Ohio University

A Real-world Application of the Capacitated Vehicle Routing Problem

Jimmy Hayes
MATH 6940
Vardges Melkonian
Spring 2024

Table of Contents

1. Introduction

The Vehicle Routing Problem (VRP) represents a fundamental challenge in logistics and optimization, aiming to determine the most efficient routes for a fleet of vehicles to serve a set of customers or locations. Initially derived as an extension of the Traveling Salesman Problem (TSP), the VRP expands the objective to account for multiple vehicles, reflecting practical scenarios encountered in transportation and distribution networks. As such, the VRP serves as a foundational model for addressing routing and scheduling dilemmas across various industries, including freight transportation and last-mile delivery services.

Extensions of the VRP, such as the Capacitated Vehicle Routing Problem (CVRP), have emerged to address specific real-world constraints. In the CVRP, each vehicle operates under a maximum load capacity, introducing an additional layer of complexity to the optimization process. This extension mirrors the operational reality faced by logistics managers and transportation planners, who must account for vehicle capacities to ensure efficient resource utilization and timely deliveries.

In solving the VRP and its extensions, several complexities arise. Subtour elimination, for instance, presents a common issue wherein routes may inadvertently form loops or suboptimal circuits, leading to inefficiencies and increased operational costs. Moreover, the dynamics of modern urban environments introduce additional complexities, including traffic congestion, time windows, and diverse vehicle types, necessitating tailored solutions to address these challenges effectively.

This paper focuses on the CVRP within the context of urban transportation networks, with a specific emphasis on road networks in major cities. The paper will analyze a real-world road network and solve a theoretical CVRP in this network with AMPL software and the CBC solver to display the complexities and theoretical concerns of the CVRP.

2. Model formulation

For the general purposes of this paper, we have chosen to model the CVRP in the manner of *vehicle flow formulation*, which uses integer variables associated with each edge (road) of the graph (network), as outlined in Toth and Vigo [1] and Munari et al. [2]. The full formulation is as follows:

Decision variables.

$$x_{ij} = \begin{cases} 1 & \text{if edge i,j is part of solution} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{iv} = \begin{cases} 1 & \text{if node i is visited by vehicle v} \\ 0 & \text{otherwise} \end{cases}$$

Sets and parameters.

W = Set of warehouses and depot
W_0 = Set of warehouses
V = Set of vehicles
K = Number of vehicles
D_i = Demand at warehouse i
cap = Capacity of each vehicle

Objective function.

Minimize $\sum_{i \in W} \sum_{j \in W} c_{i,j} x_{i,j}$

Subject to:

1. Each node is visited by exactly one vehicle:

$$\sum_{v \in V} y_{iv} = 1, \quad \forall i \in W$$

This constraint ensures that each customer node is visited exactly once by exactly one vehicle.

2. Each non-depot warehouse is visited exactly once:

$$\sum_{j \in W, j \neq i} x_{ij} = 1, \quad \forall i \in W_0$$

This constraint guarantees that each non-depot warehouse is visited exactly once by one vehicle.

3. Flow balance at non-depot warehouses:

$$\sum_{j \in W, j \neq i} x_{ij} \leq \sum_{j \in W, j \neq i} x_{ji} + (1 - y_{iv}), \quad \forall i \in W_0, \forall v \in V, i \neq 0$$

This constraint ensures that the flow of vehicles into and out of non-depot warehouses is balanced, considering whether the warehouse is visited by a particular vehicle.

4. Number of vehicles leaving the depot:

$$\sum_{i \in W_0} x_{i0} = K$$

This constraint ensures that exactly K vehicles leave the depot.

5. Number of vehicles returning to the depot:

$$\sum_{j \in W_0} x_{0j} = K$$

This constraint ensures that exactly K vehicles return to the depot.

6. Capacity constraint for each vehicle:

$$\sum_{j \in W_0} d_j \cdot y_{jv} \leq \text{cap}, \quad \forall v \in V$$

This constraint ensures that the sum of demands of the visited warehouses by each vehicle does not exceed the capacity of the vehicle.

7. Each non-depot warehouse is visited by exactly one vehicle:

$$\sum_{v \in V} y_{iv} = 1, \quad \forall i \in W_0$$

This constraint ensures that each non-depot warehouse is visited by exactly one vehicle.

8. Connectivity constraint:

$$y_{iv} + x_{ij} \leq y_{jv} + 1, \quad \forall i \in W_0, \forall j \in W_0, \forall v \in V$$

This ensures that if warehouse i is visited by vehicle v, then the edge (i, j) can only be part of the solution if warehouse j is also visited by the same vehicle v.

9. No self-loops:
$x_{ii} = 0, \quad \forall i \in W$

This constraint ensures that there are no self-loops, meaning a vehicle cannot visit the same warehouse more than once.

10. At least one warehouse is visited by each vehicle:
$\sum_{i \in W_0} y_{iv} \geq 1, \quad \forall v \in V$

This ensures that each vehicle visits at least one non-depot warehouse.

11. Binary variables:
x, y binary

3. Methods

3.1. AMPL software & initial model

To solve the CVRP model formulated in Section 2 for a specific dataset, we have opted to use AMPL, an algebraic modeling language used to describe and solve linear and nonlinear optimization problems. AMPL solves mathematical problems via the many solvers it supports, including both commercial solvers (CPLEX, Minos, Gurobi, etc.) and open-source solvers (CBC, HiGHS, SCIP, etc.).

For the purposes of the data set outlined in Section 3.2, an open-source solver is required, as the demo version of the commercial solvers do not allow for problems beyond a certain number of variables and constraints, which grow exponentially in proportion to our data. We have opted to use CBC (or the COIN Branch and Cut solver), an open-source mixed-integer program (MIP) solver written in C++.

Below is the initial AMPL formulation of the problem. This formulation does not take subtours into account, which will be discussed in Section 3.3.

```
###-----Sets and parameters-----###
set W;                  # Set of all warehouses
set W_0;                # W without the depot
set V;                  # List of vehicles
param K >= 0;           # Number of available vehicles
param c{W,W};           # Cost of going from node i to j
param cap;              # Capacity of each vehicle
param d{W};             # Demand at each warehouse
```

```
###-----Decision variables-----###
var x{W,W} binary;     # 1 if edge {i,j} is part of solution, 0 o.w.
var y{W_0, V} binary;  # 1 if node i is visited by vehicle v, 0 o.w.


###-----Objective function-----###
minimize distance:
    sum{i in W, j in W} c[i,j] * x[i,j];

###---------Constraints--------###
subject to VisitOncePerVehicle {i in W_0}:
    sum {v in V} y[i,v] = 1;

subject to DepartOncePerVehicle {i in W_0}:
    sum {j in W: i != j} x[i,j] = 1;

subject to SubsequentNodeOnSameVehicle {i in W_0, v in V}:
    sum {j in W: i != j} x[i,j] <= sum {j in W: i != j} x[j,i] + (1 - y[i,v]);

subject to VehiclesLeavingDepot:
    sum{i in W_0} x[i, 0] = K;

subject to VehiclesEnteringDepot:
    sum{j in W_0} x[0, j] = K;

subject to RouteFeasibility {v in V}:
      sum {j in W_0} d[j] * y[j,v] <= cap;

subject to NodeVisitedOnce {i in W_0}:
      sum {v in V} y[i,v] = 1;

subject to RouteContinuity {i in W_0, j in W_0, v in V}:
      y[i,v] + x[i,j] <= y[j,v] + 1;

subject to CannotVisitItself {i in W}:
      x[i, i] = 0;

subject to UseAllVehicles {v in V}:
      sum {i in W_0} y[i,v] >= 1;
```

3.2. Real-world road network data

We seek to test the AMPL formulation of our model by testing it on a real-world data set, wherein real addresses of warehouses are collected, the distances between them are calculated, and a realistic set of optimal routes is studied in accordance with the capacities of each vehicle and the demands of each warehouse. For the purposes of this example, the capacities and demands are fictionalized, and the warehouses are collected from a list of empty warehouses for rent in Columbus [3]. We then analyze a *potential* CVRP, made clearer by the real-world choice of setting and nodes.

Below is our list of addresses for the depot and warehouses, including demand, as well as a picture showing the locations. Each set of coordinates was geocoded from addresses via Google's geocoding API [4].

#	Type	Latitude	Longitude	Address	Demand
0	Depot	40.0551762	-83.072309	1985 Henderson Rd, Columbus, OH 43220, USA	0
1	Warehouse	39.9891772	-82.9146133	885 Stelzer Rd, Columbus, OH 43219, USA	51
2	Warehouse	39.8927292	-83.0499288	3400 Southpark Pl, Grove City, OH 43123, USA	45
3	Warehouse	40.068717	-82.9959151	5057 Freeway Dr E, Columbus, OH 43229, USA	55
4	Warehouse	39.9877419	-82.831454	321 Outerbelt St, Columbus, OH 43213, USA	59
5	Warehouse	39.9720326	-83.0454325	740 Grandview Ave, Columbus, OH 43215, USA	41
6	Warehouse	39.9939739	-83.0381934	1156 Chambers Rd, Columbus, OH 43212, USA	42
7	Warehouse	39.9862862	-82.9010491	3850 E 5th Ave, Columbus, OH 43219, USA	46
8	Warehouse	39.9189505	-82.9348881	2430 Performance Way, Columbus, OH 43207, USA	50
9	Warehouse	40.11299	-82.9988092	651 Lakeview Plaza Blvd, Worthington, OH 43085, USA	61

Figure 3.2.1. Table of addresses, coordinates, and demand for depot and warehouses

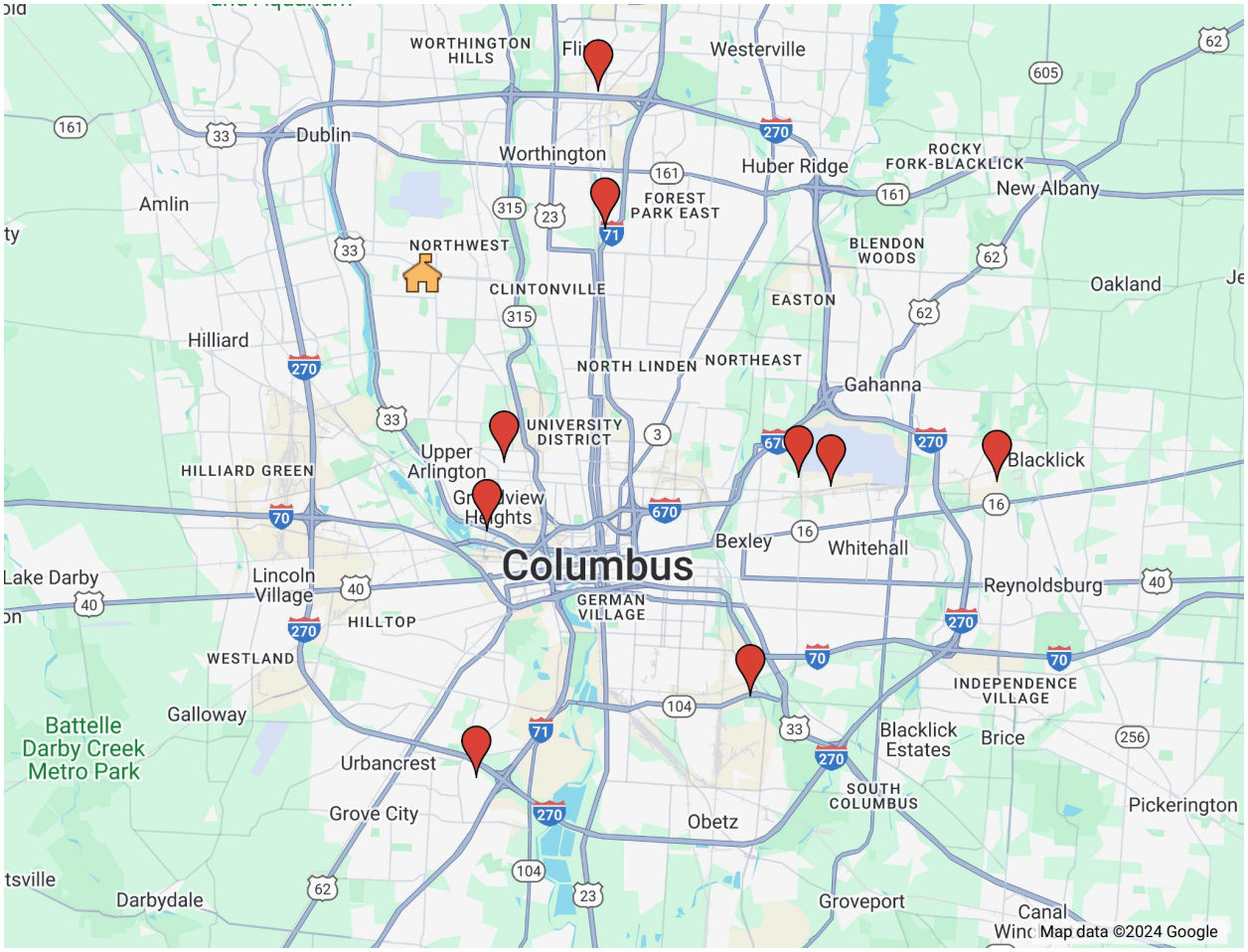

Figure 3.2.2. Map of Columbus, OH with markers on depot and warehouses.

Below is the AMPL data for the model. The distance matrix, signified by the `param c` in the code, represents the distance from warehouse *i* to warehouse *j*. One may note that, unlike many example distance matrices for VRPs and their variants, this matrix is not symmetrical across the diagonal. That is, the distance from warehouse *i* to warehouse *j* is not the same as the distance from warehouse *j* to warehouse *i*, although they are always close. This is a consequence of using real-world data collected from the Google Maps Routes API [5], as differences can be induced by traffic features such as one-way entrances and exits. The code used to obtain this data can be referenced in the appendix in Section 8.

```
set W := 0 1 2 3 4 5 6 7 8 9; # Set of all warehouses
set W_0 := 1 2 3 4 5 6 7 8 9;# W without the depot
set V := 1 2 3 4;             # List of vehicles
param K := 4;                  # Number of available vehicles

param c :          0         1         2         3         4         5         6         7         8         9 :=
        0          0     25760     28247      8934     40273     13165      8874     24855     25197     14890
        1      23322         0     24338     17897     10276     13300     14911      1567     10542     23814
        2      27751     22977         0     28299     31418     11452     19340     22072     13553     42090
        3       8915     19282     28038         0     29164     16698     13498     18377     21176      6174
        4      34135     12191     32839     28710         0     24113     25723      8216     19459     31195
        5      13056     15249     11454     18339     25131         0      3394     14344     16004     24088
        6       9507     17821     20308     12919     27703      3363         0     16916     20911     18873
        7      23732      1876     24747     18307      8868     13710     15321         0     11432     24224
        8      27624     10280     14319     21365     17648     16118     19213     12131         0     35128
        9      15839     26045     33769      5983     31387     22429     21185     27422     26906         0 ;

param cap := 150;     # Capacity of each vehicle
param d :=            # Demand at each warehouse
    0 0
    1 51
    2 45
    3 55
    4 59
    5 41
    6 42
    7 46
    8 50
    9 61;
```

3.3. Addressing subtours & final AMPL model

Naturally, in the process of solving the AMPL formulation of this model, subtours arise. A subtour is a tour (a route, in our case) in the model that, instead of returning to the depot, forms a cycle within itself, disconnected from the rest of the routes. This can happen for many reasons, often due to how the solver uniquely solves the problem (its heuristics, etc.). Subtours can be addressed with constraints, but not efficiently eliminated with one fell swoop, as the number of constraints required to fully eliminate all possible subtours grows very quickly in proportion to the number of nodes.

In order to address subtours in our example real-world problem, then, we run the initial model and determine where our first subtours are. The results of running and solving the model in AMPL are below:

```
cbc 2.10.10:              cbc 2.10.10: optimal solution; objective 157235
3508 simplex iterations
3508 barrier iterations
90 branching nodes
x [*,*]
:   0   1   2   3   4   5   6   7   8   9    :=
0   0   0   0   1   0   1   1   0   0   1
1   0   0   0   0   0   0   0   0   1   0
2   1   0   0   0   0   0   0   0   0   0
3   1   0   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   1   0   0
5   0   0   1   0   0   0   0   0   0   0
6   1   0   0   0   0   0   0   0   0   0
7   0   0   0   0   1   0   0   0   0   0
8   0   1   0   0   0   0   0   0   0   0
9   1   0   0   0   0   0   0   0   0   0
;

y [*,*]
:   1   2   3   4    :=
1   0   1   0   0
2   1   0   0   0
3   0   0   1   0
4   0   0   0   1
5   1   0   0   0
6   0   0   0   1
7   0   0   0   1
8   0   1   0   0
9   0   0   1   0
;
```

The routes above can be more clearly written as follows:

```
0-2-5-0
0-3-0
0-6-0
0-9-0
1-8-1
4-7-4
```

We see two subtours: one between warehouses 1 and 8, and another between warehouses 4 and 7. We eliminate them with the following constraints:

```
subject to SubtourElimination1:
      sum{i in S1, j in W diff S1} x[i,j] >= 1;
subject to SubtourElimination2:
      sum{i in S2, j in W diff S2} x[i,j] >= 1;
```

and define our subtour sets in the data section:

```
set S1 := 1 8;
set S2 := 4 7;
```

After implementing this subtour elimination, we run the model again, and determine if there are any more subtours. We repeat this process, writing new elimination constraints each time, until no subtours remain. Below is the full AMPL code, including both the model and data, with all subtour elimination constraints.

```
###-----Sets and parameters-----###
set W;                                  # Set of all warehouses
set W_0;                                # W without the depot
set V;                                  # List of vehicles
param K >= 0;                           # Number of available vehicles
param c{W,W};                           # Cost of going from node i to j
param cap;                              # Capacity of each vehicle
param d{W};                             # Demand at each warehouse
set S1;
set S2;
set S3;
set S4;
set S5;
set S6;
set S7;
set S8;
set S9;
set S10;
set S11;
set S12;
set S13;
set S14;
set S15;

###-----Decision variables-----###
var x{W,W} binary;                      # 1 if edge {i,j} is part of solution, 0 o.w.
var y{W_0, V} binary;                   # 1 if node i is visited by vehicle v, 0 o.w.


###-----Objective function-----###
minimize distance:
    sum{i in W, j in W} c[i,j] * x[i,j];

###---------Constraints--------###
subject to VisitOncePerVehicle {i in W_0}:
    sum {v in V} y[i,v] = 1;

subject to DepartOncePerVehicle {i in W_0}:
    sum {j in W: i != j} x[i,j] = 1;

subject to SubsequentNodeOnSameVehicle {i in W_0, v in V}:
    sum {j in W: i != j} x[i,j] <= sum {j in W: i != j} x[j,i] + (1 - y[i,v]);

subject to VehiclesLeavingDepot:
    sum{i in W_0} x[i, 0] = K;

subject to VehiclesEnteringDepot:
    sum{j in W_0} x[0, j] = K;
```

```
subject to RouteFeasibility {v in V}:
      sum {j in W_0} d[j] * y[j,v] <= cap;

subject to NodeVisitedOnce {i in W_0}:
      sum {v in V} y[i,v] = 1;

subject to RouteContinuity {i in W_0, j in W_0, v in V}:
      y[i,v] + x[i,j] <= y[j,v] + 1;

subject to CannotVisitItself {i in W}:
      x[i, i] = 0;

subject to UseAllVehicles {v in V}:
      sum {i in W_0} y[i,v] >= 1;

# Subtour Elimination constraint template
#subject to SubtourElimination1:
#     sum{i in S1, j in W diff S1} x[i,j] >= 1;

subject to SubtourElimination1:
      sum{i in S1, j in W diff S1} x[i,j] >= 1;

subject to SubtourElimination2:
      sum{i in S2, j in W diff S2} x[i,j] >= 1;

subject to SubtourElimination3:
      sum{i in S3, j in W diff S3} x[i,j] >= 1;

subject to SubtourElimination4:
      sum{i in S4, j in W diff S4} x[i,j] >= 1;

subject to SubtourElimination5:
      sum{i in S5, j in W diff S5} x[i,j] >= 1;

subject to SubtourElimination6:
      sum{i in S6, j in W diff S6} x[i,j] >= 1;

subject to SubtourElimination7:
      sum{i in S7, j in W diff S7} x[i,j] >= 1;

subject to SubtourElimination8:
      sum{i in S8, j in W diff S8} x[i,j] >= 1;

subject to SubtourElimination9:
      sum{i in S9, j in W diff S9} x[i,j] >= 1;

subject to SubtourElimination10:
      sum{i in S10, j in W diff S10} x[i,j] >= 1;

subject to SubtourElimination11:
      sum{i in S11, j in W diff S11} x[i,j] >= 1;

subject to SubtourElimination12:
      sum{i in S12, j in W diff S12} x[i,j] >= 1;
```

```
subject to SubtourElimination13:
      sum{i in S13, j in W diff S13} x[i,j] >= 1;

subject to SubtourElimination14:
      sum{i in S14, j in W diff S14} x[i,j] >= 1;

subject to SubtourElimination15:
      sum{i in S15, j in W diff S15} x[i,j] >= 1;

data;
set W := 0 1 2 3 4 5 6 7 8 9;       # Set of all warehouses
set W_0 := 1 2 3 4 5 6 7 8 9;        # W without the depot
set V := 1 2 3 4;                    # List of vehicles
param K := 4;                        # Number of available vehicles
param c :        0        1        2        3        4        5        6        7        8        9 :=
      0        0    25760    28247     8934    40273    13165     8874    24855    25197    14890
      1    23322        0    24338    17897    10276    13300    14911     1567    10542    23814
      2    27751    22977        0    28299    31418    11452    19340    22072    13553    42090
      3     8915    19282    28038        0    29164    16698    13498    18377    21176     6174
      4    34135    12191    32839    28710        0    24113    25723     8216    19459    31195
      5    13056    15249    11454    18339    25131        0     3394    14344    16004    24088
      6     9507    17821    20308    12919    27703     3363        0    16916    20911    18873
      7    23732     1876    24747    18307     8868    13710    15321        0    11432    24224
      8    27624    10280    14319    21365    17648    16118    19213    12131        0    35128
      9    15839    26045    33769     5983    31387    22429    21185    27422    26906        0 ;
param cap := 150;                    # Capacity of each vehicle
param d :=                           # Demand at each warehouse
    0 0
    1 51
    2 45
    3 55
    4 59
    5 41
    6 42
    7 46
    8 50
    9 61;
set S1 := 1 8;
set S2 := 4 7;
set S3 := 1 7;
set S4 := 4 8;
set S5 := 1 4;
set S6 := 2 8 7;
set S7 := 1 4;
set S8 := 1 8 7;
set S9 := 2 5;
set S10 := 7 8;
set S11 := 1 2 8;
set S12 := 4 5 7;
set S13 := 1 5;
set S14 := 2 8;
set S15 := 2 4 7;
end;
```

4. Results

4.1. AMPL results

After loading the model formulation at the end of Section 3.3 into AMPL, and selecting the CBC solver, we receive the following results:

```
cbc 2.10.10:              cbc 2.10.10: optimal solution; objective 193815
125828 simplex iterations
125828 barrier iterations
5040 branching nodes
x [*,*]
:   0   1   2   3   4   5   6   7   8   9    :=
0   0   0   1   0   0   1   1   0   0   1
1   1   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   1   0
3   1   0   0   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   1   0   0
5   0   0   0   0   1   0   0   0   0   0
6   1   0   0   0   0   0   0   0   0   0
7   1   0   0   0   0   0   0   0   0   0
8   0   1   0   0   0   0   0   0   0   0
9   0   0   0   1   0   0   0   0   0   0
;

y [*,*]
:   1   2   3   4    :=
1   0   0   1   0
2   0   0   1   0
3   0   0   0   1
4   1   0   0   0
5   1   0   0   0
6   0   1   0   0
7   1   0   0   0
8   0   0   1   0
9   0   0   0   1
;
```

The routes are as follows:

```
Vehicle 1: 0-7-4-5-0
Vehicle 2: 0-6-0
Vehicle 3: 0-1-8-2-0
Vehicle 4: 0-3-9-0
```

with a total cumulative distance between all routes of 193,815 meters.

4.2. Route pictures

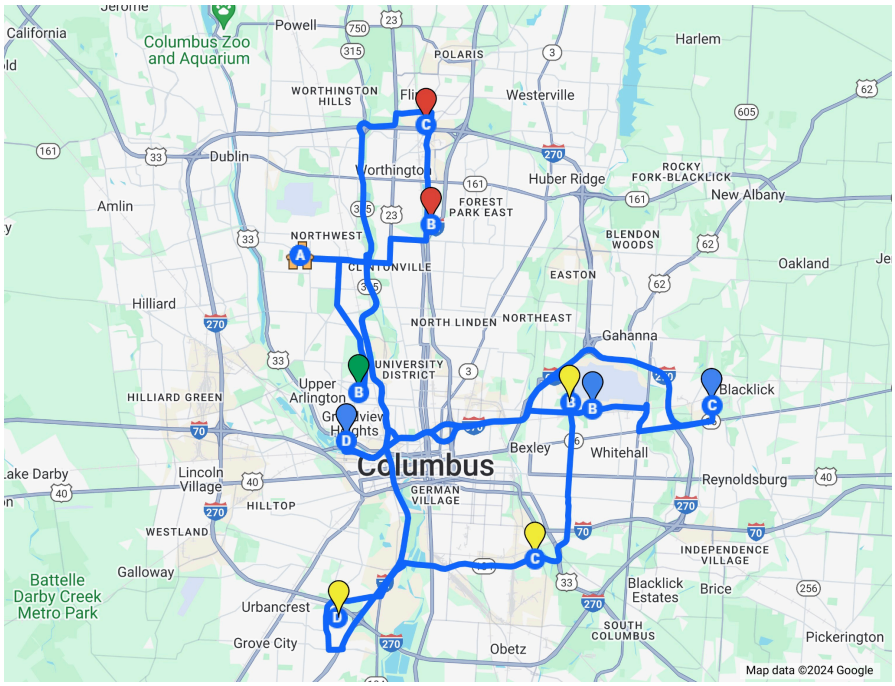

Figure 4.2.1. Map of Columbus, OH with all optimal routes outlined in blue.

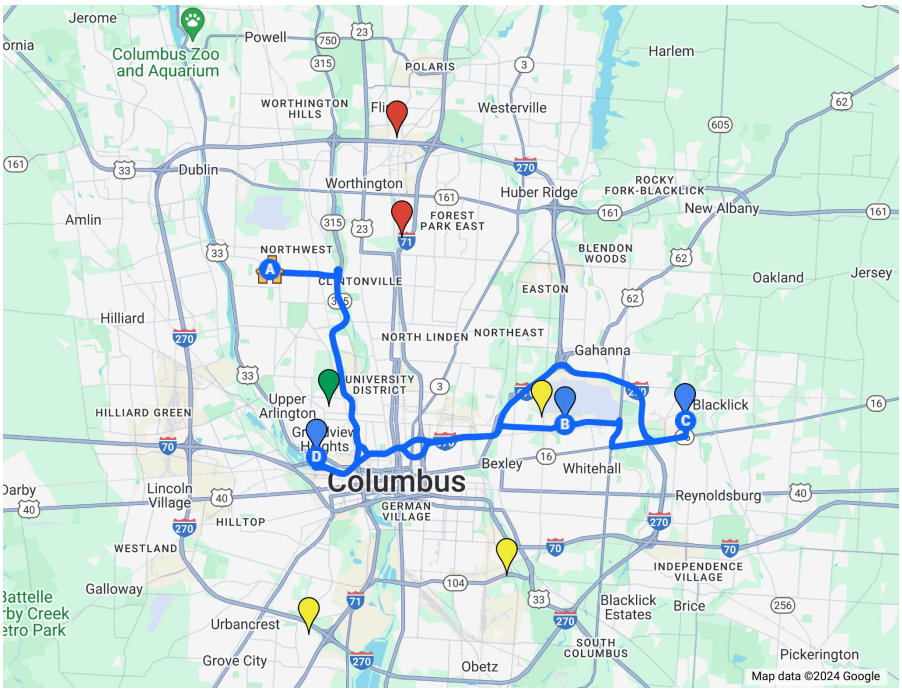

Figure 4.2.2. Optimal route taken by Vehicle 1.

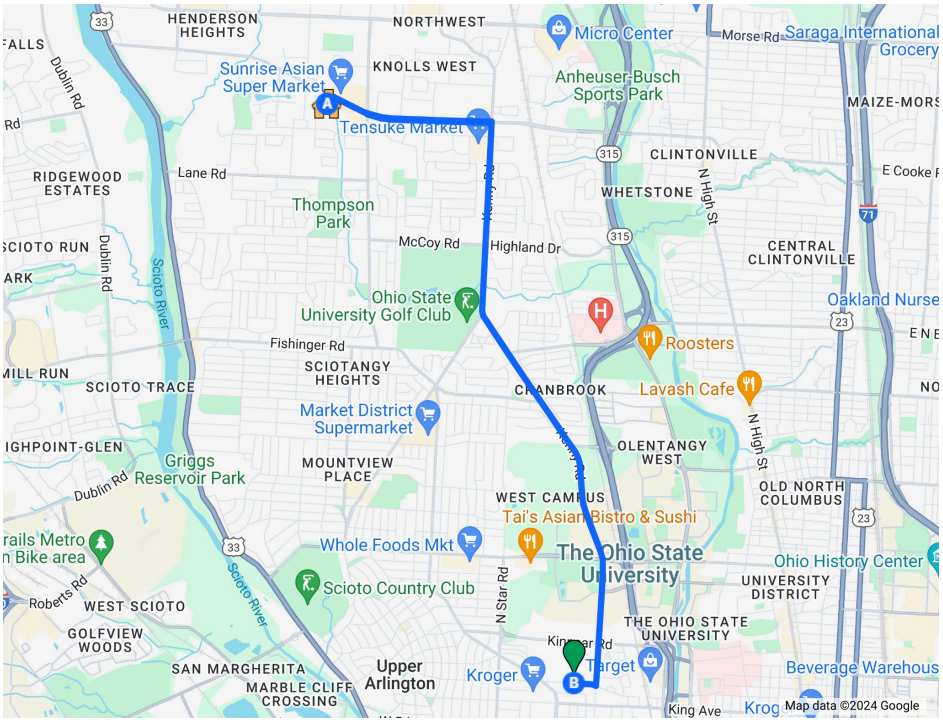

Figure 4.2.3. Optimal route taken by Vehicle 2.

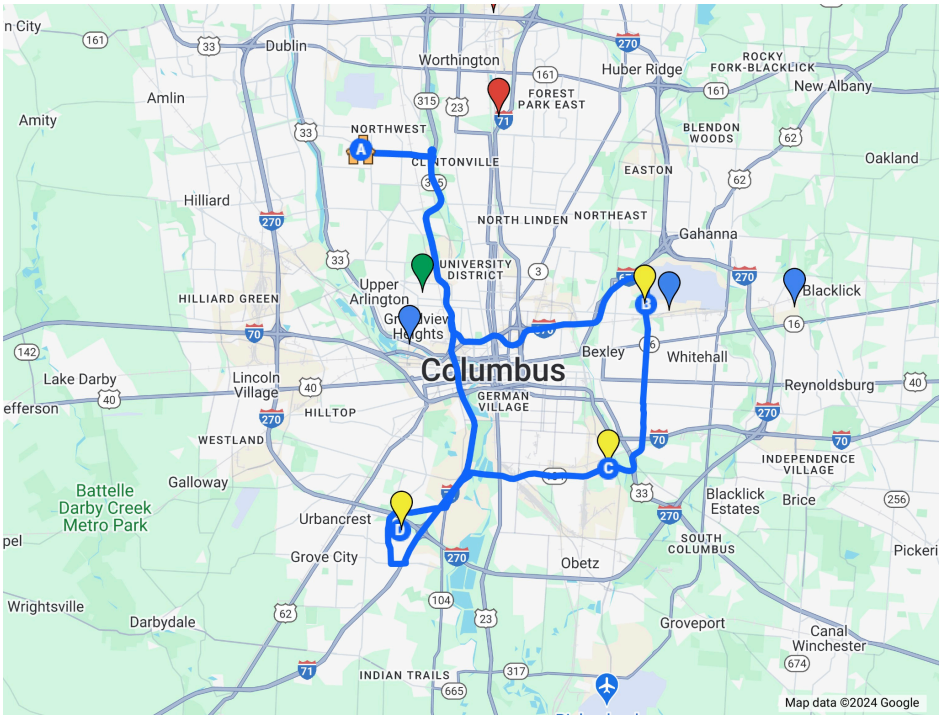

Figure 4.2.4. Optimal route taken by Vehicle 3.

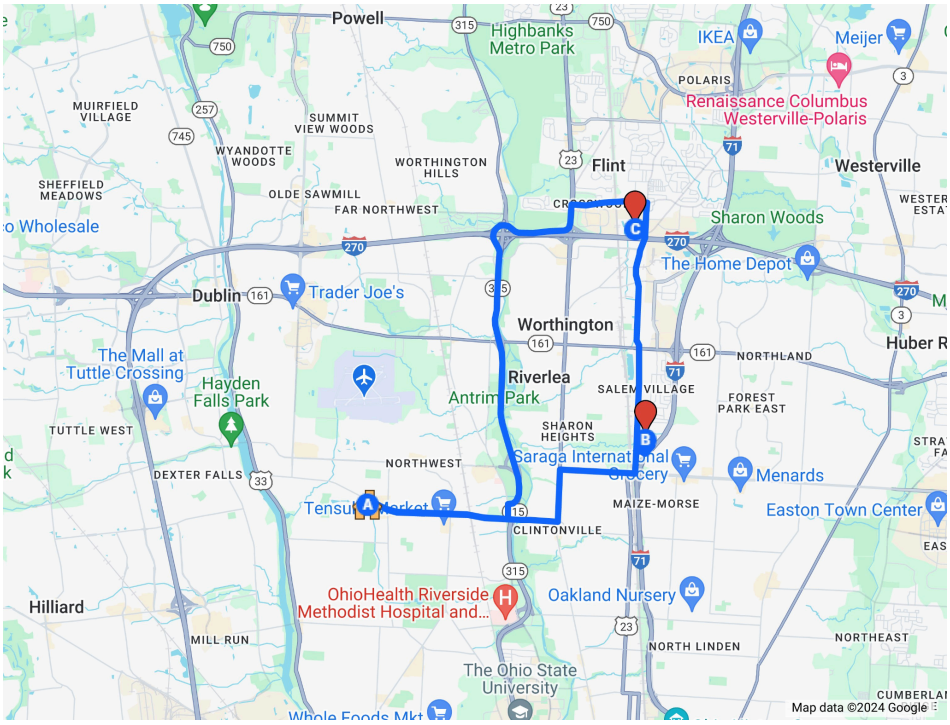

Figure 4.2.5. Optimal route taken by Vehicle 4.

5. Discussion

Looking at the visual representation of the routes brings several things into focus: firstly, that the warehouses on a route are physically close to each other. Barring special circumstances, it would be much more prohibitive for a vehicle to travel across town to a low-demand warehouse than to travel nearby to a high-demand one, even if the low-demand warehouse would affect its capacity constraints less. Many traveling salesman and vehicle routing problem formulations have an explicit integration of the triangle inequality to formalize this.

Secondly, there is little overlap between routes, but the overlap that does exist is of interest. Warehouse 1 and Warehouse 7 in the northeast section of Columbus, near the airport, are notably very close to each other, with Vehicles 4 and Vehicles 3 traveling to them respectively. Both of these routes were closer to their capacities (having three stops as opposed to Vehicle 1's two and Vehicle 2's one), and swapping either of these Warehouses for each other would have resulted in Vehicle 3 carrying 151 units of load, one unit over its maximum of 150. The visual representation alone can therefore be somewhat deceiving, as the proximity of the two may lead one to believe that another route could have been possible.

6. Conclusion

Our goal in formulating and solving the Capacitated Vehicle Routing Problem (CVRP) in order to explore how it applies to real-world urban transportation networks. By using both theoretical models and actual road network data from Columbus, OH, as well as leveraging AMPL software and the CBC solver, we have tackled the complexities of optimizing routes for a fleet of vehicles serving different warehouses with specific demands and capacity limitations.

Moreover, through subtour elimination techniques, we sought to find routes that not only balanced proximity between warehouses and the vehicles' capacity limits, but actually existed as feasible solutions to a real-world problem. The visual representation of our routes in Section 4.2 underscores the spatial relationships in these routes, and sheds light on the challenges and considerations involved in optimizing urban transportation networks, contributing to a better understanding of CVRP solutions in practical logistics scenarios.

Moving forward, Toth and Vigo outline several possible extensions to the CVRP, including implementing backhauling, time windows, and mixed service, and exploring further possibilities by combining these extensions into a complex class of problems. In any case, AMPL and the solvers integrated in it serve as solid and fundamental tools that aid in the formulation of these problems as well as solving them.

7. References

[1] Toth, Paolo, and Daniele Vigo. The Vehicle Routing Problem. Philadelphia, Society For Industrial And Applied Mathematics, 2002.

[2] Munari, Pedro Augusto et al. “A Generalized Formulation for Vehicle Routing Problems.” ArXiv (Cornell University), 1 Jan. 2016, https://doi.org/10.48550/arxiv.1606.01935.

[3] “Columbus Warehouses for Rent & Lease | LoopNet.” LoopNet, 2024, www.loopnet.com/search/listings/warehouses/columbus-oh/for-lease/.

[4] Schneider, Adam. “GPS Visualizer’s Easy Batch Geocoder: Convert Addresses to Coordinates.” Www.gpsvisualizer.com, www.gpsvisualizer.com/geocoder/. Accessed 3 May 2024.

[5] “Google Maps Platform Documentation | Routes API.” Google Developers, developers.google.com/maps/documentation/routes.

8. Appendix

8.1. Batch file for collecting distance data

```
curl -X POST -d '{
  "origins": [
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.0551762,
            "longitude": -83.072309
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9891772,
            "longitude": -82.9146133
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.8927292,
            "longitude": -83.0499288
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.068717,
            "longitude": -82.9959151
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9877419,
            "longitude": -82.831454
          }
```

```
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9720326,
            "longitude": -83.0454325
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9939739,
            "longitude": -83.0381934
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9862862,
            "longitude": -82.9010491
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9189505,
            "longitude": -82.9348881
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.11299,
            "longitude": -82.9988092
          }
        }
      }
    }
  ],
  "destinations": [
      {
```

```
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.0551762,
            "longitude": -83.072309
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9891772,
            "longitude": -82.9146133
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.8927292,
            "longitude": -83.0499288
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.068717,
            "longitude": -82.9959151
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9877419,
            "longitude": -82.831454
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9720326,
            "longitude": -83.0454325
          }
```

```
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9939739,
            "longitude": -83.0381934
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9862862,
            "longitude": -82.9010491
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 39.9189505,
            "longitude": -82.9348881
          }
        }
      }
    },
    {
      "waypoint": {
        "location": {
          "latLng": {
            "latitude": 40.11299,
            "longitude": -82.9988092
          }
        }
      }
    }
  ],
  "travelMode": "DRIVE",
  "routingPreference": "TRAFFIC_AWARE"
}' > output.txt \
-H 'Content-Type: application/json' -H 'X-Goog-Api-Key: REDACTED' \
-H 'X-Goog-FieldMask: originIndex,destinationIndex,duration,distanceMeters,status,condition'
\
'https://routes.googleapis.com/distanceMatrix/v2:computeRouteMatrix'
```

8.2. Python code for converting distance data to matrix

```
import json

def create_distance_matrix(text):
    # Parse the input text
    data = json.loads("[" + text.replace("} , {", "}, {") + "]")

    # Find the maximum index to determine the matrix size
    max_index = max(max(item["originIndex"], item["destinationIndex"]) for item in data if
"distanceMeters" in item)

    # Initialize an empty matrix
    distance_matrix = [[0] * (max_index + 1) for _ in range(max_index + 1)]

    # Fill the distance matrix
    for item in data:
        if "distanceMeters" in item:
            origin_index = item["originIndex"]
            destination_index = item["destinationIndex"]
            distance = item["distanceMeters"]
            distance_matrix[origin_index][destination_index] = distance

    return distance_matrix

def print_distance_matrix(matrix):
    size = len(matrix) - 1
    print("    ", end="")
    for i in range(size + 1):
        print(f"{i:10}", end="")
    print(" :=")
    for i in range(size + 1):
        print(f"    {i}", end="")
        for j in range(size + 1):
            print(f"{matrix[i][j]:10}", end="")
        print()
```

8.3. Google Maps with route data

Below is a link to an interactive Google Maps page with warehouse locations and route data. You can use the checkbox on the left hand side to see individual routes.
https://www.google.com/maps/d/u/0/edit?mid=1NkvVEPoi9VJ9SIm-ElXwLa36US2j7o8&usp=sharing