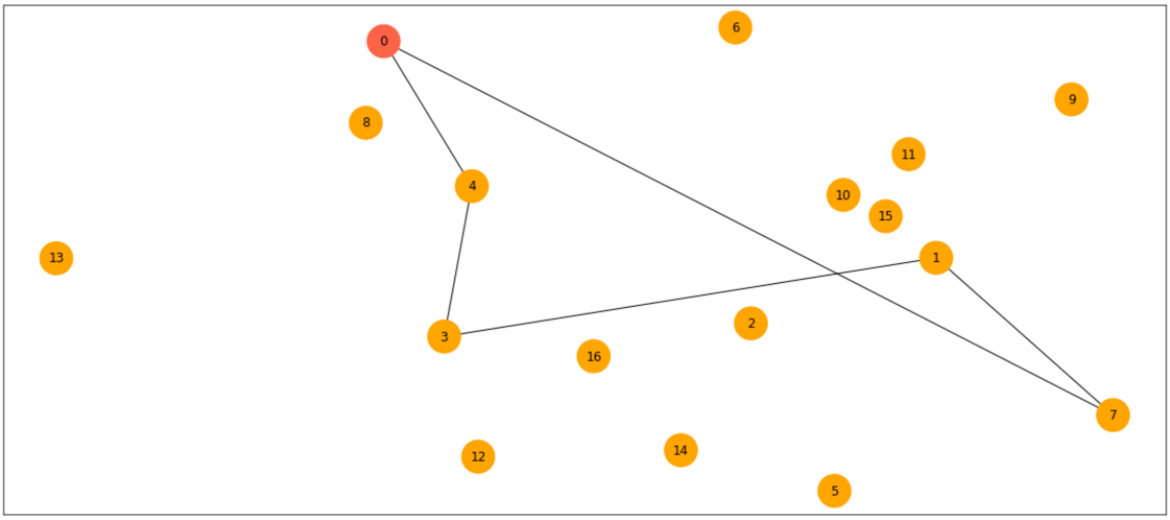
2) **Route for Driver 0 (0 -> 9 -> 10 -> 16 -> 14 -> 0)**



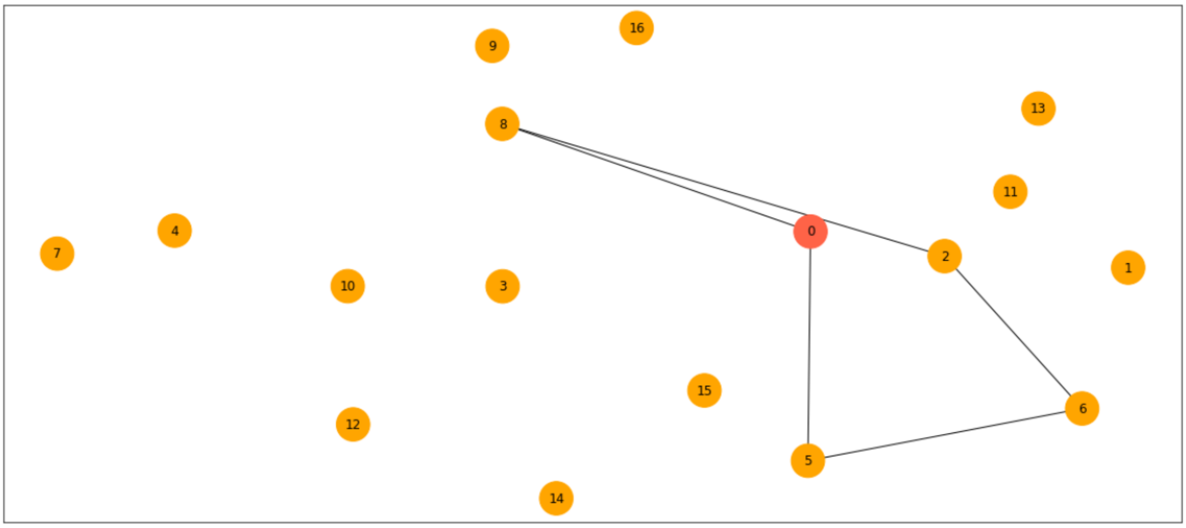
3) **Route for Driver 1 (0 -> 12 -> 11 -> 15 -> 13 -> 0)**



4) **Route for Driver 3 (0 -> 7 -> 1 -> 3 -> 4 -> 0)**



5) **Route for Driver 5 (0 -> 8 -> 2 -> 6 -> 5 -> 0)**



CONCLUSION

Delivering packages on time to the right address is essential to maintain a happy customer base (being able to track packages can help too).

By optimizing the process of transportation, we can also minimize failed deliveries, reduce costs, and improve the business scalability.

REFERENCES

Here are the sources, which we relied on for throughout the completion of this project.

1) https://developers.google.com/optimization/reference/constraint_solver/routing_index_manager
2) https://developers.google.com/optimization/routing/routing_options
3) https://developers.google.com/optimization/reference/constraint_solver/routing/RoutingModel
4) https://en.wikipedia.org/wiki/Vehicle_routing_problem
5) https://how-to.aimms.com/Articles/332/332-Formulation-CVRP.html