



RouteGenius: Optimizing Delivery Networks with CVRP Solutions

Alex Yang, Grace Eunji Kim, Sajuja Gangopadhyay
December 6, 2023

Agenda

- 1 Motivation
- 2 Formulation
- 3 Model
- 4 OptiGuide Q&A
- 5 Insights
- 6 Limitations



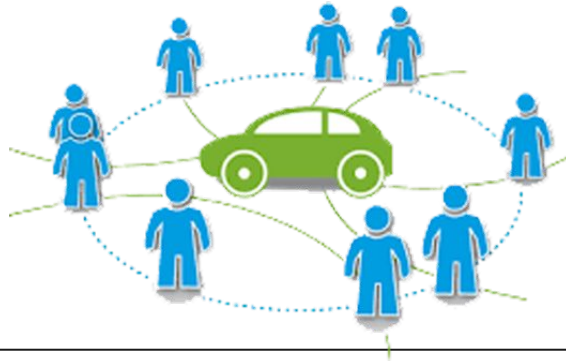
Motivation



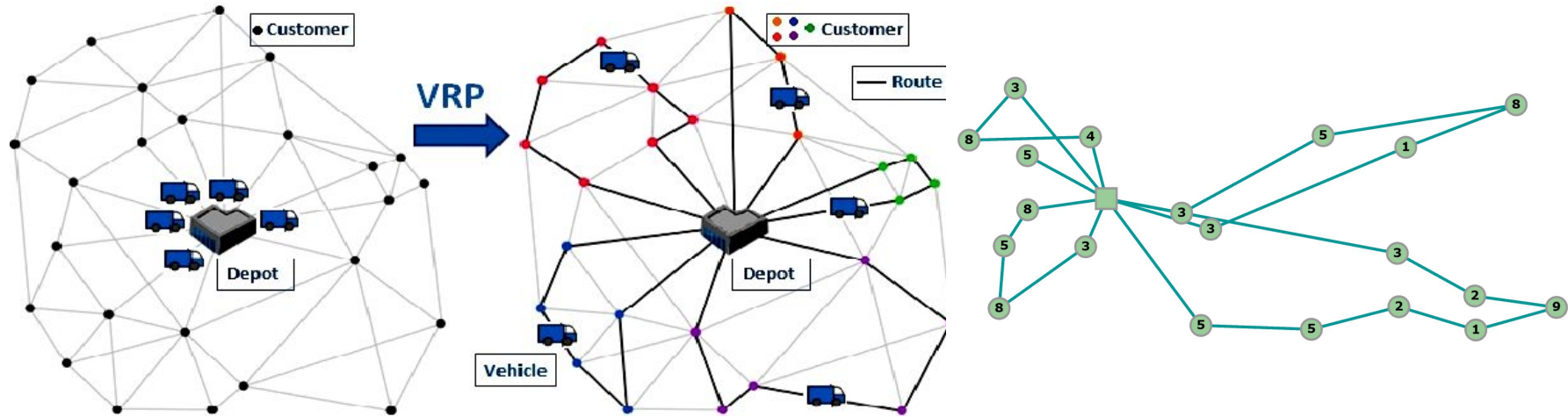
Transportation problem! → **Vehicle Routing Problem**



DOORDASH
GRUBHUB
Uber **Eats**



Main Q: What is the **optimal set of routes** for a fleet of vehicles to deliver to a set of customers?



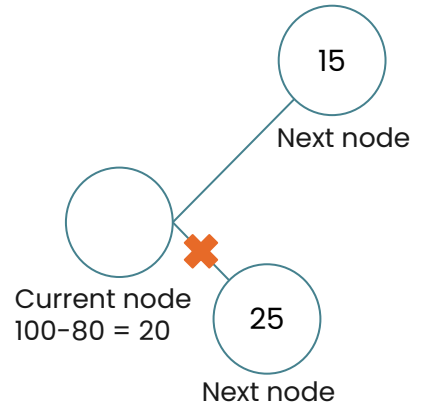
Formulation

- N = Set of nodes (including the depots and customers)
- V = Set of vehicles available
- c_{ij} = Cost (or distance) to travel from node i to node j
- d_i = Demand at node i , where i is a customer node.
- Q_v = Capacity of each vehicle $v \in V$ where V is the total number of vehicles in the fleet.

$$\text{Minimize } \sum_{i \in N} \sum_{j \in N} c_{ij} \cdot x_{ij} \quad \text{Total distance}$$

Subject to the following constraints:

- **Each customer node must be visited exactly once**
- **Subtour elimination:** ensures all routes are connected to the depot (each vehicle leaves and return to the depot)
- **Flow conservation/capacity:** ensures that the load at node j is limited by the capacity of vehicle v whenever a vehicle v travels from node i to node j .



Optimization Model using Gurobi

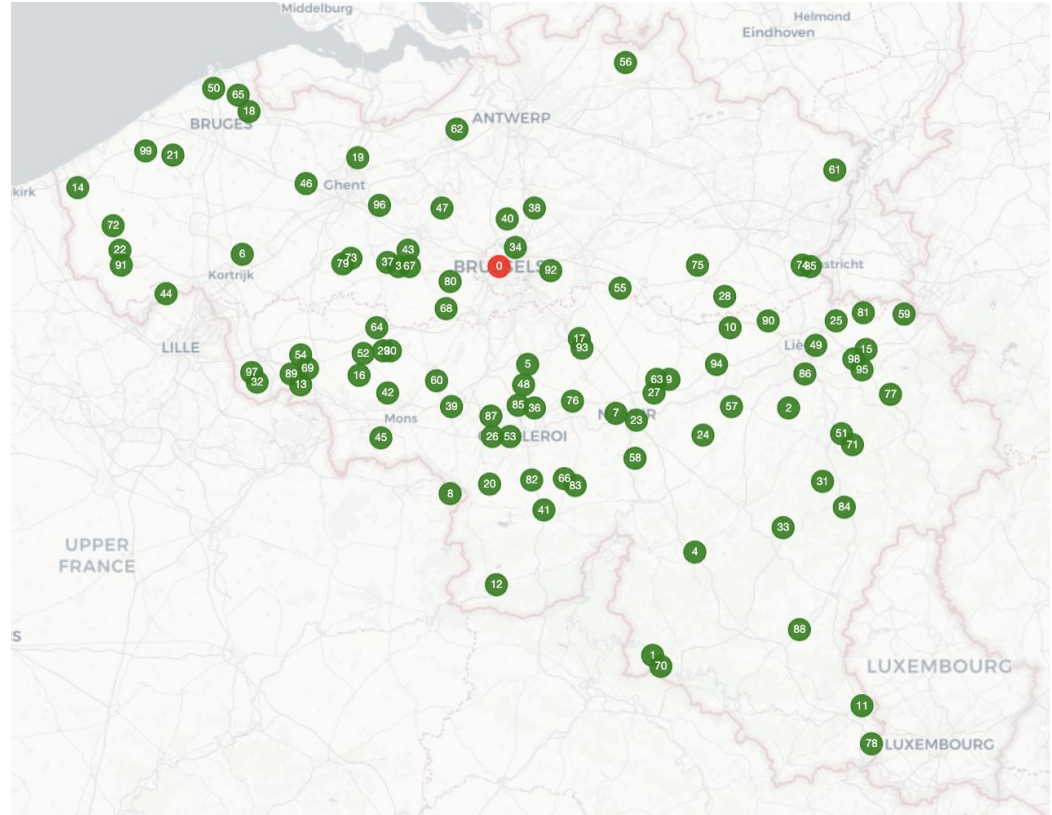


Belgium Dataset

Parameters:

- nodes = [0, 1, 2,...100]
- depot = nodes[0]
- demand
- distance_matrix (100x100)
- capacity = 250
- num_vehicles

id	lat	lon	nodeName	demand
0	50.842750	4.351550	BRUSSEL	0
1	49.841738	4.971051	ALLE	15
2	50.481299	5.519805	ANTHISNES	16
3	50.844199	3.943861	ASPELARE	31
4	50.108568	5.142958	AVE-ET-AUFFE	14



Original solution (n=100)

Gurobi Implementation

```
model = Model("CVRP")

# Variables
x = model.addVars(arcs, vtype=GRB.BINARY)
u = model.addVars(customers, vtype=GRB.CONTINUOUS)

# Objective: Minimize total distance
model.modelSense = GRB.MINIMIZE
model.setObjective(quicksum(x[i, j]*dist_matrix_dict[i, j] for i, j in arcs))

# Constraint 1: Each customer node visited exactly once
model.addConstrs(quicksum(x[i, j] for j in nodes if j != i) == 1 for i in customers)
model.addConstrs(quicksum(x[i, j] for i in nodes if i != j) == 1 for j in customers)

# Constraint 2: Flow Conservation Constraint
model.addConstrs((x[i, j] == 1) >> (u[i]+demands[j] == u[j])
| | | | | for i, j in arcs if i != 0 and j != 0)

# Constraint 3: Capacity Constraints
model.addConstrs(u[i] >= demands[i] for i in customers)
model.addConstrs(u[i] <= capacity for i in customers)

model.Params.MIPGap = 0.1
model.Params.TimeLimit = 500
model.optimize()
```

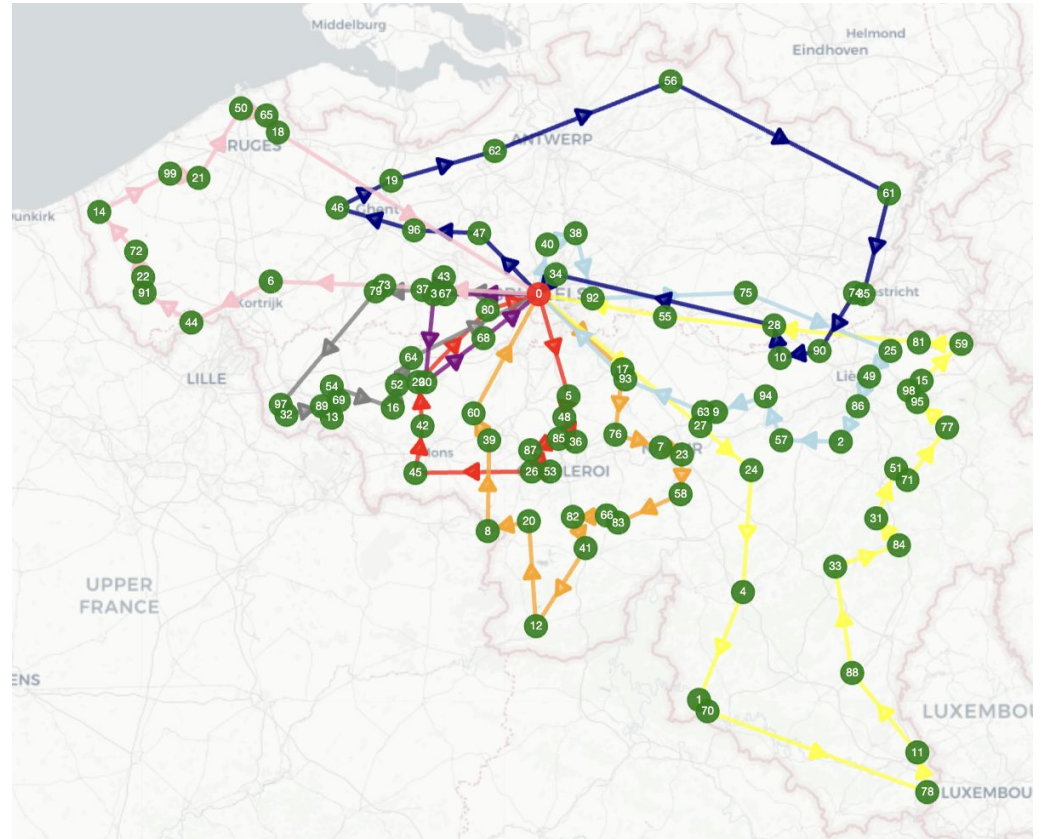
Original solution (n=100)

Gurobi Implementation

Optimal solution:

```
{0: [0, 5, 48, 36, 85, 87, 53, 26, 45, 42,
29, 80, 0],
1: [0, 17, 76, 7, 23, 58, 83, 66, 82, 41,
12, 20, 8, 39, 60, 0],
2: [0, 27, 24, 4, 1, 70, 78, 11, 88, 33,
84, 31, 51, 71, 77, 95, 98, 15, 59, 81, 55,
0],
3: [0, 37, 79, 97, 32, 89, 13, 69, 54, 16,
52, 64, 0],
4: [0, 40, 38, 92, 75, 25, 49, 86, 2, 57,
94, 9, 63, 93, 0],
5: [0, 47, 96, 46, 19, 62, 56, 61, 35, 74,
90, 10, 28, 34, 0],
6: [0, 67, 43, 3, 30, 68, 0],
7: [0, 73, 6, 44, 91, 22, 72, 14, 99, 21,
50, 65, 18, 0]}
```

Optimal solution value: 2418.00 km



Scaled-down dataset (n=14)

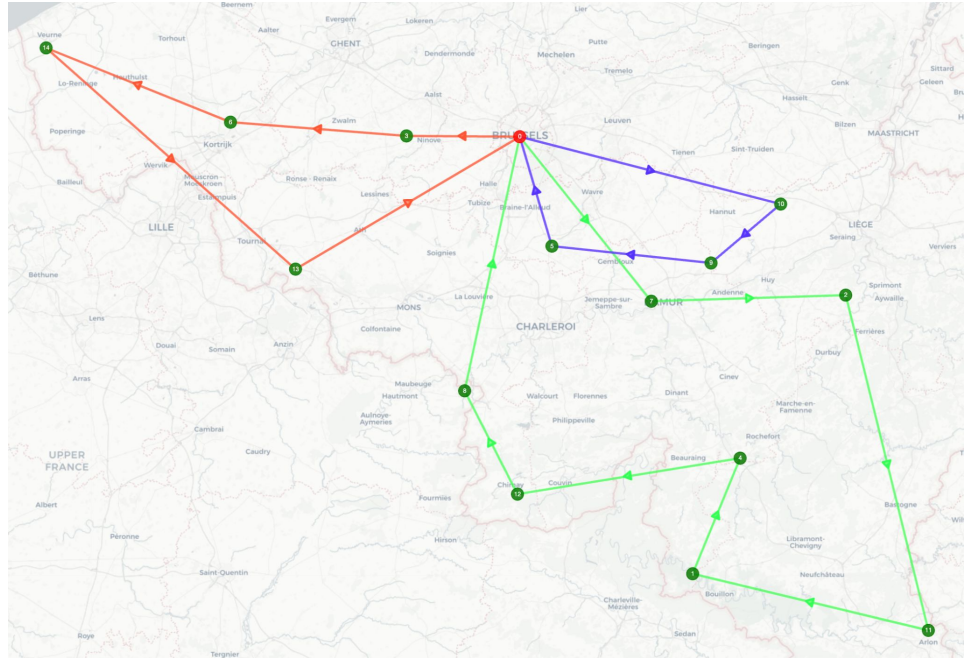
Parameters:

- nodes = [0, 1, 2,...**14**]
- depot = nodes[0]
- demand
- distance_matrix (14x14)
- capacity = **100**
- num_vehicles

Optimal solution:

```
{0: [0, 3, 6, 14, 13, 0],  
 1: [0, 7, 2, 11, 1, 4, 12, 8, 0],  
 2: [0, 10, 9, 5, 0]}
```

Optimal solution value: **1009.01 km**



Sensitivity Analysis



Different # of Vehicles (n=14)

With a total demand of 287, we start from 3 vehicles for 14 nodes.

- nodes = [0, 1, 2,...14]
- depot = nodes[0]
- demand
- Distance_matrix
- capacity = 100
- num_vehicles = range(3, 9)

Vehicles: 1 & 2 – infeasible

Vehicles: 3 Distance: 1009.005

Vehicles: 4 Distance: 1054.466

Vehicles: 5 Distance: 1117.08

Vehicles: 6 Distance: 1264.912

Vehicles: 7 Distance: 1339.801

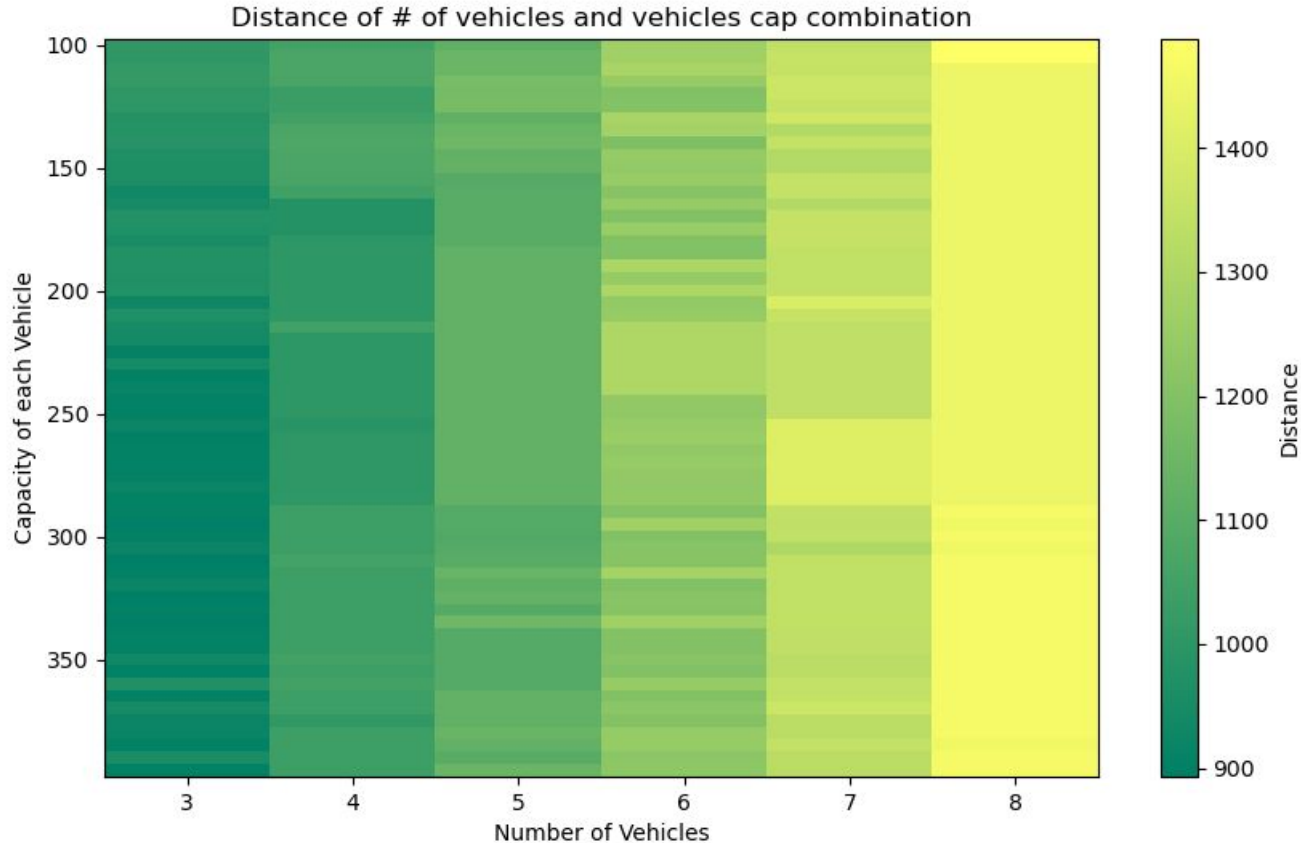
Vehicles: 8 Distance: 1487.549

Different Vehicle Capacity (n=14)

With a total demand of 287, we start from 3 vehicles for 14 nodes.

- nodes = [0, 1, 2,...14]
- depot = nodes[0]
- demand
- Distance_matrix
- capacity = range(100, 400, 50)
- num_vehicles = 3

Capacity: below 100 - infeasible
Capacity: 100 Distance: 1009.005
Capacity: 150 Distance: 962.493
Capacity: 200 Distance: 974.562
Capacity: 250 Distance: 902.75
Capacity: 300 Distance: 902.754
Capacity: 350 Distance: 934.515



Reasons for Sensitivity Analysis

1. Robustness Assessment
2. Feasibility Checking
3. Decision Support

Optiguide Q&A - What if?



OptiGuide: In-context Learning Questions

Question: What if the capacity of each vehicle is increased by 20 units?

Answer Code:

```
```python
Increase the capacity of each vehicle
additional_capacity = 20
for i in range(len(vehicle_capacity)):
 vehicle_capacity[i] += additional_capacity

Update the model's capacity constraints accordingly
for i in num_locations:
 model.addConstr(u[i] <= vehicle_capacity[0]) # Assuming all vehicles have the same updated capacity
```

Variables we can tweak:

- # of nodes
- Demand at nodes
- Capacity of vehicle
- # of vehicles

Question: What if the demand at location 3 increases by 5?

Answer Code:

```
```python
increase_in_demand = 5
demands[3] += increase_in_demand
```

Update the relevant constraints in the model

```
for k in range(num_vehicles):
    for i in range(1, num_locations):
        if i != j:
            model.addConstr(load[j, k] >= load[i, k] + demands[j] - (1 - x[i, j, k]) * vehicle_capacity[k])
```

Question: What if we discard location 2, what is the optimal solution value?

Answer Code:

```
```python
unavailable_location = 2

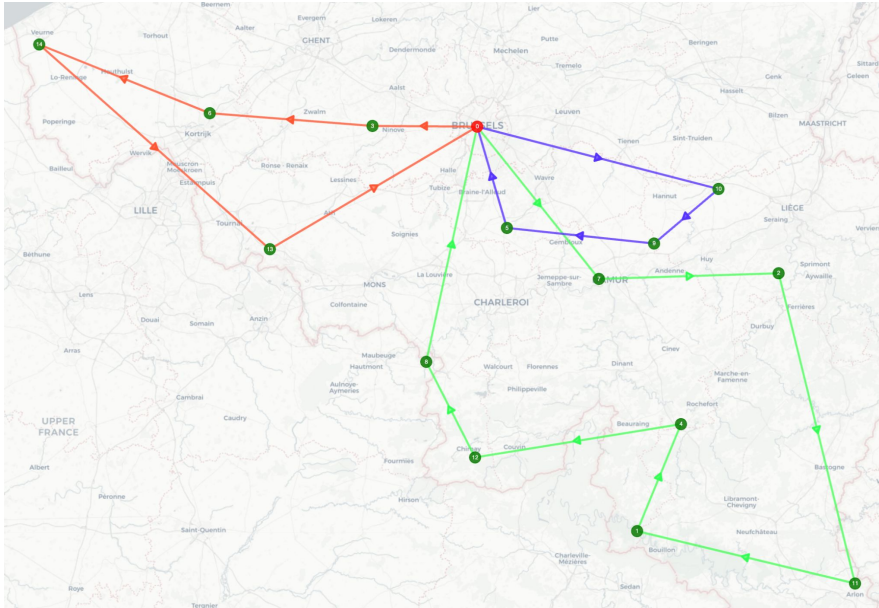
Remove all arcs involving the unavailable location
for i in V:
 if (i, unavailable_location) in x:
 model.remove(x[i, unavailable_location])
 if (unavailable_location, i) in x:
 model.remove(x[unavailable_location, i])
```

# Validation Questions

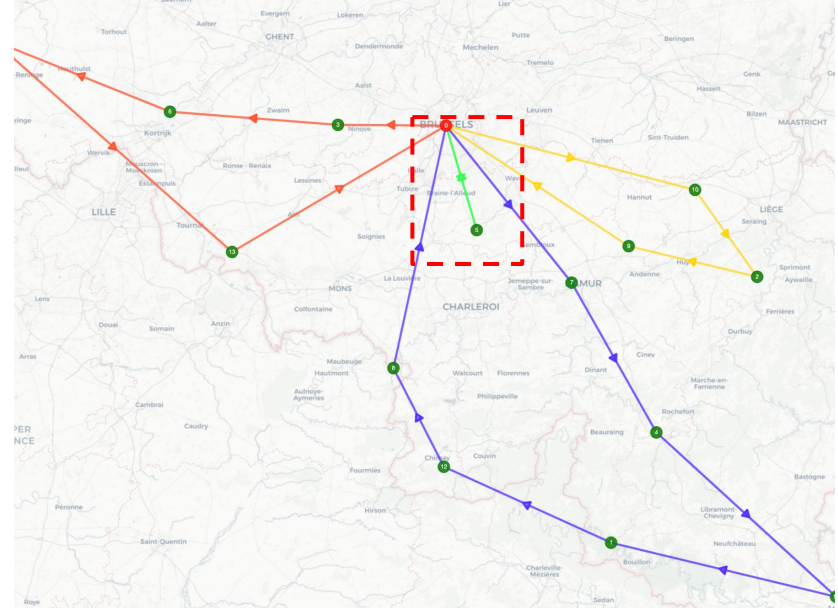


# Q1: Double **demand** at node 5

Before



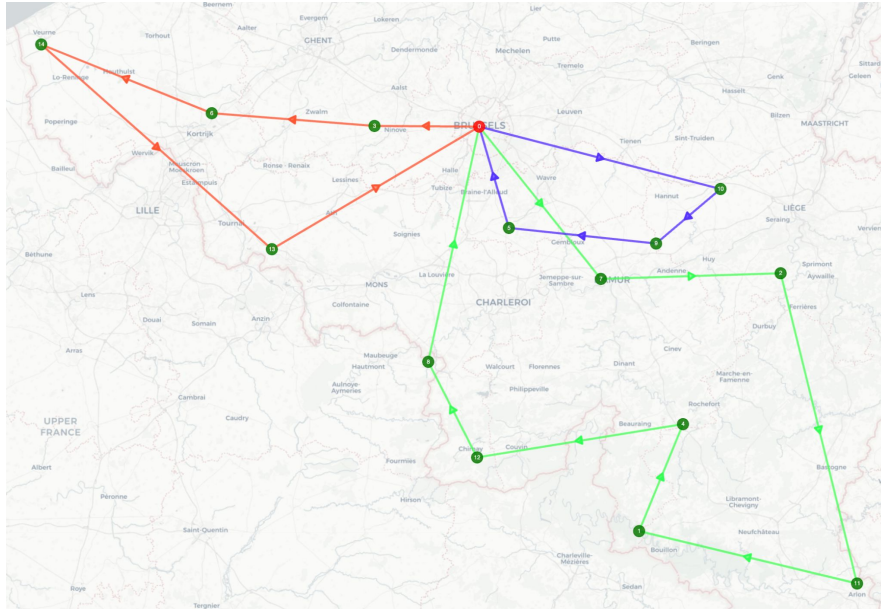
After



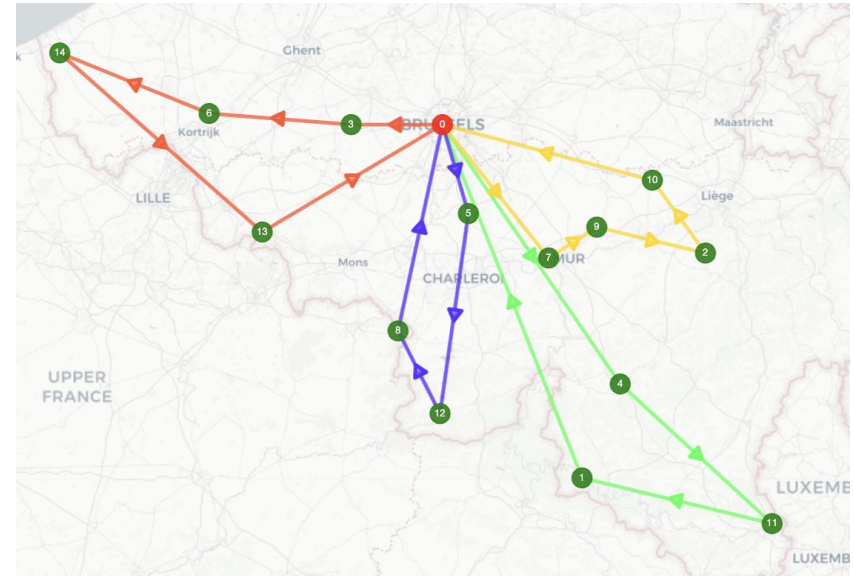
Comparing this with the original solution (1009.005), we can see that the change in demand for node 5 **has slightly increased the total distance of all routes**. However, it is important to note that the **impact of this change on the optimal solution is relatively small**.

## Q2: Increase **demand** at node 11 by 100 units

**Before**



**After**

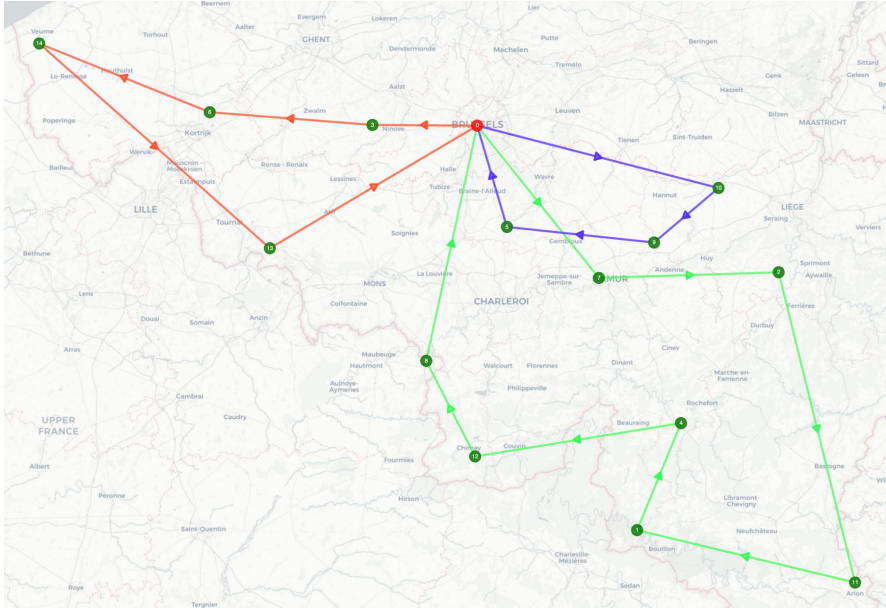


The original objective solution value was 1009.01. After increasing the demand at location 11 by 100 units, the new objective solution value is **1154.77**. This means that the increase in demand at location 11 has caused the **total distance of all routes to increase**.

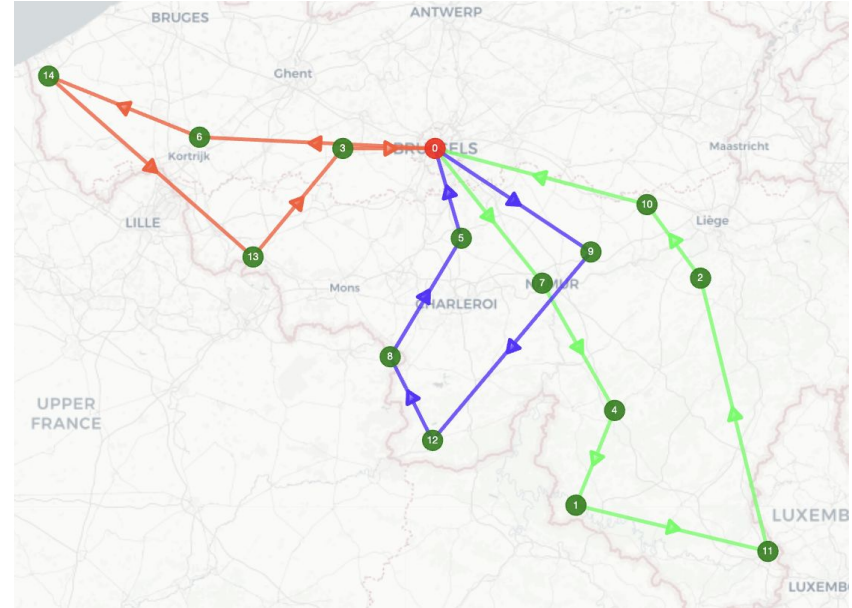


# Q3: Increase **demand** at every node by 20%?

**Before**



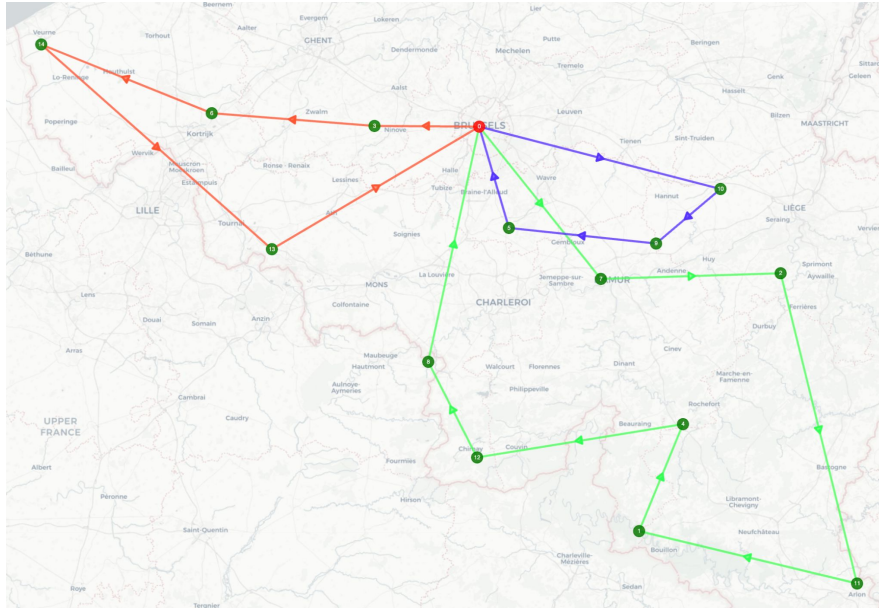
**After**



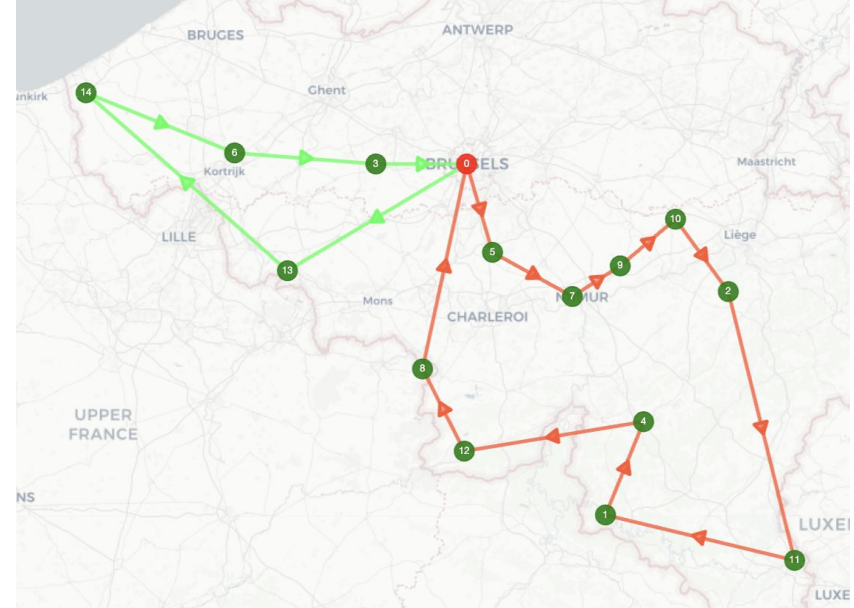
After increasing the demand of every node by 20%, the optimization problem was solved with an objective value of 1033.10. Comparing this result to the original result obtained in the beginning (1009.01), we can see that **increasing the demand of every node has resulted in a slightly higher objective value.**

# Q4: Double **capacity** of vehicle

Before



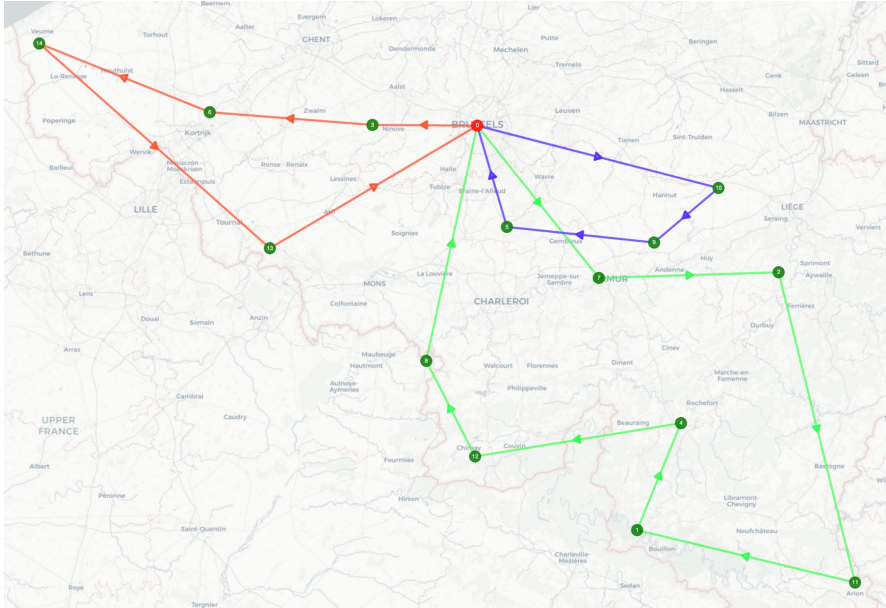
After



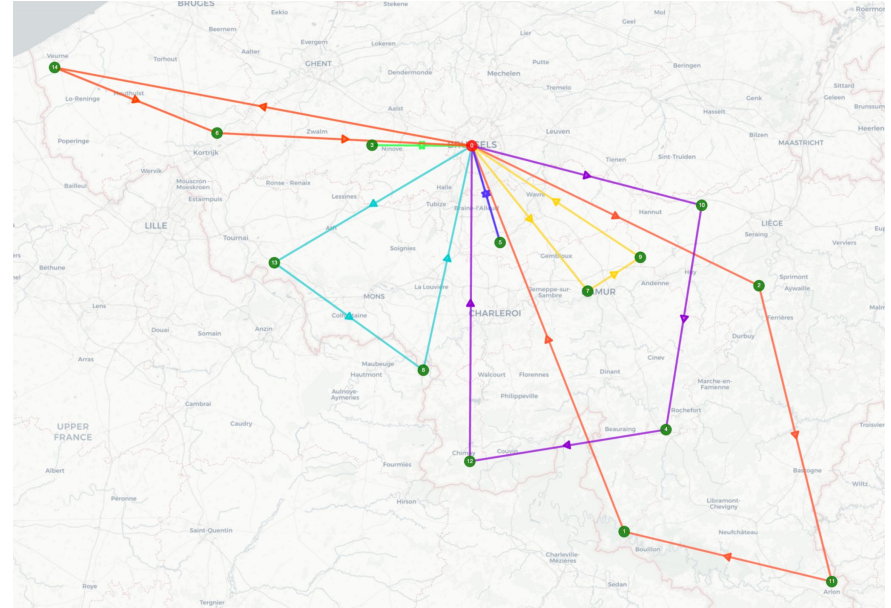
By doubling the capacity of all vehicles, we were able to solve the optimization problem with an objective value of **836.39**. This is a better objective value compared to the original result of 1009.01, indicating that **increasing the capacity has improved the efficiency of the solution**.

# Q5: If vehicles have differing **capacities** [50, 100, 150]?

**Before**



**After**

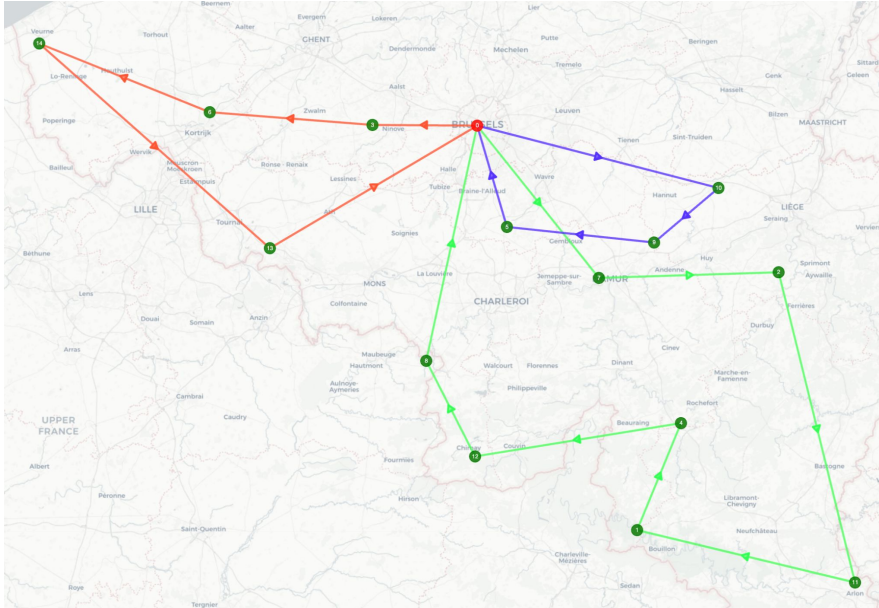


Based on the new information, the objective value obtained is **1497.97**. The new solution has a higher total distance of all routes compared to the original solution (1009.01). This is expected because the **total capacity of the vehicles in the new solution is higher, allowing for more demand to be served and potentially resulting in longer routes.**

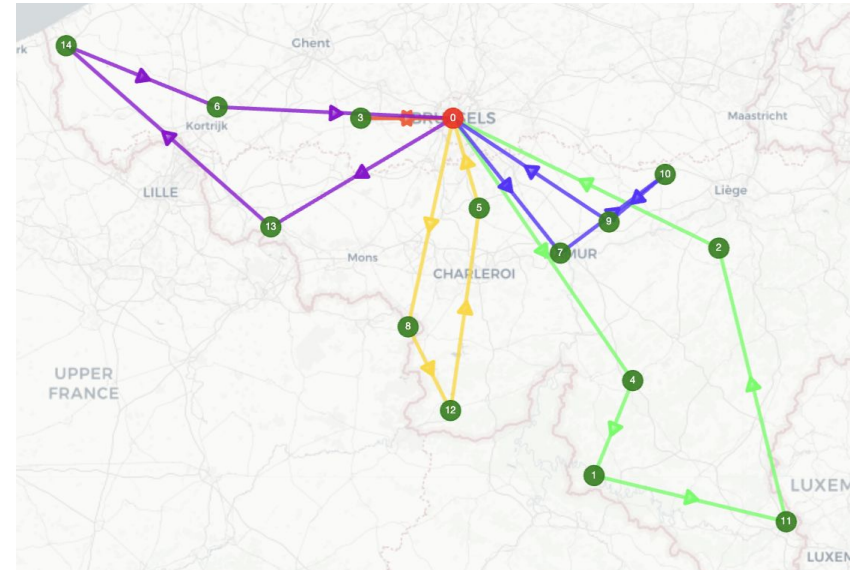


## Q6: Decrease **capacity** to 70, increase **# vehicles** to 5

Before



After

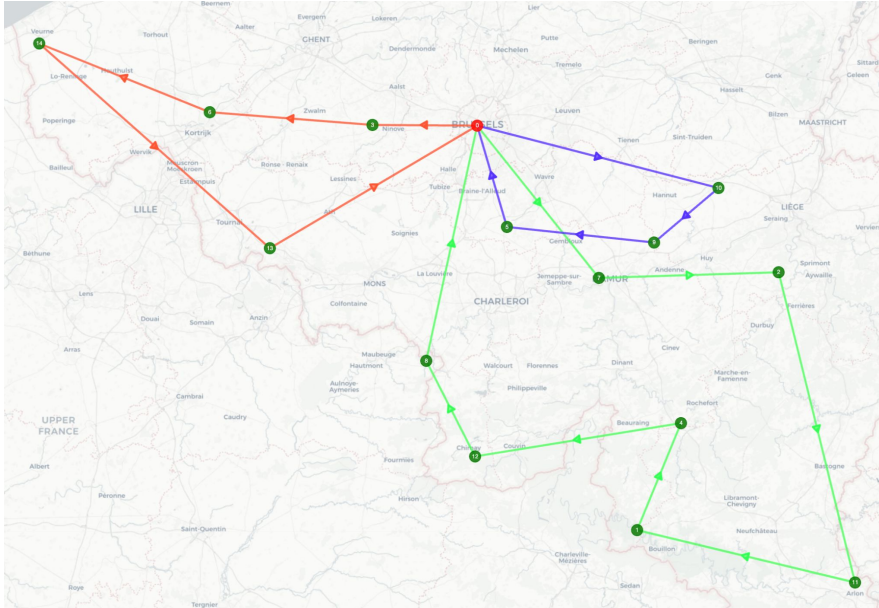


Comparing this with the original problem where the capacity was 100 and the number of vehicles was 1, the objective value has increased from 1009.005 to 1211.678. This implies that the **decrease in capacity and increase in the number of vehicles has led to a higher total distance traveled** in the optimal solution.



## Q7: If we **exclude two nodes (4,13)** from being served?

**Before**



**After**



# Exploratory Questions



## Q8: Integrate real-time traffic data?

### Optiguide Answer

# Here, you can integrate real-time traffic data into the CVRP model to optimize routes based on current traffic conditions.

# One approach is to update the distance matrix with the current traffic conditions.

# Suppose you have a function to get the **current traffic delay for each arc in the distance matrix**,  
# you can **update the distance\_matrix based on the delay factor**.

# For example, let's say you have a dictionary that maps each arc to its current traffic delay

```
traffic_delays = {
 (0, 1): 5.0, (0, 2): 2.5, # and so on
}
```

```
Update the distance matrix with the traffic delay
for arc, delay in traffic_delays.items():
 distance_matrix[arc] += delay_factor
```

# Q9: New objective: minimize # of vehicles used

## Optiguide Answer

The original objective of the code was to minimize the total distance traveled by vehicles in a supply chain problem. **The original result showed that the optimization problem was solved, and the objective value (total distance) was 1009.01.**

You wanted to modify the objective function to prioritize minimizing the number of vehicles used instead of the total distance. After making the necessary changes to the code, the modified result shows that the optimization problem was solved, and the objective value (number of vehicles used) is 1.0.

Comparing the original result to the modified result:

- Original Result: Total distance of all routes: 1009.01
- Modified Result: Number of vehicles used: 1.0

Therefore, by modifying the objective function, the code now prioritizes minimizing the number of vehicles used in the supply chain problem, resulting in the use of only 1 vehicle instead of optimizing for the total distance traveled.

# Insights & Limitations



# Insights

1. The model can **automatically come up with the optimized number of vehicles** to use without specifying it.
2. The model highlights **potential bottlenecks or congested areas** in the delivery network, providing insights into critical points where operational or infrastructure investments may be necessary.
3. The optimization model coupled with optiguide checks for the robustness of a CVRP to dynamic conditions, such as changes in demand patterns or unexpected disruptions.

# Challenges

1. Limited token for GPT to run with, hard to make questions and problems complex
2. Google *ortools* package does better job in solving transportation problems but not compatible with Optiguide
3. Prompt engineering was sensitive to deal with, especially given optiguide's limited scope

# Limitations

1. CVRP is an **NP-hard problem**, i.e., it becomes **computationally challenging** as the number of customers or vehicles increases. Solving large instances of CVRP optimally is impractical for real-world scenarios.
2. CVRP models rely on various input parameters, such as vehicle capacity, travel times, and demand for each customer. **The quality of the solution is sensitive to the accuracy of these parameters, and slight variations significantly impact the results.**
3. CVRP models, like ours, rely on predefined input data, but real-world scenarios involve uncertainties and dynamic changes. Integrating real-time data, such as traffic conditions or sudden changes in demand, is a challenge for traditional CVRP models.



**Thank you**  
**Any questions?**