

---

---

ROUTEGENIUS:  
OPTIMIZING DELIVERY NETWORKS WITH CVRP SOLUTIONS

---

---

DECEMBER 8, 2023

AUTHORS:

ALEX YANG<sup>1</sup>

GRACE EUNJI KIM<sup>2</sup>

SAJUJYA GANGOPADHYAY<sup>3</sup>

*MS IN PUBLIC POLICY AND MANAGEMENT: DATA ANALYTICS*

*MS IN HEALTHCARE ANALYTICS AND INFORMATION TECHNOLOGY*

*MS IN PUBLIC POLICY AND MANAGEMENT: DATA ANALYTICS*

---

<sup>1</sup>Alex Yang: OptiGuide Configuration and Querying

<sup>2</sup>Grace Eunji Kim: CVRP Optimization using Gurobi and visualization of network graphs using Veroviz

<sup>3</sup>Sajujya Gangopadhyay: OptiGuide Querying, Random Allocation Solution, Mathematical Formulation of CVRP

# 1 Executive Summary

RouteGenius is a cutting-edge solution that revolutionizes delivery network optimization through the implementation of the Capacitated Vehicle Routing Problem (CVRP) using Optimization through Gurobi, and an interface to converse with the model using a Natural Language Wrapper (OptiGuide).

In the rapidly evolving landscape of logistics and supply chain management, efficient route planning is a critical factor in reducing costs and enhancing overall operational effectiveness. RouteGenius addresses this challenge by seamlessly integrating CVRP algorithms to create a dynamic and adaptive system that continually optimizes delivery routes with changing demand, vehicle capacities, and even integration with real-time traffic data.

## 1.1 Key Features

- **CVRP Optimization:** RouteGenius employs CVRP algorithms to efficiently allocate resources and plan delivery routes, considering vehicle capacity constraints and optimizing for the shortest paths. This leads to reduced transportation costs and optimized route allocation which can be used by transportation services, ride-share applications and food-delivery and logistics companies like Uber, Doordash, and FedEx.
- **Real-time Adaptability:** The system, coupled with the language wrapper (Optiguide) is designed to dynamically adapt to changing conditions in real-time through human feedback in the form of both rudimentary and exploratory questions. It changes the underlying code to factor in additional constraints and changes in demand while simultaneously ensuring that routes remain optimal and responsive to the evolving operational environment.
- **Sustainability:** By minimizing fuel consumption and carbon emissions through route optimization, RouteGenius supports environmentally conscious and sustainable logistics practices.

In conclusion, RouteGenius is a game-changing solution for organizations seeking to optimize their delivery networks. By integrating CVRP algorithms and Natural Language technologies, it offers a comprehensive and adaptive approach to route planning, resulting in substantial cost savings, operational efficiency, and a positive impact on environmental sustainability.

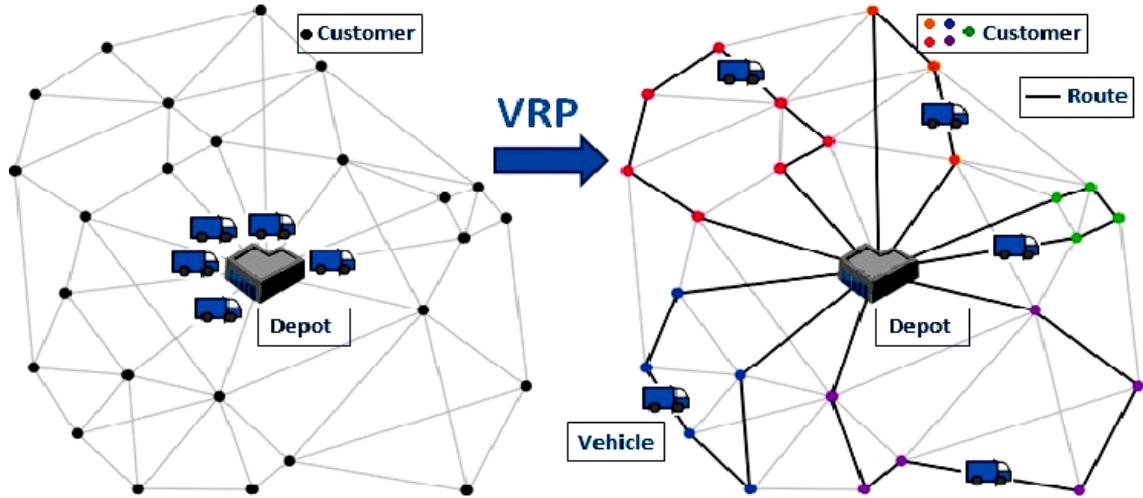
## 2 Analytical Model and Data

### 2.1 Problem Statement

In the dynamic landscape of delivery services, the effective management of delivery routes poses a critical challenge for companies striving to meet customer demands and minimize operational costs. Our model addresses this challenge through the implementation of the Capacitated Vehicle Routing problem (CVRP).

The CVRP pertains to the operations of any company that dwells with logistic operations involving the transportation of goods from one depot to customers using a predefined set of vehicles on a road network. Therefore, the primary question that motivates this problem is: *What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?*

The primary objective is to develop an optimized set of routes (solution), with each route tailored for a specific vehicle, ensuring that all customer requirements and operational constraints (i.e., limited load capacity) are met while simultaneously minimizing the overall transportation cost (objective function). The optimized cost to travel from node  $i$  to node  $j$  would be the distance between the two nodes that is travelled along the optimal route.



### 2.2 Mathematical Formulation

To define the mathematical formulation of the VRP, we would first need to define a set of parameters:

- $N$  = 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.

Now, let's define the decision variables:

- $x_{ij}$  = Binary decision variables, which equals 1 if the vehicle travels from node  $i$  to node  $j$ , 0 otherwise.
- $u_i$  = The cumulative load at node  $i$ , representing the load or demand accumulated up to node  $i$ . This is an auxiliary variable introduced to keep track of the cumulative load on the vehicles' routes.

The constraints relating to  $u_i$  and  $d_i$  in the formulation are used to ensure that the load on a vehicle doesn't exceed its capacity as it moves from one node to another. The idea is to model the accumulation of load along the routes and make sure it doesn't surpass the vehicle's capacity at any point.

Now, let's define the objective function:

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

Subject to the following constraints:

- **Each customer node must be visited exactly once:**

$$\sum_{i \in N} \sum_{j \in N} x_{ij} = 1 \quad \text{for } j \in N, j \neq 0$$

- **Each vehicle must leave and return to the depot (node 0):**

$$\sum_{i \in N, i \neq 0} x_{i0} = \sum_{i \in N, i \neq 0} x_{0i} = |V|$$

- **Flow Conservation Constraint:** This constraint 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  $j$ .

Assumption: There is no overlap of nodes between the vehicles. If the demand of a node exceeds the current supply of the vehicle, the vehicle will not visit the node and let another vehicle that has enough supply to visit.

$$u_i + d_i - u_j \leq Q_v \cdot (1 - x_{ij}) \quad \forall i, j \in N, i \neq 0, j \neq 0, v \in V$$

- $Q_v$  = Capacity of a vehicle  $v$  where  $V$  is the total number of vehicles available.
- $u_0 = 0$  (since depot has zero load)
- $Q \geq u_i \geq 0 \quad \forall i \in N$
- **Vehicle availability constraint:** This constraint ensures that the total number of vehicles used aligns with the available fleet capacity.

$$\sum_{v \in V} \sum_{i \in N, i \neq 0} x_{i0} = \sum_{v \in V} \sum_{i \in N, i \neq 0} x_{0i} = |V|$$

## 2.3 Data Summary

The dataset we used consists of 100 locations in Belgium, with a specified demand at each node. The distance matrix is also given, calculated with real-road distances. The capacity of each vehicle is defined as 250.

Parameters:

- nodes = [0, 1, 2,...100]
- depot = nodes[0]
- demand
- distance matrix (100 x 100)
- capacity = 250
- number of vehicles

### Down-scaled Dataset

In order to make our model compatible with OptiGuide, the value of the variables need to be hardcoded. Hence, we need to down-scale our original dataset to hold the whole model string under the token limitation of GPT-3.5. We chose a subset of our dataset (the first 15 nodes, including the depot) as the dataset for our baseline model.

- nodes = [0, 1, 2, ..., 14]
- depot = nodes[0]

- demand = demands from the first 14 nodes  $\{(1: 15, 2: 16, 3: 31, 4: 14, 5: 33, \dots)\}$
- distance matrix (15 x 15)  
 $\{((0, 0): 0.0, (0, 1): 142.045, (0, 2): 101.756, (0, 3): 31.554, (0, 4): 106.986, \dots)\}$
- capacity = 100

## 2.4 Random and Optimal Solutions

Initially, we used the entire Belgium dataset of 100 nodes to find the optimal solution with optimization using Gurobi. Below we can see a visualization of the optimal solution.

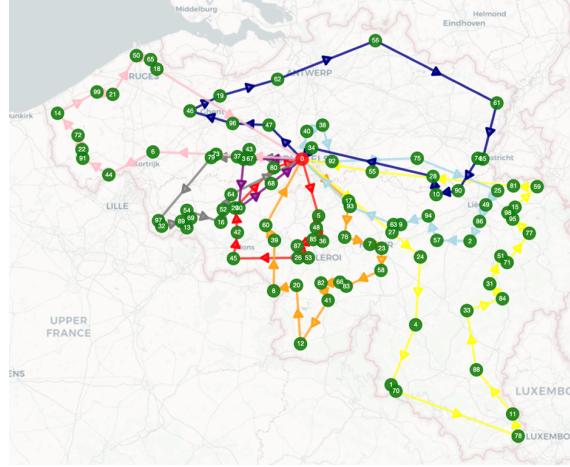


Figure 1: Optimal Solution of the Original Dataset

After scaling down the dataset to 14 nodes that could be fed into OptiGuide, we used two methods to solve the CVRP.

- Random Allocation Solution (Optimal Cost : 1857.462 km)
- Optimal Solution using Gurobi (Optimal Cost : 1009 km)

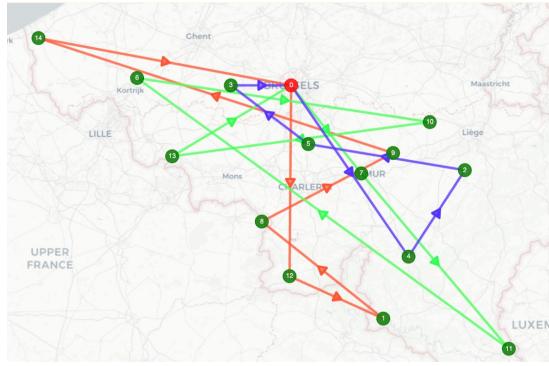


Figure 2: Random Allocation Solution

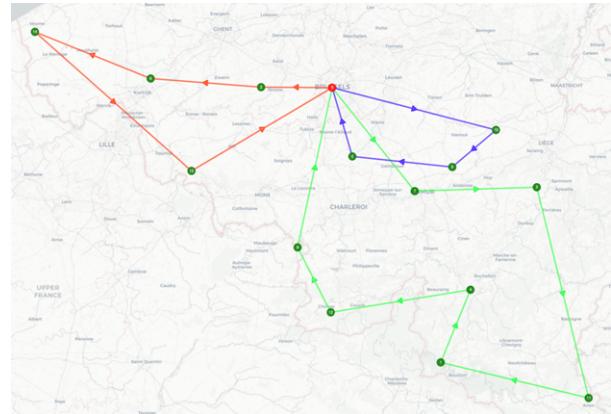


Figure 3: Optimal Solution

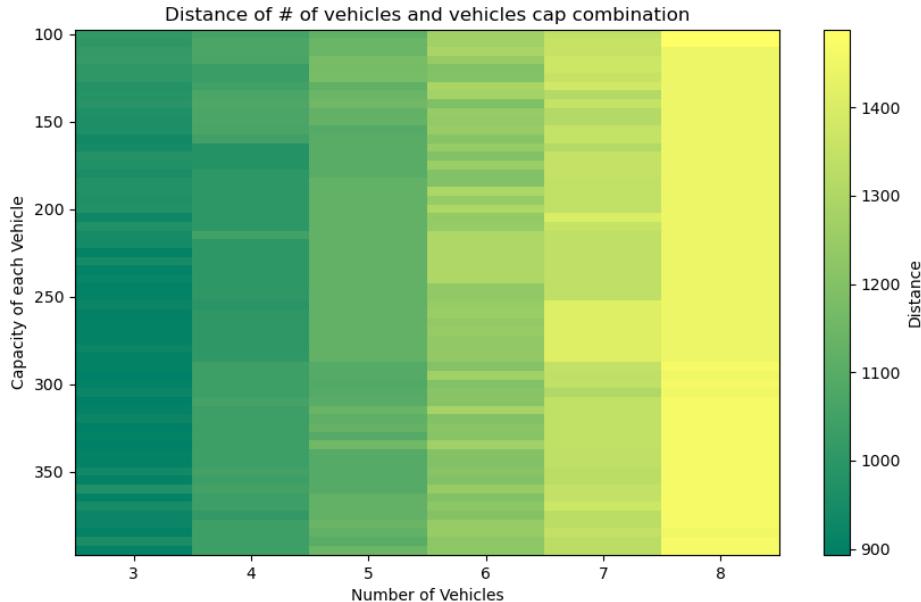
Clearly, we can see that the optimization gives us a better solution, with a lower optimal cost. Therefore, we went ahead with our optimization algorithm which was developed using Python and Gurobi.

## 2.5 Sensitivity Analysis

The true reward gained from solving such CVRPs lies in appropriating the model to real-life scenarios as possible. Therefore, prior using the language wrapper, it is imperative to do a sensitive analysis of the parameters. The benefits of doing sensitivity analysis include but are not limited to:

1. **Robustness Assessment:** It helps assess the robustness of the optimization solution to variations in input data. Real-world data can be uncertain or subject to fluctuations, and doing sensitivity analysis helps us understand how these uncertainties impact the solution's reliability.
2. **Feasibility Checking:** It helps identify potential infeasibilities in the problem instance. Sensitivity analysis can reveal situations where certain constraints or limits are violated, prompting the need to adjust the model or constraints to ensure feasibility.
3. **Decision Support:** Sensitivity analysis results can be used to provide decision-makers with a range of potential solutions and trade-offs. This enables informed decision-making by considering the impact of different options.
4. **Parameter Tuning:** Sensitivity analysis can assist in fine-tuning model parameters, such as vehicle capacity, number of vehicles, etc. Optimizing these parameters can lead to better solutions and more practical results.
5. **Scenario Testing:** Different scenarios can be analyzed to see how changes in an input parameter (e.g., the number of vehicles and their capacity) affect the optimal routes and schedules. This can help decision-makers make informed choices based on different potential scenarios.

This analysis therefore provides us with an initial idea of the hyperparameters for this model, which we can tune while simultaneously maintaining the feasibility of the model.



Some observations that we obtain from the above graph are:

- One might think that intuitively, increasing number of vehicles would increase efficiency and therefore decrease overall cost. However, sensitivity analysis shows us that a **high number of vehicles (e.g., 8) is in fact associated with a higher cost.**
- Additionally, we see that while the number of vehicles are fixed, varying the vehicle capacity does not cause any significant effect.

Therefore, we conclude that an optimal strategy in reducing costs is to simultaneously decrease the number of vehicles in the fleet and increase the capacity of each vehicle. This observation serves as a bedrock of the questions that we ask OptiGuide in the following section.

### 3 Analysis with the Language Wrapper

#### 3.1 Quantitative Questions

1. I'm working on a Capacitated Vehicle Routing Problem (CVRP) where the demand for node 5 is doubled. How can I modify the optimization model to account for this change and analyze its impact on the optimal solution?

```
# Double the demand at location 5
demands[5] = demands[5] * 2

# Adjust the constraints with respect to the new demand at
# location 5
model.addConstr(u[5] >= demands[5])
model.addConstr(u[5] <= CAPACITY)
```

**Response from OptiGuide:**

After doubling the demand at location 5 and adjusting the constraints accordingly, the optimization model was able to find an optimal solution. The total distance of all routes in this new solution is 1009.005 (approx.), which is the same as the original result. This means that even though demand at location 5 was increased, this change did not affect the optimal routing solution under the current constraints and vehicle capacity - likely because we are able to meet this additional demand within the existing capacity and routing construct. It might be necessary to review other factors such as increase in cost or time due to this additional demand.

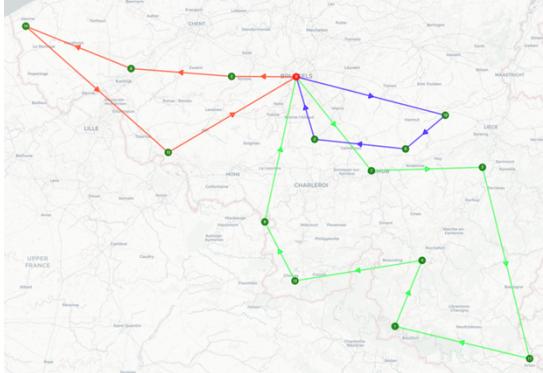


Figure 4: Original Optimal Route

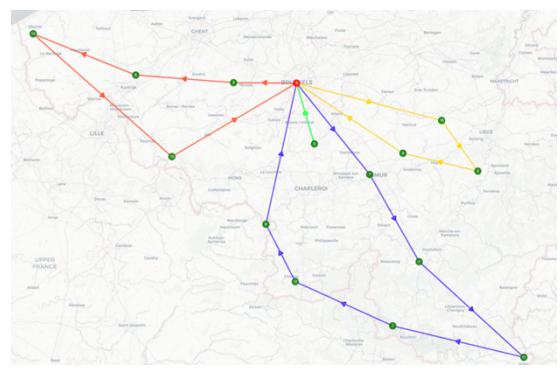


Figure 5: New Optimal Route

2. What if the demand at location 11 is increased by 50, what will the optimal solution be?

```
# Update demand at location 11
demands[11] += 50
# Update constraint 2 in the model
for i, j in arcs:
    if i != 0 and j != 0:
        model.remove(model.getConstrByName(f'c0_{i}_{j}'))
        model.addConstr((x[i, j] == 1) >>
                        (u[i] + demands[j] == u[j]))

# Update constraint 3 in the model
model.remove(model.getConstrByName('c1_11'))
model.addConstr(u[11] >= demands[11])

# Note that if the updated demand at location 11 is higher than
# the vehicle capacity
# (i.e., demands[11] > CAPACITY), the problem will not have a
# feasible solution.
```

### Response from OptiGuide:

The total minimum distance required to deliver goods has increased to 1154.77 units after increasing the demand at location 11 by 50 units. This is an increase from the previous minimum distance of 1009.005 units when the demand at location 11 was lower. This increase in the minimum distance indicates that the vehicles now need to travel further to meet the increased demand at location 11. Please remember, if demand at a location is increased such that it exceeds vehicle's capacity, then either the vehicle's capacity or the number of vehicles must be increased to meet the demand and have a feasible solution.

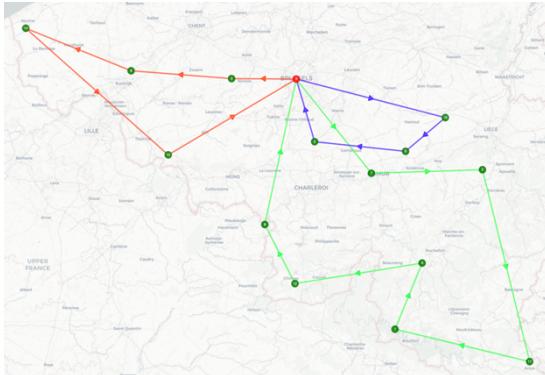


Figure 6: Original Optimal Route

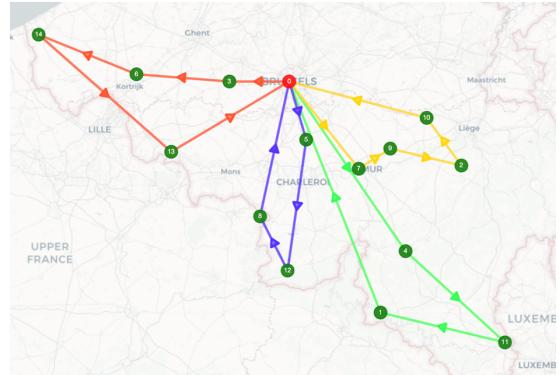


Figure 7: New Optimal Route

### 3. What if all demands are increased by 20%, what will the optimal solution be?

```
# Increase all demands by 20%
for i in demands:
    demands[ i ] *= 1.2

# Update constraints
for i in num_locations:
    model.addConstr(u[ i ] >= demands[ i ], name='demand'+str( i ))

# Optimize the model
model.optimize()
```

### Response from Optiguide:

Certainly. After increasing all demands by 20%, the optimization problem is solved again. The new objective value, which is the total distance of all routes, becomes 1033.102. Compared to the original total distance (1009.005), the new total distance is now longer due to the increase in all demands. Therefore, the optimal solution has changed with the increase in demands, yielding a longer total route distance.

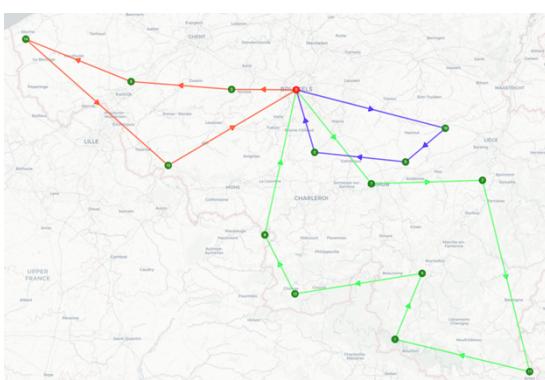


Figure 8: Original Optimal Route

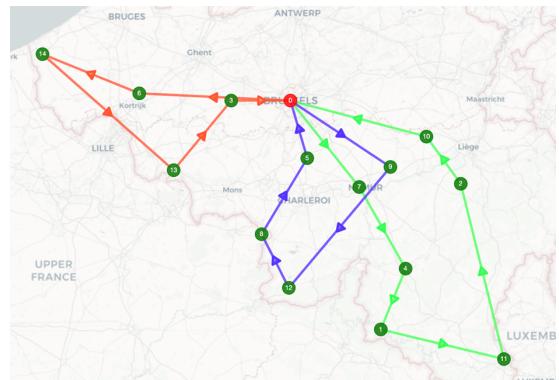


Figure 9: New Optimal Route

4. If I want to double the capacity of all vehicles, where should I make the adjustment in the CVRP code?

CAPACITY = 200

**Response from OptiGuide:**

Sure, by doubling the capacity of all vehicles from 100 to 200, we are allowing each vehicle to service more destinations before returning to the depot. As a result, the total distance of all routes is reduced from 1009.005 to 863.213. These changes reduce the overall travel cost, which makes the supply chain more efficient.

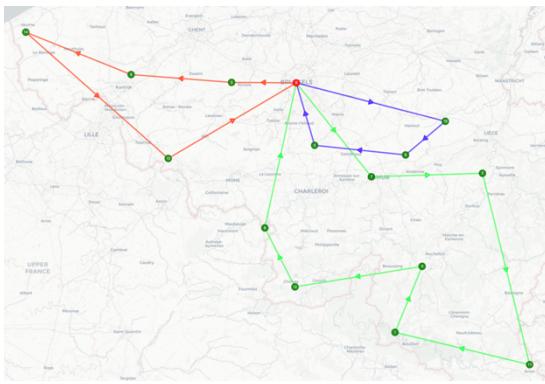


Figure 10: Original Optimal Route

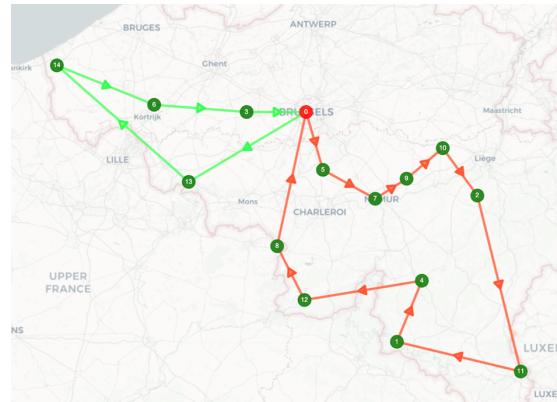


Figure 11: New Optimal Route

5. If each vehicle has a different capacity, say [50, 100, 150], how should we modify the model to account for variable capacities?

```

# First, we must replace the uniform capacity variable
# (CAPACITY) with a dictionary that provides the capacity for
# each vehicle.

# Assume vehicle_capacity is that given dictionary with
# {1:50, 2:100, 3:150}
vehicle_capacity = {1:50, 2:100, 3:150}

# We must also replace the variable u which was tracking capacity
# for a single vehicle with a matrix
# U[i,j] where i is the customer location and j is the
# vehicle's ID.

# Replace the line
# u = model.addVars(num_locations, vtype=GRB.CONTINUOUS)
# with the following
U = model.addVars(num_locations, range(num_vehicles))
, vtype=GRB.CONTINUOUS)

# Similarly, we must redefine the capacity constraints to account
# for the different vehicles.

# Replace the line
# model.addConstrs(u[i] >= demands[i] for i in num_locations)
# model.addConstrs(u[i] <= CAPACITY for i in num_locations)
# with the following
model.addConstrs(U[i,j] >= demands[i] for i in num_locations
for j in range(num_vehicles))
model.addConstrs(U[i,j] <= vehicle_capacity[j+1]
for i in num_locations for j in range(num_vehicles))

```

```

# Finally , we must redefine the flow conservation constraints
# to account for the different vehicles.

# Replace the line
# model.addConstrs((x[i, j] == 1) >> (u[i] + demands[j] == u[j]))
#           for i, j in arcs if i != 0 and j != 0)
# with the following
model.addConstrs((x[i, j] == 1) >> (U[i,k] + demands[j] == U[j,k])
                  for k in range(num_vehicles) for i, j in arcs
                  if i != 0 and j != 0)

```

#### Response from OptiGuide:

Absolutely, let's clarify the changes we made and their impact on the results. Initially, we had a model that assumed all vehicles had the same capacity. To adapt the model for vehicles with different capacities, we replaced the uniform capacity variable with a dictionary assigning different capacities to each vehicle. Furthermore, instead of a single usage variable 'u' for a vehicle, we now have a matrix 'U' that keeps track of the load of each vehicle at each customer location. When we solved the optimization with the updated constraints, we obtained an overall objective value (total distance of all routes) of 1500.821. This is higher than the original objective value of 1009.004. The increase in total distance is logical because of the differing vehicle capacities. The larger vehicles can deliver to more locations before returning to the depot, reducing overall travel. In contrast, vehicles with smaller capacities might need to make more frequent trips. Depending on the distribution of demand and vehicle capacities, the increase in total distance could be significant.

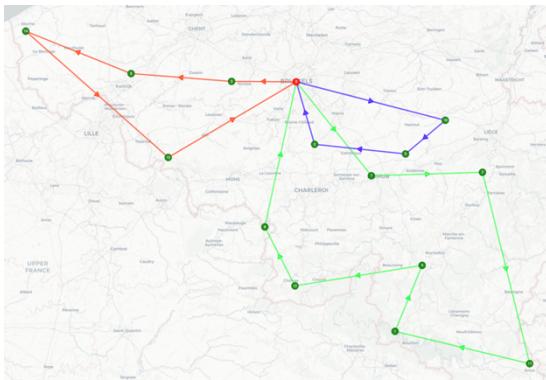


Figure 12: Original Optimal Route

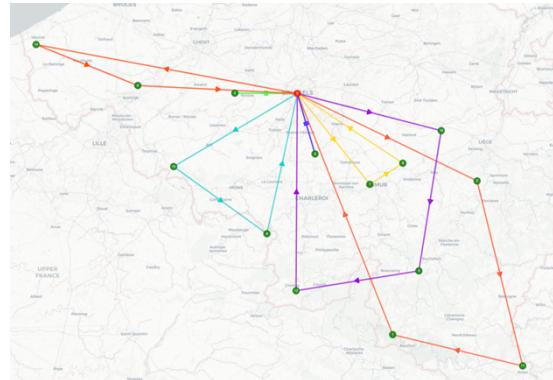


Figure 13: New Optimal Route

6. *What if the capacity of vehicles is decreased to 70, and we will need 5 vehicles now, what will the optimal solution be?*

```

# Update the vehicle capacity and number of vehicles
CAPACITY = 70
num_vehicles = 5

# Update constraints with new capacity and number of vehicles
model.addConstrs(u[i] <= CAPACITY for i in num_locations)
model.addConstr(quicksum(x[0, j] for j in num_locations)
               == num_vehicles)

```

#### Response from OptiGuide:

After reducing the capacity of each vehicle to 70 and increasing the number of vehicles to 5, the new optimal solution for the total distance of all routes has increased to 1225.568. When comparing this new solution to the original solution, which had a total distance of 1009.005 with a larger vehicle capacity of 100 and only 2 vehicles, we see that decreasing vehicle capacity while increasing the number of vehicles actually resulted in an increase in

the total distance traveled. This suggests that having fewer vehicles with larger capacity is more optimal in terms of total distance traveled for this specific supply chain problem.

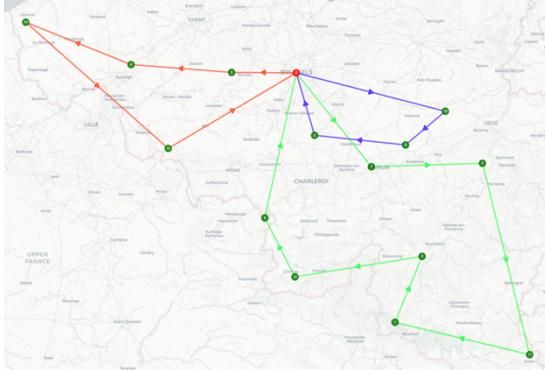


Figure 14: Original Optimal Route

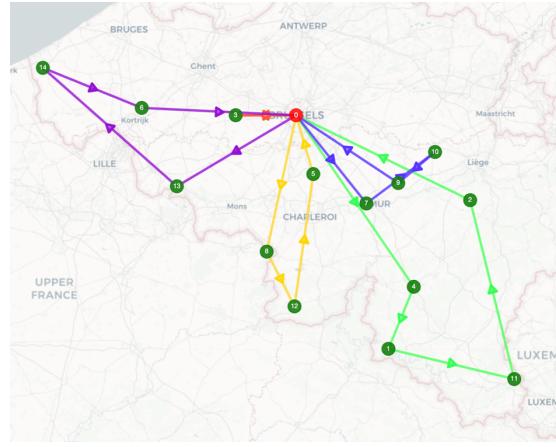


Figure 15: New Optimal Route

### 7. If we remove node 13 and node 4, what will the optimal solution value be?

```
# Update variables for the modified problem
num_locations = [1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14]
V = [0] + num_locations
demands = {1: 15, 2: 16, 3: 31, 6: 7, 7: 9, 8: 17, 9: 33, 10: 26,
           11: 18, 12: 10, 14: 26}
```

#### Response from OptiGuide:

After removing nodes 13 and 4, the new optimal solution value for the modified problem is 942.581. This is a decrease in the objective value compared to the original problem, where the objective value was 1009.005. Removing these nodes led to a more efficient solution, reducing the total distance traveled by the vehicles.

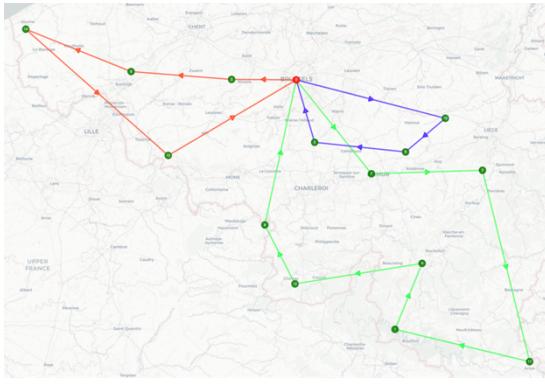


Figure 16: Original Optimal Route

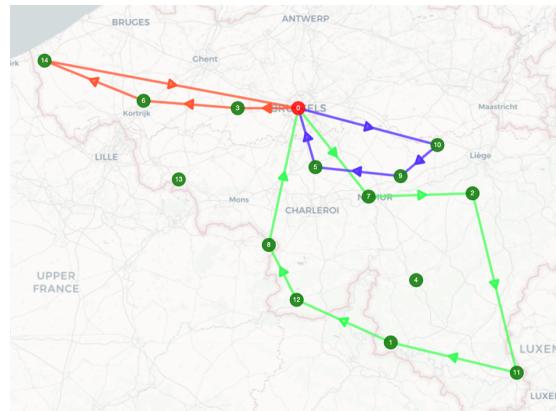


Figure 17: New Optimal Route

## 3.2 Exploratory Questions

### 1. Is it possible to integrate real-time traffic data into the CVRP model to optimize routes based on current traffic conditions? Explain the approach.

```
# To resolve the NameError , you need to define the function
fetch_traffic_data() that fetches
# real-time traffic data from a traffic API or database.
```

```
# Here's an updated code snippet:
```

```
# Function to fetch real-time traffic data
```

```

def fetch_traffic_data():
    # This function should be implemented to fetch the current
    # traffic data
    # from a traffic API or database and return the data as a
    # dictionary or matrix
    # For example:
    """
    traffic_data = {
        (0, 0): 0.0,
        (0, 1): 0.8,
        (0, 2): 1.2,
        ...
    }
    """
    # Implement the logic to fetch and process the data
    # Return the traffic data
    return traffic_data

# Function to update the distance matrix based on real-time
# traffic data
def update_distance_matrix(distance_matrix):
    # Fetch real-time traffic data
    traffic_data = fetch_traffic_data()

```

**Response from OptiGuide:**

Although OptiGuide failed to provide a text answer for this question, we were pleasantly surprised at its ability to provide us with the source code for integrating real-time traffic data.

2. *How can we modify the objective function to prioritize minimizing the number of vehicles used rather than the total distance traveled?*

```

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

# ADD NEW CODE HERE
# Determine the highest possible total distance based on the
# distance matrix
max_distance = sum(distance_matrix[i, j]
for i, j in arcs if i != j)

# Change the objective function to prioritize minimizing the
# number of vehicles
# A very large constant (max_distance) is multiplied with the
# number of vehicles used
# to make the number of vehicles the primary objective to minimize
model.setObjective(quicksum(x[i, j] * distance_matrix[i, j]
for i, j in arcs) + max_distance * quicksum(x[0, j]
for j in num_locations))

```

**Response from OptiGuide:**

The code was modified to prioritize minimizing the number of vehicles used rather than the total distance traveled. A large constant derived from the maximum possible distance was multiplied with the number of vehicles used.

Before this modification, the model was optimized based only on the total distance, giving an optimal total distance of roughly 1009.005.

After the modification, the objective value was also found to be 1009.005. This means that the solution found by the model did not change. This could be due to the fact that the original solution already employed the minimum number of vehicles possible, or that the distance and vehicle numbers trade-off did not require additional vehicles to significantly reduce the total distance.

Remember, while the objective value did not change, the priorities of the model did. The model now seeks to minimize the number of vehicles even if that might slightly increase the total distance traveled. In other words, the model will now prefer solutions that use fewer vehicles, even if they are slightly longer in terms of distance.

3. *Is it possible to group certain customers and ensure they are serviced by the same vehicle? How can we represent this in the model?*

```
# First create a list of customers that should be grouped
# together
# For example: group = [1, 3, 5], which means customers 1, 3,
# and 5 should be serviced by the same vehicle

group = [1, 3, 5]

# Add a new constraint to the model that ensures all customers
# in the group are serviced by the same vehicle
# The sum of the arcs coming from the depot to the group
# members should be 1
# This means that there is only one vehicle serving this group

model.addConstr(quicksum(x[0, i] for i in group) == 1,
"group_served_by_same_vehicle")
```

#### **Response from OptiGuide:**

After incorporating a new constraint to our model - which ensures that specific customers (in this case, customers 1, 3, and 5) are serviced by the same vehicle - the model is still able to find an optimal solution.

The optimization problem is still solved successfully and the total distance of all routes remains the same (1009.005). This indicates that in the original optimal configuration, these customers were already being serviced by the same vehicle. If they hadn't been, the total distance would likely have increased due to the additional constraint restricting the possible solutions.

In conclusion, the inclusion of such constraints allows us to adapt the model to specific business requirements without sacrificing the overall efficiency of the solution.

## 4 Appendix

### 4.1 Transcription

Link to the conversation with ChatGPT

## References

- [1] Li, Beibin and Mellou, Konstantina and Zhang, Bo and Pathuri, Jeevan and Menache, Ishai. *Large Language Models for Supply Chain Optimization*. arXiv preprint arXiv:2307.03875, 2023.
- [2] De Smet, *Vehicle routing with real road distances*. OptaPlanner Blog. <https://www.optaplanner.org/blog/2014/09/02/VehicleRoutingWithRealRoadDistances.html>. Accessed on November 12, 2023.
- [3] ge0ffrey. *vRP-dataset-generator*. <https://github.com/ge0ffrey/vRP-dataset-generator/tree/master>.
- [4] Gupta, Ashima and Saini, Sanjay. *An Enhanced Ant Colony Optimization Algorithm for Vehicle Routing Problem with Time Windows*. 267-274. 10.1109/ICoAC.2017.8441175, 2017.