The Vehicle Routing Problem with Heterogeneous Locker Boxes (VRPHLB)

Adam Bouafia

Matricula: 293137

Email: adam.bouafia@student.univaq.it

June 2024

Professor: Fabrizio Rossi
Professor: Andrea Manno

Course: Data Analytics and Data Driven Decision

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

Abstract

This document presents the implementation of the Vehicle Routing Problem with Heterogeneous Locker Boxes (VRPHLB) using Mixed-Integer Linear Programming (MILP). The problem involves optimizing the last-mile delivery process by incorporating locker boxes as intermediate delivery points. This approach enhances logistic efficiency and customer convenience. The document outlines the mathematical model, data generation, solution approach, and visualization of results.

1 Introduction

My name is Adam Bouafia. This work is part of the course on Data Analytics and Data Driven Decision, dated June 17, 2024, at the Università degli Studi dell'Aquila.

The Vehicle Routing Problem with Heterogeneous Locker Boxes (VRPHLB) is an extension of the classical Vehicle Routing Problem (VRP). In VRPHLB, parcels can be delivered to customers' homes or to locker boxes, which act as intermediate delivery points. This model aims to optimize the last-mile delivery process by incorporating alternative delivery points, enhancing logistic efficiency, and improving customer convenience.

1.1 Key Components

- **Depot**: The starting and ending point for delivery vehicles.
- **Customers**: Each customer has a specific demand that needs to be delivered.
- **Locker Boxes**: Stations with slots of different sizes where parcels can be delivered and picked up by customers.

1.2 Objectives

The primary objective is to minimize the total delivery cost, which includes:

- **Routing Cost**: The cost associated with the travel distance of vehicles.
- **Compensation Cost**: The cost incurred when parcels are delivered to locker boxes instead of directly to customers' homes.

2 Mathematical Model

2.1 Decision Variables

- x_{ij}^v: Binary variable indicating if vehicle v travels directly from node i to node j.
- y_{ik}^v: Binary variable indicating if customer i is served at locker box station k by vehicle v.
- z_i^v: Binary variable indicating if customer i is served at home.
- S_i: Continuous variable for the service start time at node i.

2.2 Objective Function

The objective is to minimize the total travel distance and compensation costs for using locker boxes:

$$\min \sum_{i \in N} \sum_{j \in N} \sum_{v \in V} d_{ij} x_{ij}^v + \sum_{i \in N} \sum_{k \in B_i} \sum_{v \in V} c y_{ik}^v \tag{1}$$

2.3 Constraints

1. **Route Continuity**: Ensures that if a vehicle enters a node, it must leave the node:

$$\sum_{j \in N \setminus \{i\}} x_{ij}^v = \sum_{j \in N \setminus \{i\}} x_{ji}^v \quad \forall i \in N, \forall v \in V \tag{2}$$

2. **Vehicle Usage**: Ensures that each vehicle is used at most once:

$$\sum_{j \in N \setminus \{0\}} x_{0j}^v \leq 1 \quad \forall v \in V \tag{3}$$

3. **Customer Service**: Ensures that each customer is served either at home or at a locker box:

$$\sum_{v \in V} \sum_{k \in B_i} y_{ik}^v + \sum_{v \in V} z_i^v = 1 \quad \forall i \in C \tag{4}$$

4. **Locker Box Capacity**: Ensures that the total demand served at a locker box does not exceed its capacity:

$$\sum_{i \in C} \sum_{v \in V} q_i y_{ik}^v \leq Q_k \quad \forall k \in B \tag{5}$$

5. **Time Windows**: Ensures that the service starts within the specified time window:

$$E_i \sum_{j \in N} \sum_{v \in V} x_{ij}^v \leq S_i \leq L_i \sum_{j \in N} \sum_{v \in V} x_{ij}^v \quad \forall i \in N \tag{6}$$

6. **Service Time Continuity**: Ensures continuity in service times:

$$S_i + s_i + d_{ij} - M(1 - x_{ij}^v) \leq S_j \quad \forall i, j \in N, \forall v \in V \tag{7}$$

2.4 Discussion on MILP

Mixed-Integer Linear Programming (MILP) is a powerful mathematical optimization technique that is widely used for solving combinatorial optimization problems, such as the VRPHLB. MILP models consist of linear objective functions and linear constraints, with some variables constrained to be integer values.

The VRPHLB is particularly suited for MILP due to the following reasons:

- **Binary Variables**: The decision variables x_{ij}^v, y_{ik}^v, and z_i^v are binary, representing whether a vehicle travels between nodes, serves a customer at a locker box, or serves a customer at home, respectively. MILP handles binary variables efficiently.
- **Complex Constraints**: The problem involves multiple constraints, including route continuity, vehicle usage, customer service, locker box capacity, and time windows. MILP allows the inclusion of such complex constraints in a structured manner.
- **Optimality**: MILP solvers, like Gurobi, provide optimal solutions within reasonable computational times for moderately sized instances, ensuring that the best possible routes and locker box assignments are found.

MILP was chosen for this project due to its ability to model the VRPHLB accurately and its effectiveness in finding optimal solutions for complex optimization problems.

3 Data Generation

To test our model, we generated a complex GraphML file representing the problem. This file includes nodes representing the depot, customers, and locker boxes, as well as edges with associated travel costs.

```
import networkx as nx
import random

# Create a new directed graph
G = nx.DiGraph()

# Add nodes with attributes
# Add depot
G.add_node(0, label="Depot", type="depot", x=random.uniform(0, 100), y=random.uniform(0, 100))

# Add customers
for i in range(1, 31):
    earliest_time = random.randint(8, 10)
    latest_time = random.randint(11, 12)
    G.add_node(i, label=f"Customer {i}", type="customer", demand=random.randint(1, 5), earliest=earl

# Add locker boxes
for i in range(31, 41):
    G.add_node(i, label=f"Locker {i-30}", type="locker", capacity=random.randint(10, 20), x=random.u

# Add edges with random weights
for i in G.nodes():
    for j in G.nodes():
        if i != j:
            G.add_edge(i, j, cost=random.uniform(1.0, 10.0))

# Save to GraphML file
nx.write_graphml(G, "vrphlb_graph.graphml")
```

4 Implementation and Solution

The model was implemented using Gurobi, a powerful optimization solver. Below are the steps for implementing and solving the VRPHLB.

```
# Extract nodes and edges
nodes = list(G.nodes(data=True))
edges = list(G.edges(data=True))

# Identify depot, customers, and locker boxes based on node attributes
depot = [n for n, attr in nodes if attr.get('type') == 'depot'][0]
customers = [n for n, attr in nodes if attr.get('type') == 'customer']
locker_boxes = [n for n, attr in nodes if attr.get('type') == 'locker']

# Extract distances from edges
distances = {(u, v): data['cost'] for u, v, data in edges}

# Define demands and capacities (assuming attributes are provided)
demands = {n: attr['demand'] for n, attr in nodes if 'demand' in attr}
capacities = {n: attr['capacity'] for n, attr in nodes if 'capacity' in attr}

# Define time windows
time_windows = {n: (attr['earliest'], attr['latest']) for n, attr in nodes if 'earliest' in attr and
```

```
# Define the compensation cost
compensation_cost = 5  # Example value

# Display extracted data
print("Depot:", depot)
print("Customers:", customers)
print("Locker Boxes:", locker_boxes)
print("Distances:", distances)
print("Demands:", demands)
print("Capacities:", capacities)
print("Time Windows:", time_windows)
print("Compensation Cost:", compensation_cost)

import gurobipy as gp
from gurobipy import GRB

# Create model
model = gp.Model("VRPHLB")

# Decision variables
x = model.addVars(distances.keys(), vtype=GRB.BINARY, name="x")
y = model.addVars(customers, locker_boxes, vtype=GRB.BINARY, name="y")
z = model.addVars(customers, vtype=GRB.BINARY, name="z")
S = model.addVars(G.nodes(), vtype=GRB.CONTINUOUS, name="S")

# Objective function: minimize total travel distance + compensation costs
model.setObjective(gp.quicksum(distances[i, j] * x[i, j] for i, j in distances) +
                   gp.quicksum(compensation_cost * y[i, k] for i in customers for k in locker_boxes)

# Constraints
# Each customer is visited exactly once (either at home or locker box)
for i in customers:
    model.addConstr(gp.quicksum(x[i, j] for j in G.nodes() if j != i) +
                    gp.quicksum(y[i, k] for k in locker_boxes) == 1)

# Route continuity
for i in G.nodes():
    model.addConstr(gp.quicksum(x[i, j] for j in G.nodes() if j != i) ==
                    gp.quicksum(x[j, i] for j in G.nodes() if j != i))

# Locker box capacity constraints
for k in locker_boxes:
    model.addConstr(gp.quicksum(demands[i] * y[i, k] for i in customers) <= capacities[k])

# Time window constraints
for i, j in distances:
    if i != depot and j != depot:
        model.addConstr(S[i] + distances[i, j] - (1 - x[i, j]) * 1e5 <= S[j])

# Optimize model
model.optimize()

# Display solution
if model.status == GRB.OPTIMAL:
    print("\nOptimal solution found:")
    for i, j in distances:
        if x[i, j].X > 0.5:
```

```
            print(f"Route: {i} -> {j}")
    for i in customers:
        for k in locker_boxes:
            if y[i, k].X > 0.5:
                print(f"Customer {i} assigned to locker box {k}")
else:
    print("No optimal solution found.")

DrawSol(G,x)
```

5 Results

After running the model, the optimal routes and locker box assignments were visualized to illustrate the solution. The results showed a significant improvement in delivery efficiency by utilizing locker boxes.

6 Conclusion

The implementation of the VRPHLB using MILP and Gurobi demonstrates the potential for optimizing last-mile delivery through the use of locker boxes. This approach not only reduces delivery costs but also increases customer convenience. Future work could explore dynamic locker box capacities and real-time route adjustments.

7 References

1 Grabenschweiger, J., Doerner, K. F., Hartl, R. F., & Savelsbergh, M. W. P. (2021). The vehicle routing problem with heterogeneous locker boxes. *Central European Journal of Operations Research*, 29, 113–142. https://doi.org/10.1007/s10100-020-00725-2