Optimization Methods Project (ORIE 5380 / CS 5727)

Alex Bean (atb95) & Arjun Hegde (ah2362)

December 2, 2024

Question 1

Decision Variables

- $x_{ij} \in \{0, 1\}$: A binary variable where $x_{ij} = 1$ if a driver delivers from warehouse i to customer j, and $x_{ij} = 0$ otherwise.

Parameters

- M: Set of warehouses, indexed by i.
- N: Set of customers, indexed by j.
- d_{ij}: Euclidean distance between warehouse i and customer j. This is represented as the variable distance in the code.
- s_i: Supply capacity of warehouse i.
- k: Total number of drivers available.

Objective Function

$$\text{Minimize } Z = \sum_{i \in M} \sum_{j \in N} d_{ij} \cdot x_{ij}$$

Constraints

1. **Customer Delivery Constraint:** Each customer can receive at most one delivery:

$$\sum_{i \in M} x_{ij} \leq 1, \quad \forall j \in N$$

2. **Driver Capacity Constraint:** The total deliveries cannot exceed the number of drivers or customers, whichever is smaller:

$$\sum_{i \in M} \sum_{j \in N} x_{ij} = \min(k, |N|)$$

3. **Warehouse Supply Constraint:** The total deliveries from a warehouse cannot exceed its supply:

$$\sum_{j \in N} x_{ij} \leq s_i, \quad \forall i \in M$$

4. **Binary Decision Variable Constraint:** The decision variable is binary:

$$x_{ij} \in \{0, 1\}, \quad \forall i \in M, j \in N$$

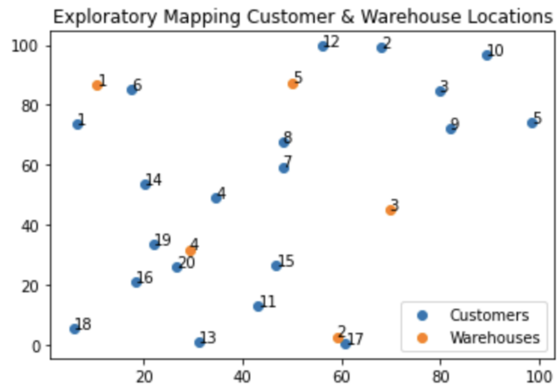


Figure 1: Optimal 1:1 Route Between Warehouse and Customer

Optimal Delivery Route

The solution to the problem, as derived using Gurobi, is as follows:

$$\begin{aligned}
&\text{Driver 1: Warehouse } W01 \rightarrow \text{Customer } C01 \text{ (Distance: 13.8 units)}\\
&\text{Driver 2: Warehouse } W01 \rightarrow \text{Customer } C06 \text{ (Distance: 7.1 units)}\\
&\text{Driver 3: Warehouse } W02 \rightarrow \text{Customer } C17 \text{ (Distance: 2.8 units)}\\
&\text{Driver 4: Warehouse } W04 \rightarrow \text{Customer } C19 \text{ (Distance: 7.4 units)}\\
&\text{Driver 5: Warehouse } W04 \rightarrow \text{Customer } C20 \text{ (Distance: 6.2 units)}\\
&\text{Driver 6: Warehouse } W05 \rightarrow \text{Customer } C12 \text{ (Distance: 13.6 units)}
\end{aligned}$$

Total Distance

Total Distance: 50.91 units.

Question 2

Objective Function

The objective function minimizes the total distance traveled by all items transported from warehouses to customers:

$$\min \sum_{i \in M \cup N, j \in N} y_{ij} d_{ij}$$

where:

- y_{ij} is the flow of items from node i to customer j.
- d_{ij} is the Euclidean distance between nodes i and j.

Constraints

The problem is subject to the following constraints:

1. Warehouse supply constraint:

$$\sum_{j \in N} y_{ij} \leq s_i, \quad \forall i \in M$$

where s_i is the supply available at warehouse i.

2. Flow conservation constraint for customers:
$$\sum_{i \in M \cup N} y_{ij} = \sum_{i \in N} y_{ji} + 1, \quad \forall j \in N$$
This ensures that one unit of demand is fulfilled for each customer.
3. Non-negativity constraint:
$$y_{ij} \geq 0, \quad \forall i \in M \cup N, j \in N$$

Variables and Parameters

The key variables and parameters used in the problem are:

- y_{ij}: Decision variable representing the number of items transported from node i to customer j.
- d_{ij}: Euclidean distance between nodes i and j. In the code it is represented simply as the variable distance.
- s_i: Supply at warehouse i.
- M: Set of warehouses.
- N: Set of customers.

Optimal Delivery Route

Using Python and the Gurobi optimization library, the following optimal delivery routes were determined:

- Flow from warehouse W01 to customer C01: 1.00 with a distance of 13.83 units.
- Flow from warehouse W01 to customer C06: 1.00 with a distance of 7.07 units.
- Flow from warehouse W01 to customer C14: 1.00 with a distance of 34.68 units.
- Flow from warehouse W02 to customer C11: 1.00 with a distance of 18.82 units.
- Flow from warehouse W02 to customer C13: 1.00 with a distance of 27.93 units.
- Flow from warehouse W02 to customer C15: 1.00 with a distance of 26.69 units.
- Flow from warehouse W02 to customer C17: 1.00 with a distance of 2.82 units.
- Flow from warehouse W03 to customer C05: 1.00 with a distance of 40.79 units.
- Flow from warehouse W03 to customer C07: 1.00 with a distance of 25.93 units.
- Flow from warehouse W03 to customer C09: 1.00 with a distance of 29.82 units.
- Flow from warehouse W04 to customer C04: 1.00 with a distance of 18.07 units.
- Flow from warehouse W04 to customer C16: 1.00 with a distance of 15.30 units.
- Flow from warehouse W04 to customer C18: 1.00 with a distance of 35.17 units.
- Flow from warehouse W04 to customer C19: 1.00 with a distance of 7.36 units.
- Flow from warehouse W04 to customer C20: 1.00 with a distance of 6.19 units.
- Flow from warehouse W05 to customer C02: 1.00 with a distance of 21.44 units.
- Flow from warehouse W05 to customer C03: 1.00 with a distance of 29.66 units.
- Flow from warehouse W05 to customer C08: 1.00 with a distance of 19.84 units.
- Flow from warehouse W05 to customer C10: 1.00 with a distance of 40.11 units.
- Flow from warehouse W05 to customer C12: 1.00 with a distance of 13.63 units.

With that, the total distance of the optimal flow is 435.18 units.

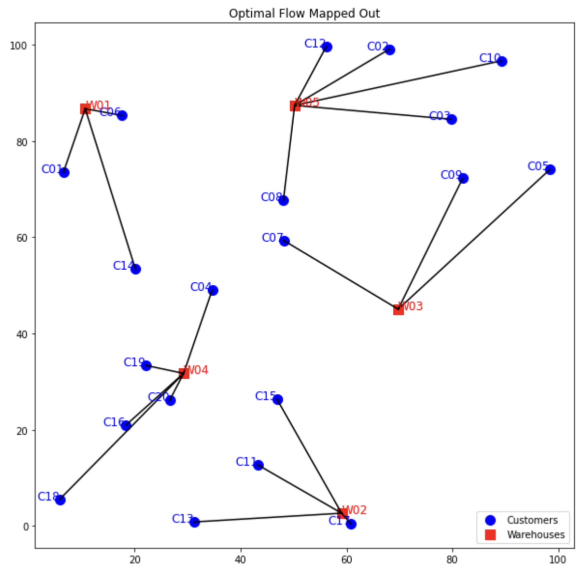


Figure 2: Optimal Route from warehouse to numerous customers

Understanding Our Linear Programming Solution

The optimal delivery schedule generated by solving the linear program assigns flows from warehouses to customers such that the total distance traveled is minimized. Under the assumption of an unlimited number of drivers, the solution achieves this goal by:

- Assigning flows directly from warehouses to customers without requiring a single driver to serve multiple customers.
- Minimizing the Euclidean distance traveled for each unit of flow, effectively reducing the overall distance.
- Ensuring that the demand of each customer is met exactly, while not exceeding the supply capacity of any warehouse.

This approach results in the schedule that minimizes the total distance traveled, as it leverages the unlimited availability of drivers to eliminate the need for combining deliveries into multi-stop routes. Each driver is responsible for exactly one delivery, avoiding additional travel that would arise from combining customer deliveries into a single route.

Does this schedule minimize the total distance? Yes, the LP solution minimizes the total distance traveled under the given assumption of unlimited drivers. This is because the model directly optimizes the objective function, which is the total distance weighted by the flow, without being constrained by the need to group deliveries into fewer routes.

Limitations of the model: While the LP provides the optimal solution under the assumption of unlimited drivers, it may not be realistic for scenarios where:

- There is a limited number of drivers, which would require grouping deliveries into routes.
- Other operational constraints exist, such as time windows or driver working hours.
- Real-world factors, such as road networks, traffic conditions, or delivery priorities, are present.

In conclusion, the LP schedule is optimal for minimizing total distance in the specific case of unlimited drivers. However, practical applications may require modifying the model to account for additional real-world constraints.

Question 3

Decision Variables

- $z_{ij} \in \{0, 1\}$: A binary variable indicating whether a route exists from node i to node j.
- $y_{ij} \in \mathbb{Z}_{\geq 0}$: An integer variable representing the number of deliveries on the route from node i to node j.

Parameters

- Q: The maximum number of deliveries a driver can handle.
- d_{ij}: The Euclidean distance between node i and node j. his is represented as the variable distance in the code.
- k: The number of available drivers, here assumed to be 6.
- supply$[i]$: The supply available at warehouse i.
- demand$[j]$: The demand at customer j.

Objective Function

Minimize the total distance traveled:

$$\text{Minimize} \quad \sum_{i,j} d_{ij} \cdot z_{ij}$$

Constraints

1. Each driver starts at a warehouse and does not return to any warehouse during their delivery path:
$$\sum_{j \in \text{customers}} z_{ij} \leq 1, \quad \forall i \in \text{warehouses}$$
2. Each customer is visited by exactly one driver:
$$\sum_{i \in \text{nodes}, i \neq j} z_{ij} = 1, \quad \forall j \in \text{customers}$$
3. Flow conservation for customers:
$$\sum_{i \in \text{nodes}, i \neq j} y_{ij} - \sum_{k \in \text{customers}, k \neq j} y_{jk} = 1, \quad \forall j \in \text{customers}$$
4. Each route between nodes is only active if there is flow along it:
$$z_{ij} \leq y_{ij}, \quad y_{ij} \leq Q \cdot z_{ij}, \quad \forall i, j \in \text{nodes}, i \neq j$$
5. The total number of routes starting from warehouses is limited by the number of drivers:
$$\sum_{i \in \text{warehouses}} \sum_{j \in \text{customers}} z_{ij} \leq k$$
6. The supply at each warehouse cannot be exceeded:
$$\sum_{j \in \text{customers}} y_{ij} \leq \text{supply}[i], \quad \forall i \in \text{warehouses}$$

Optimal Delivery Schedule

After solving the integer program with $k = 6$ using Gurobi, the optimal delivery schedule is as follows:

- **Driver 1:** Warehouse W01 → Customer C06 → Customer C01 Total distance: 23.22 units
- **Driver 2:** Warehouse W02 → Customer C17 → Customer C15 → Customer C11 → Customer C13 Total distance: 63.31 units
- **Driver 3:** Warehouse W03 → Customer C09 → Customer C03 → Customer C10 → Customer C05 Total distance: 82.11 units
- **Driver 4:** Warehouse W04 → Customer C19 → Customer C20 → Customer C16 → Customer C18 Total distance: 45.56 units
- **Driver 5:** Warehouse W05 → Customer C08 → Customer C07 → Customer C04 → Customer C14 Total distance: 60.40 units
- **Driver 6:** Warehouse W05 → Customer C12 → Customer C02 Total distance: 25.61 units

Total distance traveled: 300.22 units.

Question 4

The total distance traveled by the delivery drivers for k values ranging from 4 to 10 to calculate the total distance:

```
different = {}

for k in range(4, 11):
    different[k] = optimal_k(warehouses, customers, supply, distance, k)

different
```

The results for the total distance traveled were as follows:

$$\{4 : 372.49, 5 : 316.20, 6 : 300.21, 7 : 296.56, 8 : 293.87, 9 : 291.50, 10 : 289.19\}$$

The following Python code was used to generate the trend plot:

```
k_values = list(different.keys())
total_distances = list(different.values())

plt.plot(k_values, total_distances, marker='o')
plt.xlabel('Number of Drivers (k)')
plt.ylabel('Total Distance Traveled')
plt.title('Trend of Total Distance Traveled')
plt.grid(True)
plt.show()
```

Resulting Plot

The plot shows a decreasing trend in the total distance traveled as the number of drivers k increases. This is consistent with expectations, as more drivers reduce the workload and the distance traveled per driver. However, the reduction in total distance diminishes with higher values of k, indicating a decrease in the returns of additional drivers beyond a certain point.

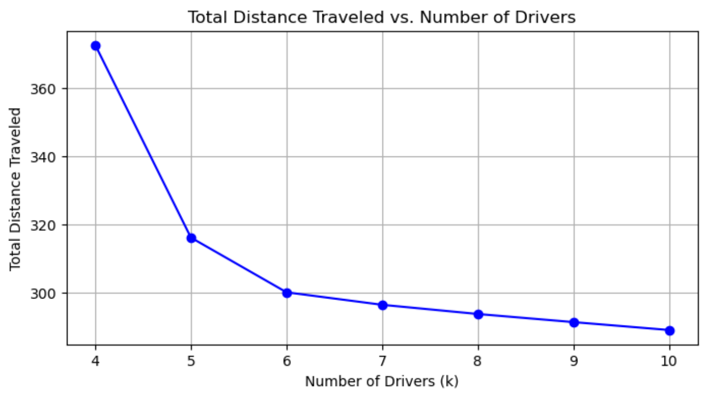
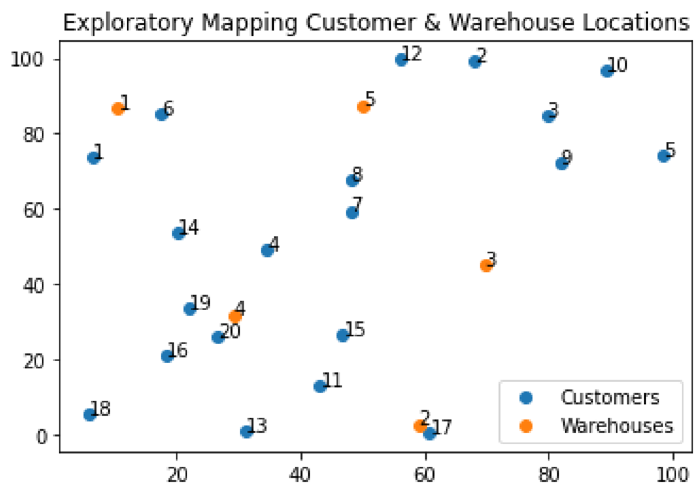


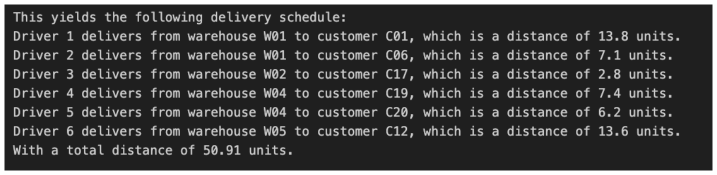
Figure 3: Trend of Total Distance Traveled as k Changes from 4 to 10

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gurobipy as gp
from gurobipy import GRB
from scipy.spatial.distance import cdist

warehouses = pd.read_csv('/Users/alexbean/Desktop/data/warehouse_locations.csv')
customers = pd.read_csv('/Users/alexbean/Desktop/data/customer_locations.csv')
supply = pd.read_csv('/Users/alexbean/Desktop/data/supply.csv')
driver = pd.read_csv('/Users/alexbean/Desktop/data/driver_locations.csv')



# Problem 1

warehouses.columns = ['Warehouse', 'X', 'Y']
customers.columns = ['Customer', 'X', 'Y']
supply.columns = ['Warehouse', 'Supply']
driver.columns = ['Driver', 'X', 'Y']

plt.scatter(customers['X'], customers['Y'], label = 'Customers')
plt.scatter(warehouses['X'], warehouses['Y'], label = 'Warehouses')
plt.title('Exploratory Mapping Customer & Warehouse Locations')
plt.legend()

# Reindexing so the labels are more intuitive
for index, row in customers.iterrows():
    plt.text(row['X'], row['Y'], int(row['Customer']+1))

for index, row in warehouses.iterrows():
    plt.text(row['X'], row['Y'], int(row['Warehouse']+1))

plt.show()

# Creating dictionaries for warehouse, customer, and supply
```

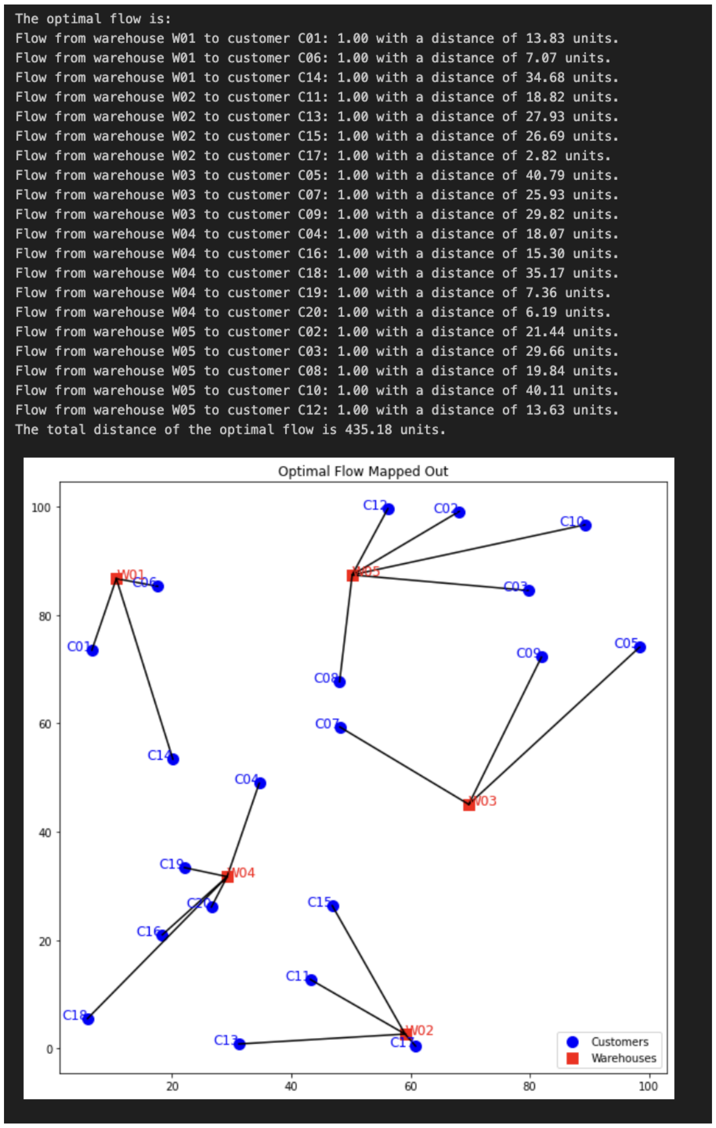
```
warehouses = {f'W{i+1:02}': {'coord': np.array([x, y])} for i, (x, y) in enumerate(zip(warehouses['X'
customers = {f'C{i+1:02}': {'coord': np.array([x, y])} for i, (x, y) in enumerate(zip(customers['X'],
supply = {f'W{i+1:02}': s for i, (s) in enumerate(supply['Supply'])}

# Euclidean distances
distance = {}
for i in warehouses:
    for j in customers:
        distance[i, j] = np.linalg.norm(warehouses[i]['coord'] - customers[j]['coord'])

# Given in instructions as assumption
k = 6

model = gp.Model("DeliveryMinimization")
model.setParam('OutputFlag', 0) # Turn off output

# Variables indicating delivery from warehouse i to customer j
x = {(i, j): model.addVar(vtype=GRB.BINARY, name=f"x_{i}_{j}")
     for i in warehouses for j in customers}

# Minimize total distance
total_distance = gp.quicksum(distance[i, j] * x[i, j] for i in warehouses for j in customers)
model.setObjective(total_distance, GRB.MINIMIZE)

# Each customer receives at most one delivery
for j in customers:
    model.addConstr(gp.quicksum(x[i, j] for i in warehouses) <= 1, name=f"Customer_{j}_OneDelivery")

# The total number of deliveries is determined by the lesser of the number of drivers or customers
total_deliveries = gp.quicksum(x[i, j] for i in warehouses for j in customers)
max_deliveries = min(k, len(customers))  # choosing between drivers or customers as smallest number
model.addConstr(total_deliveries == max_deliveries, name="TotalDeliveries_Limit")

# Warehouse supply
for i in warehouses:
    model.addConstr(gp.quicksum(x[i, j] for j in customers) <= supply[i], name=f"Warehouse_{i}_Supply

model.optimize()

if model.status == GRB.OPTIMAL:
    delivery_schedule = []
    total_distance = 0

    for (i, j), var in x.items():
        if var.X > 0.5:  # Check if the variable is selected in the optimal solution
            delivery_info = {
                'warehouse': i,
                'customer': j,
                'distance': distance[i, j],
                'start_coords': warehouses[i]['coord'],
                'end_coords': customers[j]['coord']
            }
            delivery_schedule.append(delivery_info)
            total_distance += distance[i, j]

    print("This yields the following delivery schedule:")
    for idx, delivery in enumerate(delivery_schedule, start=1):
        print(f"Driver {idx} delivers from warehouse {delivery['warehouse']} "
              f"to customer {delivery['customer']}, which is a distance of {delivery['distance']:.1f}
    print(f"With a total distance of {total_distance:.2f} units.")
```




```
This yields the following delivery schedule:
Driver 1 delivers from warehouse W01 to customer C01, which is a distance of 13.8 units.
Driver 2 delivers from warehouse W01 to customer C06, which is a distance of 7.1 units.
Driver 3 delivers from warehouse W02 to customer C17, which is a distance of 2.8 units.
Driver 4 delivers from warehouse W04 to customer C19, which is a distance of 7.4 units.
Driver 5 delivers from warehouse W04 to customer C20, which is a distance of 6.2 units.
Driver 6 delivers from warehouse W05 to customer C12, which is a distance of 13.6 units.
With a total distance of 50.91 units.
```

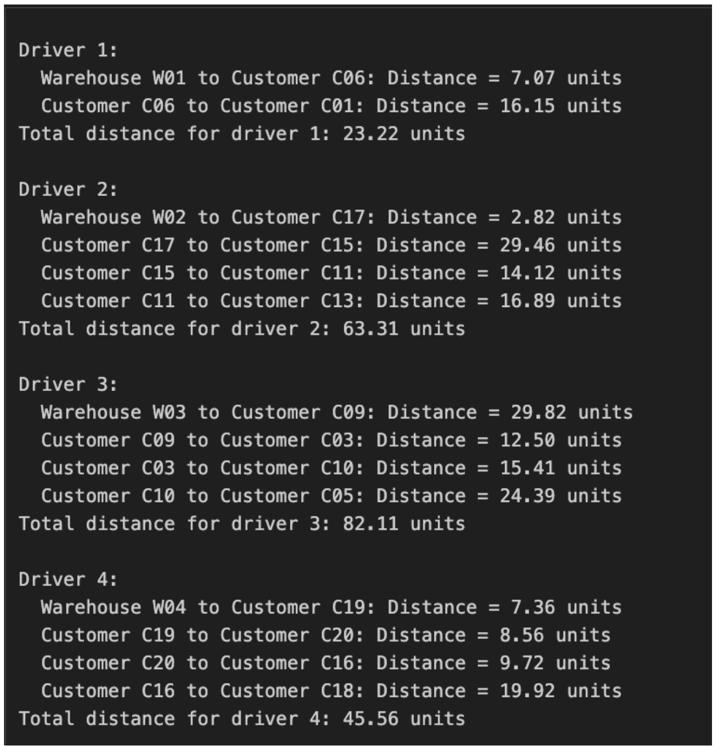
```
# Problem 2

nodes = {**warehouses, **customers}

# Euclidean distances between nodes
distance = {
    (i, j): np.linalg.norm(nodes[i]["coord"] - nodes[j]["coord"])
    for i in nodes for j in customers if i in warehouses or i in customers
}

model = gp.Model("MinimumCostFlow")
model.setParam("OutputFlag", 0) # Turn off output

# y[i, j] is the flow from node i to customer j
y = model.addVars(nodes.keys(), customers.keys(), name = "y", lb = 0, vtype = GRB.INTEGER)

# Minimize the total distance weighted by the flow
model.setObjective(
    gp.quicksum(distance[i, j] * y[i, j] for i, j in distance.keys()), GRB.MINIMIZE
)

# Warehouse constraints
for i in warehouses:
    model.addConstr(gp.quicksum(y[i, j] for j in customers) <= supply[i], name = f"Supply_{i}")
```

```
# Flow conservation for customer
for j in customers:
    inf = gp.quicksum(y[i, j] for i in nodes if i != j)
    out = gp.quicksum(y[j, k] for k in customers if k != j)
    model.addConstr(inf - out == 1, name = f"Flow_{j}")

model.optimize()

print("The optimal flow is:")

if model.status == GRB.OPTIMAL:
    flows = {
        (i, j): y[i, j].X
        for i in nodes for j in customers if y[i, j].X > 1e-6 # we picked a threshold of 1e-6 arbitra
    }

    for (i, j), flow_value in flows.items():
        if i in warehouses:
            print(f"Flow from warehouse {i} to customer {j}: {flow_value:.2f} with a distance of {dis
        else:
            print(f"Flow from node {i} to customer {j}: {flow_value:.2f}")

    print(f"The total distance of the optimal flow is {model.objVal:.2f} units.")

    plt.figure(figsize=(10, 10))
    plt.scatter(
        [node["coord"][0] for node in customers.values()],
        [node["coord"][1] for node in customers.values()],
        label="Customers",
        color="blue",
        marker='o',
        s=100
    )
    plt.scatter(
        [node["coord"][0] for node in warehouses.values()],
        [node["coord"][1] for node in warehouses.values()],
        label="Warehouses",
        color="red",
        marker='s',
        s=100
    )

    for customer_id, customer_data in customers.items():
        plt.text(
            customer_data["coord"][0],
            customer_data["coord"][1],
            f"{customer_id}",
            fontsize=12,
            ha='right',
            color="blue"
        )

    for warehouse_id, warehouse_data in warehouses.items():
        plt.text(
            warehouse_data["coord"][0],
            warehouse_data["coord"][1],
            f"{warehouse_id}",
            fontsize=12,
            ha='left',
            color="red"
        )

    plt.title('Customer and Warehouse Locations')

    for (i, j), flow_value in flows.items():
        if flow_value > 1e-6:
            plt.plot(
                [nodes[i]["coord"][0], nodes[j]["coord"][0]],
                [nodes[i]["coord"][1], nodes[j]["coord"][1]],
                color="black"
            )

    plt.title("Optimal Flow Mapped Out")
    plt.legend(loc = "lower right")
    plt.show()
```

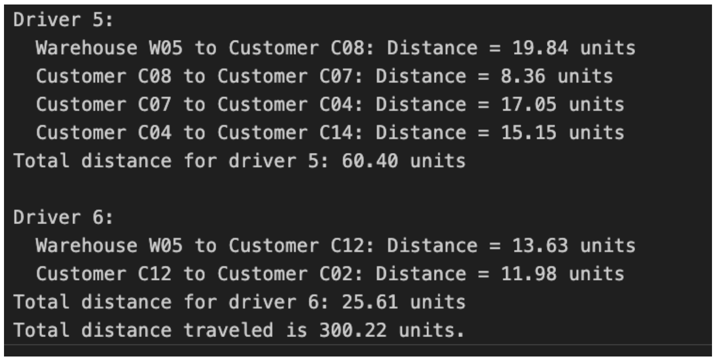


```
The optimal flow is:
Flow from warehouse W01 to customer C01: 1.00 with a distance of 13.83 units.
Flow from warehouse W01 to customer C06: 1.00 with a distance of 7.07 units.
Flow from warehouse W01 to customer C14: 1.00 with a distance of 34.68 units.
Flow from warehouse W02 to customer C11: 1.00 with a distance of 18.82 units.
Flow from warehouse W02 to customer C13: 1.00 with a distance of 27.93 units.
Flow from warehouse W02 to customer C15: 1.00 with a distance of 26.69 units.
Flow from warehouse W02 to customer C17: 1.00 with a distance of 2.82 units.
Flow from warehouse W03 to customer C05: 1.00 with a distance of 40.79 units.
Flow from warehouse W03 to customer C07: 1.00 with a distance of 25.93 units.
Flow from warehouse W03 to customer C09: 1.00 with a distance of 29.82 units.
Flow from warehouse W04 to customer C04: 1.00 with a distance of 18.07 units.
Flow from warehouse W04 to customer C16: 1.00 with a distance of 15.30 units.
Flow from warehouse W04 to customer C18: 1.00 with a distance of 35.17 units.
Flow from warehouse W04 to customer C19: 1.00 with a distance of 7.36 units.
Flow from warehouse W04 to customer C20: 1.00 with a distance of 6.19 units.
Flow from warehouse W05 to customer C02: 1.00 with a distance of 21.44 units.
Flow from warehouse W05 to customer C03: 1.00 with a distance of 29.66 units.
Flow from warehouse W05 to customer C08: 1.00 with a distance of 19.84 units.
Flow from warehouse W05 to customer C10: 1.00 with a distance of 40.11 units.
Flow from warehouse W05 to customer C12: 1.00 with a distance of 13.63 units.
The total distance of the optimal flow is 435.18 units.
```

```
# Problem 3

nodes = {**warehouses, **customers}

distance = {}

for i in warehouses:
    for j in customers:
        distance[i, j] = np.linalg.norm(warehouses[i]['coord'] - customers[j]['coord'])

for i in customers:
    for j in customers:
        distance[i, j] = np.linalg.norm(customers[i]['coord'] - customers[j]['coord'])


model = gp.Model("Problem3")
model.setParam('OutputFlag', 0) # Turn off output

# Maximum number of deliveries per driver
Q = 5

z = model.addVars(distance.keys(), vtype = GRB.BINARY)
y = model.addVars(distance.keys(), lb=0, ub=Q,  vtype = GRB.INTEGER)

model.setObjective(gp.quicksum(distance[i, j] * z[i, j] for i, j in distance.keys()), GRB.MINIMIZE)

for i in warehouses:
    model.addConstr(gp.quicksum(y[i, j] for j in customers) <= supply[i])

for j in customers:
    inflow = gp.quicksum(y[i, j] for i in nodes if i != j)
    outflow = gp.quicksum(y[j, k] for k in customers if k != j)
    model.addConstr(inflow - outflow == 1)

for j in customers:
    model.addConstr(gp.quicksum(z[i, j] for i in nodes if i != j) == 1)

for j in customers:
    model.addConstr(gp.quicksum(z[j, i] for i in customers if i != j) <= 1)

model.addConstr(gp.quicksum(z[i, j] for i in warehouses.keys() for j in customers.keys()) <= k)

for i in nodes:
    for j in customers.keys():
        if i!=j:
            model.addConstr(z[i, j] <= y[i, j])
            model.addConstr(y[i, j] <= Q * z[i, j])

model.optimize()


def get_routes(z_vars, warehouses, customers):
    routes = []
    edges = {i: [] for i in warehouses}
    for (i, j), var in z_vars.items():
        if var.X > 0.5:
            edges.setdefault(i, []).append(j)

    visited = set()

    for start in warehouses:
        for node in edges[start]:
            if node not in visited:
                route = [start]
                current = node
                route.append(current)
                visited.add(current)
                while any(next_node for next_node in edges.get(current, []) if next_node not in visit
                    for potential_next in edges[current]:
                        if potential_next not in visited:
                            current = potential_next
                            route.append(current)
                            visited.add(current)
                            break
                routes.append(route)

    return routes


if model.status == GRB.OPTIMAL:
    delivery_routes = get_routes(z, warehouses, customers)

    for driver_id, route in enumerate(delivery_routes, start=1):
        route_distance = 0
        route_steps = []
        previous_node = None

        for idx in range(len(route) - 1):
            start_node = route[idx]
            end_node = route[idx + 1]
            if end_node == previous_node:
                continue
            distance_between_nodes = distance.get((start_node, end_node), 0)
            route_distance += distance_between_nodes
            start_label = f"{start_node} -> " if start_node in warehouses else ""
            end_label = f"{end_node}"
            route_steps.append(f"{start_label}{end_label}")
            previous_node = end_node

    for driver_id, route in enumerate(delivery_routes, start=1):
        print(f"\nDriver {driver_id}:")
        route_distance = 0
        for idx in range(len(route)-1):
            start_node = route[idx]
            end_node = route[idx+1]
            distance_between_nodes = distance.get((start_node, end_node), 0)
            route_distance += distance_between_nodes
            if start_node in warehouses:
                start_label = f"Warehouse {start_node}"
            else:
                start_label = f"Customer {start_node}"
            end_label = f"Customer {end_node}"
            print(f"  {start_label} to {end_label}: Distance = {distance_between_nodes:.2f} units")
        print(f"Total distance for driver {driver_id}: {route_distance:.2f} units")

    total_distance = sum(distance.get((start_node, end_node), 0) for route in delivery_routes for sta
    print(f"Total distance traveled is {total_distance:.2f} units.")
```



```
Driver 1:
  Warehouse W01 to Customer C06: Distance = 7.07 units
  Customer C06 to Customer C01: Distance = 16.15 units
Total distance for driver 1: 23.22 units

Driver 2:
  Warehouse W02 to Customer C17: Distance = 2.82 units
  Customer C17 to Customer C15: Distance = 29.46 units
  Customer C15 to Customer C11: Distance = 14.12 units
  Customer C11 to Customer C13: Distance = 16.89 units
Total distance for driver 2: 63.31 units

Driver 3:
  Warehouse W03 to Customer C09: Distance = 29.82 units
  Customer C09 to Customer C03: Distance = 12.50 units
  Customer C03 to Customer C10: Distance = 15.41 units
  Customer C10 to Customer C05: Distance = 24.39 units
Total distance for driver 3: 82.11 units

Driver 4:
  Warehouse W04 to Customer C19: Distance = 7.36 units
  Customer C19 to Customer C20: Distance = 8.56 units
  Customer C20 to Customer C16: Distance = 9.72 units
  Customer C16 to Customer C18: Distance = 19.92 units
Total distance for driver 4: 45.56 units
```



```
Driver 5:
  Warehouse W05 to Customer C08: Distance = 19.84 units
  Customer C08 to Customer C07: Distance = 8.36 units
  Customer C07 to Customer C04: Distance = 17.05 units
  Customer C04 to Customer C14: Distance = 15.15 units
Total distance for driver 5: 60.40 units

Driver 6:
  Warehouse W05 to Customer C12: Distance = 13.63 units
  Customer C12 to Customer C02: Distance = 11.98 units
Total distance for driver 6: 25.61 units
Total distance traveled is 300.22 units.
```

```
# Problem 4

def optimal(warehouses, customers, supply, distance, k, Q = 5):

    model = gp.Model("VehicleRoutingProblem")
    model.setParam('OutputFlag', 0)  # Suppress output

    z = model.addVars(distance.keys(), vtype = GRB.BINARY)
    y = model.addVars(distance.keys(), lb = 0, ub = Q,  vtype = GRB.INTEGER)

    model.setObjective(gp.quicksum(distance[i, j] * z[i, j] for i, j in distance.keys()), GRB.MINIMIZ

    for i in warehouses:
        model.addConstr(gp.quicksum(y[i, j] for j in customers) <= supply[i])

    for j in customers:
        inflow = gp.quicksum(y[i, j] for i in nodes if i != j)
        outflow = gp.quicksum(y[j, k] for k in customers if k != j)
        model.addConstr(inflow - outflow == 1)

    for j in customers:
        model.addConstr(gp.quicksum(z[i, j] for i in nodes if i != j) == 1)

    for j in customers:
        model.addConstr(gp.quicksum(z[j, i] for i in customers if i != j) <= 1)

    model.addConstr(gp.quicksum(z[i, j] for i in warehouses.keys() for j in customers.keys()) <= k)

    for i in nodes:
        for j in customers.keys():
            if i!=j:
                model.addConstr(z[i, j] <= y[i, j])
                model.addConstr(y[i, j] <= Q * z[i, j])

    model.optimize()
    return model.objVal

different = {}

for k in range(4, 11):
    different[k] = optimal(warehouses, customers, supply, distance, k)

k_values = list(different.keys())
total_distances = list(different.values())

plt.figure(figsize=(10, 6))
plt.plot(k_values, total_distances, marker='o', linestyle='-', color='b')
plt.title('Total Distance Traveled vs. Number of Drivers')
plt.xlabel('Number of Drivers (k)')
plt.ylabel('Total Distance Traveled')
plt.show()
```