Route Optimization for Delivery Businesses

Eng M 540 Final Project

Fall 2018

Igor Malis 1102098

Anton Kroess 1593158

Son Le 1432746

Table of Contents

Abstract

In this report, we take on the role of a local delivery business that would like to minimize costs associated with delivering a product to their customers, located in the Edmonton area. The day-to-day operating decisions faced by this business are primarily logistics-related and can influence the profitability of the company.

For instance, questions like:

- What route should our delivery trucks take?
- How does the size of truck rented (and the amount of product that can be delivered in a single trip) impact the optimal route?
- Is there a benefit in renting an additional location to store product?
- Should we rent an additional truck?
- How much do these decisions impact our delivery costs?

can help the business understand how they can manage their delivery expenses effectively and have a lot of value for the business, especially as the business expands and attracts more and more customers. We explore utilizing integer programming (IP) optimization techniques to answer these valuable questions.

We first begin by presenting a mathematical model of problem, in the form of an integer program. We define the problem inputs, decision variables, objective function, as well as constraints, for several cases of the problem:

- Symmetric and asymmetric distance matrices
- Satisfying customer demand using a single delivery route, as well as multiple routes

We pick 52 random Edmonton addresses to solve our problem instance on. We utilize two techniques to obtain a distance matrix (a necessary input to our problem):

- Estimate the distance matrix from the GPS coordinates (lat, lng) of the 52 locations
- Retrieve an accurate (driving profile) distance matrix from Google Maps

We formulated our problem in Python, and utilize several solvers throughout the project to solve the problem (ie. coin-or-branch and cut, GLPK, SCIP, and Gurobi), through the use of two freely available Python connectors (PuLP and PySCIPOpt). For our base case of 50 customers, a single truck and warehouse, and a truck capacity of 50, we obtained a delivery cost of $ 252.28.

We present a visual representation of the solutions, along with the obtained optimal objective function value, and also perform sensitivity analysis on several of the inputs:

- Varying truck capacity between 50 and 30 deliveries
- Examining the impact of an additional warehouse
- Having two delivery trucks available instead of one

The results of these scenarios were analyzed, and we found that the truck capacity had the largest impact on the delivery cost, decreasing the optimal route cost to $ 250.570.

Introduction

The motivation behind this project was to model how a small business could utilize data driven decision making. With advances in technology and available data, businesses have the ability to make better more highly informed decisions, rather than operating on "gut-feeling". While many areas of an online grocers' business could benefit from data driven decision making (directed marketing to give an example), we opted to focus our model on the logistical side of the business. This approach presented a clear idea of which factors that drive costs and which variables we have under our control.

Background

An online grocery company (Company Co.) has recently begun operating in the Edmonton market. Company Co. allows customers to order their food and grocery products online from the comfort of their homes, with convenient delivery all the way to the door. Most of the customer base are repeat customers, requiring weekly deliveries. For most customers, the exact delivery day is not important, as long as their order is delivered within a 5-day window every week. Each delivery is composed the product(s) ordered by the customer and a small flyer, placed in a standard-size, branded package (a cardboard-based box). The boxes are shipped by a driver working directly for Company Co. using leased vehicles.

The difficulty faced by Company Co. revolves around scheduling deliveries and route-planning. Ideally, they would like to generate a pre-planned route for their driver every day, so that distribution expenses are minimized, while ensuring that all orders for the week are fulfilled. They pay a fixed monthly price for every vehicle leased, and thus would like to use the least number of trucks possible, while also keeping driver overtime under control. Currently Company Co. has 200 customers and deliver 5 times per week, the largest number of deliveries required in one day is 50. This report will focus on a sample of 50 customers, which represents the largest number of deliveries Company Co. would make in one day. The addresses of 50 customers and the company warehouse have been plotted on a map of Edmonton in figure 1 below.

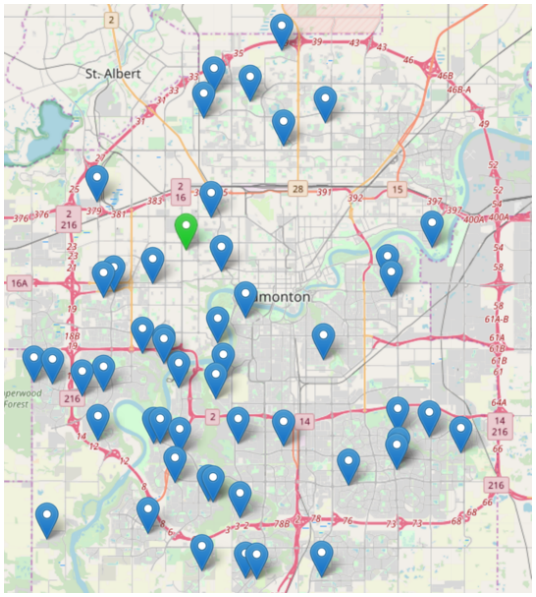


Figure 1 - Map of customer address sample (blue) and warehouse (green)

The truck that Company Co. is currently using has a capacity of 50 boxes (equivalent to 50 customer orders) and uses roughly 2.5 liters of fuel per 10 km of city driving. As the lease of this truck will expire soon Company Co. wants to know if they would be better off leasing 2 smaller trucks with a capacity of 30 boxes each. The driver is paid an hourly wage of $25. Company Co. would like to optimize their delivery route(s), such that they incur minimum costs for deliveries.

Model Formulation

The aim of the problem model is to optimize the delivery operations of Company Co. The model relates the primary delivery costs: driver hourly wage and fuel costs (stated in terms of the cost per km) with decision variables for each possible step of the delivery route. The model also allows Company Co. to analyse the effect of using more than one truck with varying truck capacity, as well as using an additional warehouse.

This gives the model the ability to not only plan day to day operations, but also provides decision making input for how many trucks to rent and drivers to hire. Additionally, the model could suggest how many warehouses Company Co. should have to ensure every truck has a realistic carrying capacity; i.e. it might not be realistic to have one truck making 50 deliveries to 50 different locations in one day.

Assumptions and simplifications

To keep the problem manageable in terms of complexity we assume that delivery routes can be planned without taking the production schedule into account; all orders are prepped the day/night before. The model also does not take into consideration the customers preference for which day of the week, or time of day, that delivery is made. In the future Company Co. could expand the model with more constraints to include these features, but for now they are left out.

The costs for travelling each distance are calculated based on distance travelled in kilometers assuming good road and traffic conditions, such as:

- All roads are dry and flat,
- There is negligible traffic, and
- There are no roadblocks.

The delivery truck(s) are leased by Company Co at a fixed rate, i.e. the distance driven by each truck does not increase the cost of the lease.

Solution Format

The analysis will calculate, based on the number and locations of the destinations for delivery, the optimal travel path or the travel path which will incur the lowest costs. In simpler terms, say there are 50 locations chosen in random that the company must deliver to without neglecting a single location per day. Upon choosing the number of trucks, the carrying capacity for each truck, and the number of warehouses used, the optimal path can be calculated.

Problem Formulation

The problem which Company Co. faces share many similarities with what's known as a traveling salesman problem (TSP). TSPs are a form of network problem where each node of the network only can be visited once, resembling the path of a traveling salesman visiting different cities.

Both binary and integer variables were utilized in the analysis (in the case of cutting planes method, the variables were initially declared as continuous, and later converted to binary/integer). Binary variables can only be assigned a value of either 1 or 0, while integer variables can be assigned any integer in the range specified. They are further categorized into two types: inputs and decision variables.

Inputs

Inputs are variables that are provided to the solver as constants. They are predetermined for a given instance of the problem.

$\boldsymbol{n_c}$: number of customers

This variable indicates the total number of delivery locations that is chosen for analysis. For example, $n = 50$ means that there is a total of 50 locations, or customers, that Company Co. must make deliveries to.

$\boldsymbol{n_w}$: number of warehouses

This variable indicates the number of warehouses available that the trucks could start from. It could also serve as a location where the trucks could stop by and restock to make further deliveries.

$\boldsymbol{D}$: distance matrix

This is a $(n_c + n_w) \times (n_c + n_w)$ matrix indicating the distance between any pair of locations based on their indices. $n_c + n_w$ is the total number of locations available for the analysis. For example:

$d_{i,j}$: the distance from node i (warehouse or customer) to node j (warehouse or customer)

We used two distance matrices in our analysis (see next section for more details):

1) Approximation using GPS coordinates (symmetric)
2) One obtained from Google Maps Distance Matrix API (asymmetric) with 'driving' mode, which returns distance calculations using the road network (see 'Solutions' section for more details)

$\boldsymbol{t}$: number of trucks

This variable indicates the number of trucks being utilized to make sure every single location, or customer, is covered in the solution.

$\boldsymbol{c}$: truck capacity

This is the carrying capacity of each truck being utilized that is equal to the number of locations, or customers, inputted. This, however, does not mean that $c = n_c$. For example, n_c could be 50 and c could be 30, meaning that even though there are 50 locations that Company Co. must make deliveries to, each of their trucks could only carry enough to deliver to 30 of those 50 locations.

$\boldsymbol{c_f}$: Cost of fuel coefficient

This is a constant indicating the cost of fuel per kilometer of distance travelled by a truck. It is here assumed that a truck with capacity for 50 boxes uses 2.5 liters of fuel per 10km, whilst a truck with capacity for 30 boxes uses 2.2 liters of fuel per 10 km. The cost of fuel is estimated to be $1.25 per liter. This gives us:

$$\boldsymbol{c_{f50}} = \frac{2.5 * 1.25}{10} = 0.3125 \left(\frac{\$}{km}\right)$$

$$\boldsymbol{c_{f30}} = \frac{2.2 * 1.25}{10} = 0.275 \left(\frac{\$}{km}\right)$$

$\boldsymbol{c_w}$: Cost of wage coefficient

This is a constant indicating the cost of a driver's hourly wage, but in order to make it proportional to distance travelled we express it per km. Here we assume that city traffic allows the truck to travel at an average speed of 30km per hour. With the driver being paid a wage of being $25 per hour this gives us:

$$c_w = \frac{25}{30} = 0.833 \left(\frac{\$}{km}\right)$$

Decision Variables

Decision variables are variables that the solver is allowed to vary within the range allowed by the constraints to find an optimal solution. The solution the solver comes up with will be composed of the values for the decision variables, as well as the objective function value.

$\boldsymbol{x_{i,j}}$: whether path from node i to node j is taken (binary)

These binary variables indicate which path the truck should choose to make the delivery. The subscripts i and j are indices of the pair of locations on the which path the truck is taking. For example, if $x_{1,2} = 1$, then the path from location #1 to location #2 is chosen to be a part of the final delivery path for one of the total number of trucks being utilized by Company Co.

$\boldsymbol{L_i}$: node order (integer between 1 and min(n_c, c))

These integer variables indicate where location i is found in the sequence of locations that is being used in the analysis, with the first customer visited corresponding to $L_i = 1$.

Note: this decision variable is only utilized in the multiple-path formulation.

Objective function

We define our objective function as a cost minimization, where the total length of the minimum distance route is multiplied by a constant factor, resulting in an estimate of the cost of the route, including labour and fuel costs.

Define our locations as two sets:

$\boldsymbol{w}$: $\boldsymbol{set\ of\ warehouses}\ \{\mathbf{1}, \dots, \boldsymbol{n_w}\}$
$\boldsymbol{c}$: $\boldsymbol{set\ of\ customers}\ \{\mathbf{1}, \dots, \boldsymbol{n_c}\}$

The objective function is:

$$\boldsymbol{minimize}: \quad z = \sum_{i \in \{w,c\}} \sum_{\substack{j \in \{w,c\} \\ i \neq j}} (c_f + c_w) \cdot d_{i,j} \cdot x_{i,j}$$

This minimizes the total cost incurred by the business to make the delivery, based on the costs of fuel and labour.

Constraints

Case 1: Single Path

The first formulation generates a solution using only a single path. This implies that the truck capacity, must be bigger or equal to the number of customers: $c \geq n_c$

In addition, for this case, the following must be true as well:

- Number of warehouses, $n_w = 1$
- Number of trucks, t = 1

Two Connections Per Node Constraint

There are two versions of this constraint, one for symmetric and one for asymmetric distance matrices:

Table 1: Symmetric and Asymmetric Constraints for the Single Path Formulation

Symmetric	Asymmetric
$\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{i,j} + \sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{j,i} = 2 \quad for\ \forall\, i \in \{w,c\}$	$\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{i,j} = 1 \quad for\ \forall\, i \in \{w,c\}$ $\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{j,i} = 1 \quad for\ \forall\, i \in \{w,c\}$

The symmetric formulation allows us to use fewer constraints. We use this problem formulation to solve distance matrix (1).

The asymmetric distance matrix requires double the number of constraints. This ensures that for each location, we have one path to and one path from that location. This formulation is required to accurately solve distance matrix (2).

Attempting to solve an asymmetric distance matrix with the symmetric problem formulation would result in an artificially low objective function value, as the optimal solution could contain some locations with 2 paths going to the location, or 2 paths coming from the location.

Subtour Elimination Constraint

A common obstacle in this type of problem is to eliminate subtours, where the solution involves every node being visited once, but not in one closed loop. This renders an infeasible solution to the problem and constraints must be in place to eliminate subtours from forming. A visualization of a subtour compared to a closed loop is presented in figure 2 below:

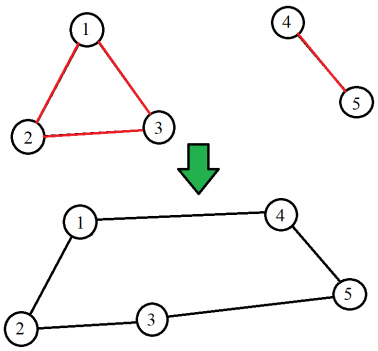


Figure 2 - Subtour visualization

Subtour elimination approach

Our initial understanding of linear programming led us into method of adding one constraint for each possible subtour, in retrospect this turned out to be a version of the Miller-Tucker-Zemlin (MTZ) formulation. It became apparent that as the number of customers (n) is increased, the number of constraints that are required to fully describe the problem and guarantee an optimal solution increases very quickly, much faster than n. This is known as an 'NP-hard' problem.

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{i,j} \leq |S| - 1 \qquad for\ \forall\ S \sqsubseteq \{w,c\};\ S \neq \emptyset; |S| > 1$$

This says that every proper subset of the full set of locations (warehouses & customers) can be at most one less than the number of locations in the subset (ignoring the empty set and sets of size 1). If this constraint is violated, then we have subtour.

Subtour Elimination Improvement: Cutting Planes

It quickly proved infeasible to compute a solution for more than 20 customers using the MTZ formulation. We then utilized PySCIPOpt to selectively add constraints, as subtours are found in the current solution, until a solution with no subtours was found. For more information, see [15].

Case 2: Multiple Paths

The multiple path formulation is required to solve a problem where at least one of the following conditions is true:

1) Truck capacity is less than the number of customers: $c < n_c$
2) More than one truck is used to make the deliveries: $t > 1$
3) More than one warehouse is available: $n_w > 1$

Two Connections Per Node Constraint

To address point (3), we reformulate the constraint used in case (1) such that *i* does not include {w}:

Table 2: Symmetric and Asymmetric Constraints for the Multiple Paths Formulation

Symmetric	**Asymmetric**
$\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{i,j} + \sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{j,i} = 2 \qquad for\ \forall\ i \in \{c\}$	$\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{i,j} = 1 \qquad for\ \forall\ i \in \{c\}$ $\sum_{\substack{j \in \{w,c\} \\ i \neq j}} x_{j,i} = 1 \qquad for\ \forall\ i \in \{c\}$

This ensures that each customer location still requires two edges to be connected to it. However, this doesn't require warehouses to be connected by two edges. This gives the

formulation more flexibility because not all warehouses are required to be in a path, and a warehouse may be used for more than a single path (if it is optimal).

Start/End at Warehouse Constraint

We define an input variable, which indicates how many total paths are required to solve the problem:

$$\boldsymbol{paths}: \quad \boldsymbol{max}\left(\boldsymbol{ceil}\left(\frac{\boldsymbol{n_c}}{\boldsymbol{c}}\right), \boldsymbol{t}\right)$$

This ensures that we have enough paths to cover all customers, given that the maximum number of customers visited in a single path can be c. It also allows us to use more than the minimum number of paths required, if more trucks are available.

Since we removed {w} from the previous constraint, we need an additional constraint to ensure that each route starts and ends at a warehouse:

$$\sum_{i \in \{w\}} \sum_{\substack{j \in \{c\} \\ i \neq j}} x_{j,i} \quad = \quad paths$$

Reformulating the constraint in this way allows any of the warehouses to be used. Using this constraint, it is even possible for the path to start at one warehouse and end at another, if this results in a smaller total distance travelled than starting and ending at the same warehouse.

Node Order Constraint

We use a constraint utilizing the node order, L_i, specifying that L_i must increase with each customer visited. This allows us to limit the maximum length of a single path:

$$L_i - L_j + n_c \cdot x_{i,j} \leq n_c - 1 \qquad for\ all\ i,j \in \{c$$

Data Set

Addresses

We picked a random sample of 50 Edmonton addresses from the City of Edmonton's online Property Assessment Data dataset [2]. We also picked two warehouse locations in industrial areas: one in the northwest side of Edmonton, and one on the southeast side:

The addresses came with GPS coordinates, which are listed in Appendix 1. We utilized Leaflet[3] to plot the data points on an interactive map, using the openstreetmap.org standard tiles[4] (see figure 3). The code for this is provided in Appendix 2. We initially prepared our data set in an Excel file, and then imported it into .csv (comma-separated value) format, so that it could be imported into our solver (see next section).

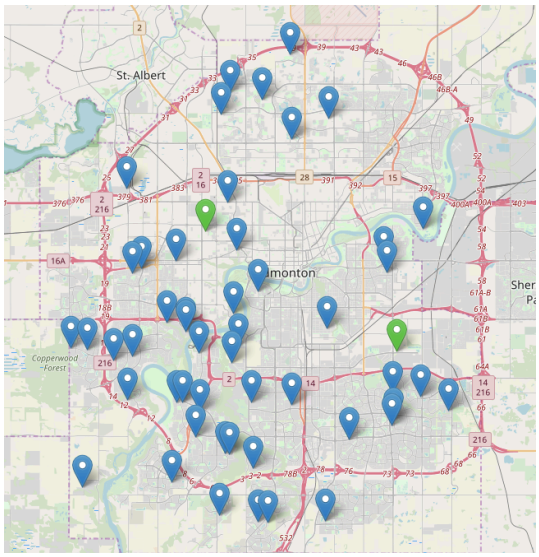


Figure 3 - Addresses and both warehouses

Distance Matrix Approximation

We initially generated an approximation of the distance matrix based on straight-line distance between locations:

$$d_{i,j} = \sqrt{\left(110.574km \times \left(lat_i - lat_j\right)\right)^2 + \left(111.320km \times \cos\left(\frac{lat_i + lat_j}{2}\right) \times \left(lng_i - lng_j\right)\right)^2}$$

We utilized an Excel spreadsheet to calculate this matrix. The result is provided in Appendix 3.

MapBox Distance Matrix

We initially attempted to use the MapBox Matrix API to retrieve our distance matrix [5]. We used the 'mapbox/driving' profile to retrieve estimates of travel times and distances between each pair of locations. This API has a limitation because it allows a maximum of 25 addresses per request (625 matrix elements), with the option to specify either any or all of those addresses as the source and destination addresses.

This meant we needed to make many requests to their API to retrieve the entire distance matrix of 52 * 52 (max 2 warehouses, 50 customers). This is 2704 elements, which is much higher than 625 (25 * 25 = maximum number that can be returned in a single request). For this reason, we had to split up the distance matrix into multiple segments, and rebuild the distance matrix from the segments:

Table 3: 5 x 5 Segmentation of the Distance Matrix

	1-12	13-24	25-36	37-48	49-52
1-12	1	6	11	16	21
13-24	2	7	12	17	22
25-36	3	8	13	18	23
37-48	4	9	14	19	24
49-52	5	10	15	20	25

We utilized a short Python script to load the locations from the dataset file (csv format), split up the locations into sets of 12, retrieve the distance & time matrices in 25 requests, and save the results in csv. The code for this is provided in Appendix 6.

We found the MapBox matrix to be highly asymmetric and to not produce reliable results, and we used Google Maps instead (see next section). The MapBox distance matrix is provided in Appendix 4.

The MapBox API has the following endpoint for retrieving a distance matrix:

```
https://api.mapbox.com/directions-matrix/v1/mapbox/driving/53.512735,-113.423675;53.572226,-
113.58381;53.45332914,-113.5832276;53.45903088,-113.4195745;53.5459037,-113.5493732;53.534394,-
113.6363747;53.56959423,-113.5568043;53.44479748,-113.5553688;53.41155502,-
113.4750256;53.46937759,-
113.5031153?destinations=0;1;2;3;4;5;6;7;8;9&sources=0;1;2;3;4;5;6;7;8;9&annotations=duration,dis
tance&access_token=API_KEY
```

Google Maps Distance Matrix

We used Google Map's Distance Matrix API to retrieve an estimate of the distance matrix for our 52 locations [7]. This API has a limitation and can return a maximum of 100 matrix elements per request. This required us to split up the distance matrix into 36 requests:

Table 4: 6 x 6 Segmentation of the Distance Matrix

	1-10	11-20	21-30	31-40	41-50	51-52
1-10	1	7	13	19	25	31
11-20	2	8	14	20	26	32
21-30	3	9	15	21	27	33
31-40	4	10	16	22	28	34
41-50	5	11	17	23	29	35
51-52	6	12	18	24	30	36

We used a short python script to load the data set (in csv format), split up the locations into sets of 10, and perform the request for every combination of sets. The code for this is provided in Appendix 7. The distance matrix obtained is provided in Appendix 5.

The requests performed to the Distance Matrix API are of the following format:

```
https://maps.googleapis.com/maps/api/distancematrix/json?origins=53.512735,-
113.423675|53.572226,-113.58381|53.45332914,-113.5832276|53.45903088,-113.4195745|53.5459037,-
113.5493732|53.534394,-113.6363747|53.56959423,-113.5568043|53.44479748,-
113.5553688|53.41155502,-113.4750256|53.46937759,-113.5031153&destinations=53.512735,-
113.423675|53.572226,-113.58381|53.45332914,-113.5832276|53.45903088,-113.4195745|53.5459037,-
113.5493732|53.534394,-113.6363747|53.56959423,-113.5568043|53.44479748,-
113.5553688|53.41155502,-113.4750256|53.46937759,-113.5031153
```

This is different than the MapBox endpoint, because here the origins and destinations parameters both accept a vector of GPS (lat,lng) coordinates.

Solving the Problem

To solve a problem with 50 customers and 2 warehouses requires a distance matrix that is 52 * 52 = 2704 elements. A minimum of 2704 – 52 = 2652 $x_{i,j}$ decision variables are required. Due to the difficulty in manually inputting this many decision variables (and constraints), we decided to use a programmatic approach:

- Develop a simple Python program that could read our dataset.csv
- Create a linear integer programming (LIP) model
- Add the $x_{i,j}$ decision variables dynamically, based on the size of the data set
- Add the necessary constraints
- Use a Python connector to a MIP solver to pass the problem formulation and retrieve the solution
- Add additional constraints if necessary and resolve (cutting planes method)
- Provide optimal objective function value
- Save optimal solution $x_{i,j}$ and L_i (if applicable) decision variables to a .csv file

Utilizing Python

We decided to write our program in Python (an open-source programming language with a similar syntax to MATLAB – see reference [7]). This allowed us to read in our data set, dynamically generate the problem formulation, pass it to a solver, and plot the resulting output.

Solvers Used

We attempted several solvers in obtaining the optimal solution:

- Cbc: coin-or-branch and cut: part of the Coin-Or project (see reference [8])
- GLPK: GNU Linear Programming Kit (see reference [9])
- Gurobi: commercial solver with academic license available (see reference [10])
- SCIP Optimization Suite (see reference [11])
- Held-Karp: a python implementation of the Held-Karp dynamic programming algorithm. We initially attempted this solver using publicly available Python code (see reference [16]), but due to poor performance, we did not include it in our analysis

These solvers had to be installed as separate applications on the computer that we ran our solver. In order to pass the problem formulation from Python to these programs, use utilized Python connectors (see next section).

Connectors

The solvers we attempted to use were written in C/C++ (for high performance) and were provided in native assembly format (executable). In order to pass the problem formulation from our Python program to these solvers, we utilized a Python connector. We used the following:

- PuLP: a flexible connector that works with multiple solvers (Cbc, Gurobi, GLPK). See [12] for more details on PuLP.
- PySCIPOpt: connector made for the SCIP Optimization Suite, with support for cutting planes method (see [13]).

Programmatic Problem Formulation

The connector libraries above have a slightly different syntax for defining the problem model. We provide a brief summary of how the problem is formulated using these connectors:

Table 5: Formulation for the PuLP Solver

Imports	`import pulp`
Define model	`z = pulp.LpProblem('EngM540', pulp.LpMinimize)`
Decision variables	`x[i,j] = pulp.LpVariable('x_' + str(i) + '_' + str(j), 0, 1, pulp.LpInteger)` `l[i] = pulp.LpVariable('l_' + str(i), 1, min(n,c), pulp.LpInteger)`
Objective function	`z += pulp.lpSum([d[i][j] * x[i,j] for i in range(n+w) for j in list(range(i)) + list(range(i+1,n+w))])`

Constraints	`for i in range(w,n+w):`
	`  z += pulp.lpSum([x[i,j] for j in range(n+w) if i != j]) == 1`
	`  z += pulp.lpSum([x[j,i] for j in range(n+w) if i != j]) == 1`
	`for i in range(w, n+w):`
	`  for j in range(w, n+w):`
	`  if i!=j:`
	`  z += pulp.lpSum([l[i], -1*l[j], n*x[i,j], -n+1]) <= 0`
	`z += pulp.lpSum([x[i,j] for i in range(w) for j in range(w,n+w) if i != j]) == paths`

Solving	`status = z.solve() # Coin-OR CBC`
	`status = z.solve(pulp.GLPK()) # GLPK`
	`status = z.solve(pulp.GUROBI_CMD()) # Gurobi commercial solver`

Accessing Solution	`pulp.value(x[i,j])`
	`pulp.value(l[i])`
	`z.objective.value()`

Our code for PySCIPOpt is based on the example code provided with this library [14].

Table 6: Formulation for the PySCIOpt

Imports	`import pyscipopt`
Define model	`model = Model("EngM540")`
Decision variables	`x[i,j] = model.addVar(ub=1, name="x(%s,%s)"%(i,j))` `l[i] = model.addVar(ub=min(c,n),lb=1, name="l(%s)"%(i))`
Objective function	`model.setObjective(quicksum(d[i,j]*x[i,j] for (i,j) in x), "minimize")`

Constraints	`for i in range(w, n+w):`
	`  model.addCons(quicksum(x[j,i] for j in range(n+w) if j != i) == 1, "In(%s)"%i)`
	`  model.addCons(quicksum(x[i,j] for j in range(n+w) if j != i) == 1, "Out(%s)"%i)`
	`for i in range(w, n+w):`
	`  for j in range(w, n+w):`
	`  if i!=j:`
	`  model.addCons(l[i] -l[j] +n*x[i,j] <= n-1, "Li(%s,%s)"%(i,j))z +=`
	`pulp.lpSum([x[i,j] for i in range(w) for j in range(w,n+w) if i != j]) == paths`

Solving	`model.optimize()`

Cutting Planes 1 *one-time*	`model.freeTransform()`
	`model.chgVarType(x[i,j], "B") # binary`
	`model.chgVarType(l[i], "I") # integer`
	`model.optimize()`

Cutting Planes 2 *repeat*	`model.freeTransform()`
	`model.addCons(quicksum(x[i,j] for i in S for j in S if j!=i) <= len(S)-1)`
	`model.optimize()`

Accessing Solution	`model.getVal(x[i,j])`
	`model.getVal(l[i])`
	`model.getObjVal()`

When solving case (1) of the formulation, the cutting planes code above is used to eliminate subtours as they come up in the solution, one at a time, until no more subtours remain.

In addition, the Gurobi solver (with PuLP connector) also implements cutting planes under-the-hood, but there is no control over that process, unlike when using PySCIPOpt. The full code for the program we wrote is provided in Appendix 8.

Comparison

The solvers we used did not perform the same, and not all of them were able to find an optimal solution for our base case (50 customers, a single truck & warehouse, and a truck capacity of 50). Generally, the cbc and GLPK solvers performed the worst and did not scale for problem sizes containing 50 customers:

- Coin-or-branch and cut (cbc): the default solver utilized by the PuLP connector. This solver took unreasonably long for problem instances containing more than 25 customers
- GLPK: this solver performed faster than cbc for instances of the problem containing ~ 10 customers or less, but was significantly slower for larger problem instances

The cutting planes method along with the SCIP solver (utilized via the PySCIPOpt connector), and the Gurobi solver (utilized via the PuLP connector) performed the best:

- Both SCIP and Gurobi were able to find solutions for problem sizes containing 50 customers
- Gurobi usually found a solution faster than SCIP

Solution Visualization

A simple GUI was developed using the matplotlib library (see [15]), which allowed us to quickly run several different scenarios and examine the effect on the optimal solution. In some cases, we were performing trial and error to determine which permutations of inputs were solvable using the algorithm selected (some would run for over a few hours, with no way of the % progress completed as far). The GUI is shown in figure 4 below:

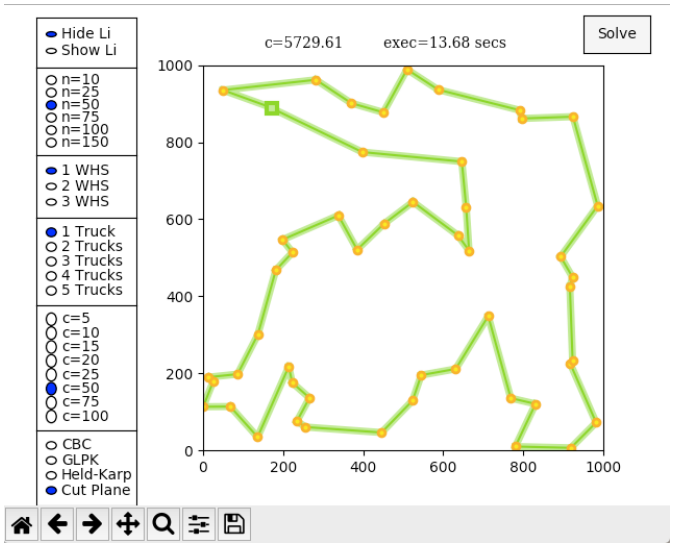


Figure 4 – Graphical user interface for the solver

The program initially solved the problem on a randomly generated dataset of the size selected in the interface but was later modified to import the dataset.csv file.

Solutions

Approximate Distance Matrix (GPS)

We first solved the problem by using the GPS coordinates of every location to compute the straight-line distance, in km. This gave us the following solution:

Parameters	$n_c = 50$ $n_w = 1$ $t = 1$ $c = 50$
Optimal route distance	**128.089 km**
Optimal route cost	**$ 146.73**

Here, we utilized the symmetric version of the problem formulation, as this approach always yields a symmetric distance matrix. This produces the following solution:

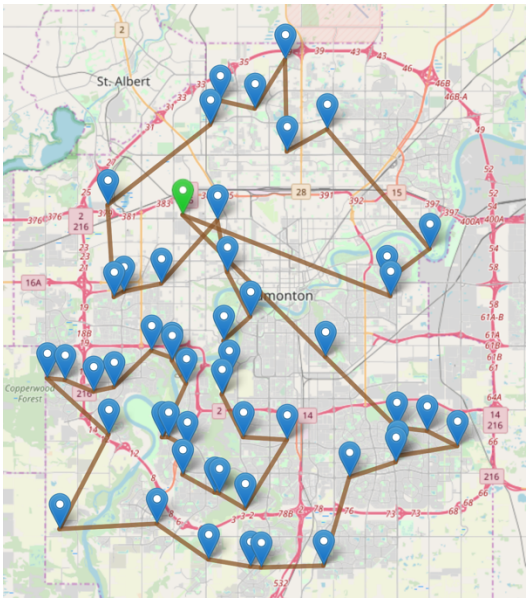


Figure 5 – Problem solution using approximation of D, n_c=50, c=50, t=1, wh=1

Accurate Distance Matrix (Google Maps)

Then, we solved the problem using a distance matrix that we retrieved from Google Maps (see 'Dataset' section for more details). The result is provided below:

Table 7: Google Maps Distance Matrix

Parameters	$n_c = 50$ $n_w = 1$ $t = 1$ $c = 50$
Optimal route distance	**220.24 km**
Optimal route cost	**$ 252.28**

This yields the following path:

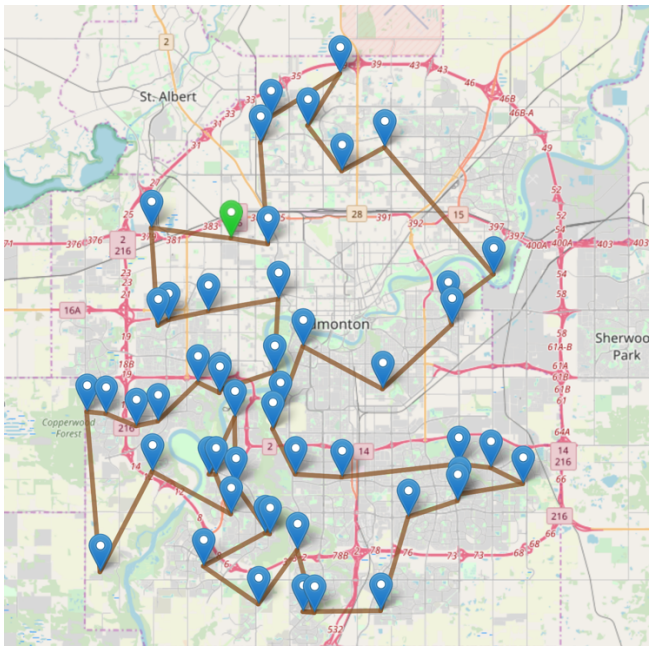


Figure 6 – Problem solution using Google Maps D, n_c=50, c=50, t=1, wh=1

Sensitivity Analysis

In this section, we solve the problem formulation for different combinations of inputs and compare the objective function values to see how sensitive the objective function is to the input variables.

Capacity = 30

The solution where the truck capacity is reduced from 50 to 30 is provided below. This implication of this change is that the truck now has to take two paths, returning to the warehouse a single time to pick up more product. Note that our formulation will allow both paths to have a length between 20 and 30, depending on what results in the shortest route.

Table 8: Optimal route distance and cost for capacity = 30

Parameters	$n_c = 50$ $n_w = 1$ $t = 1$ $c = 30$
Optimal route distance	**226.147 km**
Optimal route cost	**$ 250.570**

Compared to the base case (where c=50), we see that:

The optimal route distance increases by **5.154 km** (which is expected, as now 2 trips are required instead of one), and the optimal route cost decreased by **$ 1.71** (due to the lower fuel cost c_f for c=30 compared to c=50). The optimal route is shown is figure 7.

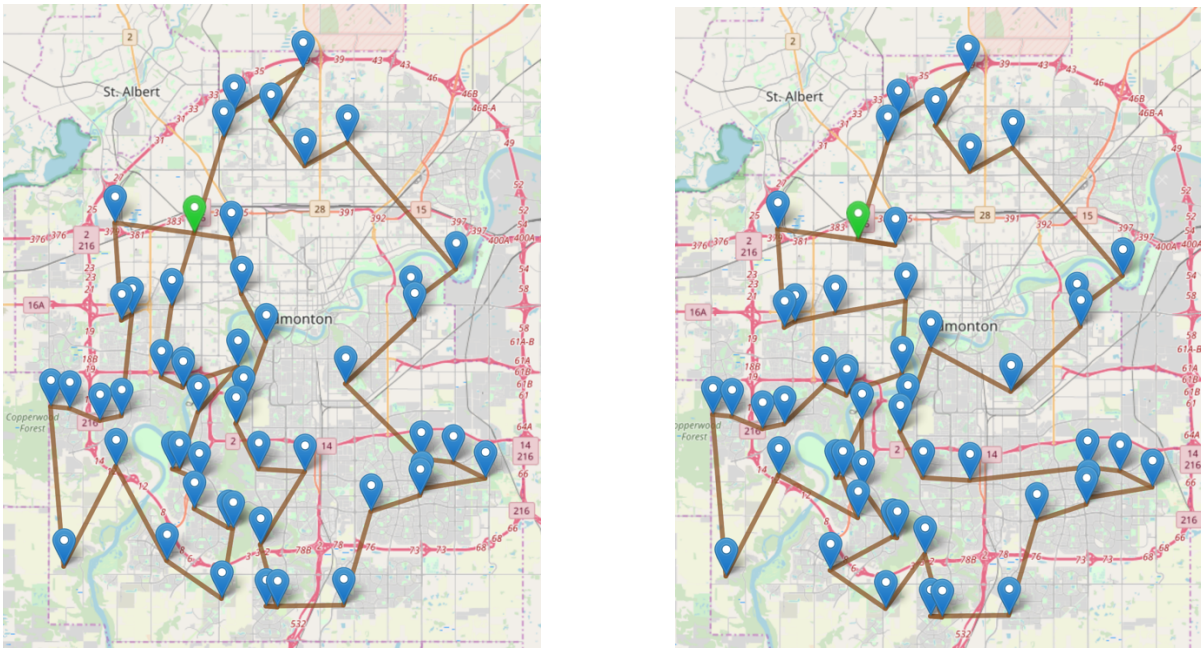


Figure 7 – Problem solution using Google Maps D, LEFT: n_c=50, c=30, t=1, wh=1; RIGHT: n_c=50, c=50, t=2, wh=1

Trucks = 2

Next, the problem is solved for the case where t = 2. This implies that two paths must be utilized, but without any limitation placed on the lengths of the paths. So, the length of each path could be anywhere between 1 and 49.

Table 9: Optimal route distance and cost with two trucks

Parameters	$n_c = 50$ $n_w = 1$ $t = 2$ $c = 50$
Optimal route distance	**222.469 km**
Optimal route cost	**$ 254.838**

We see that this solution is better than c=30, but worse than the base case of t = 1 & c = 50. This is expected, because this version of the problem is a less constrained version of the case where c=30.

Compared to the base case, the optimal route distance is **2.233 km** longer, while the cost is **$ 2.59** higher. See figure 7 above for a visualization. Here, one route contains a single customer, while the other route contains 49 customers.

Warehouses = 2

Next, the problem is solved when there are two warehouses instead of one. The formulation gives more flexibility to the route, allowing it start/end at either warehouse to be consistent with the constraints. We obtained the following solution:

Table 10: Optimal route distance and cost for having two warehouses

Parameters	$n_c = 50$ $n_w = 2$ $t = 1$ $c = 50$
Optimal route distance	**219.787 km**
Optimal route cost	**$ 251.766**

As expected, the optimal distance and cost are better than the base case with w = 1, as this is a less constrained version of the problem.

This solution is **0.449 km shorter** than the route with n_w = 1, and **$0.514 cheaper**. This is shown in figure 8. The solution is interesting, as the optimal path begins at one of the warehouses, and ends at the other.

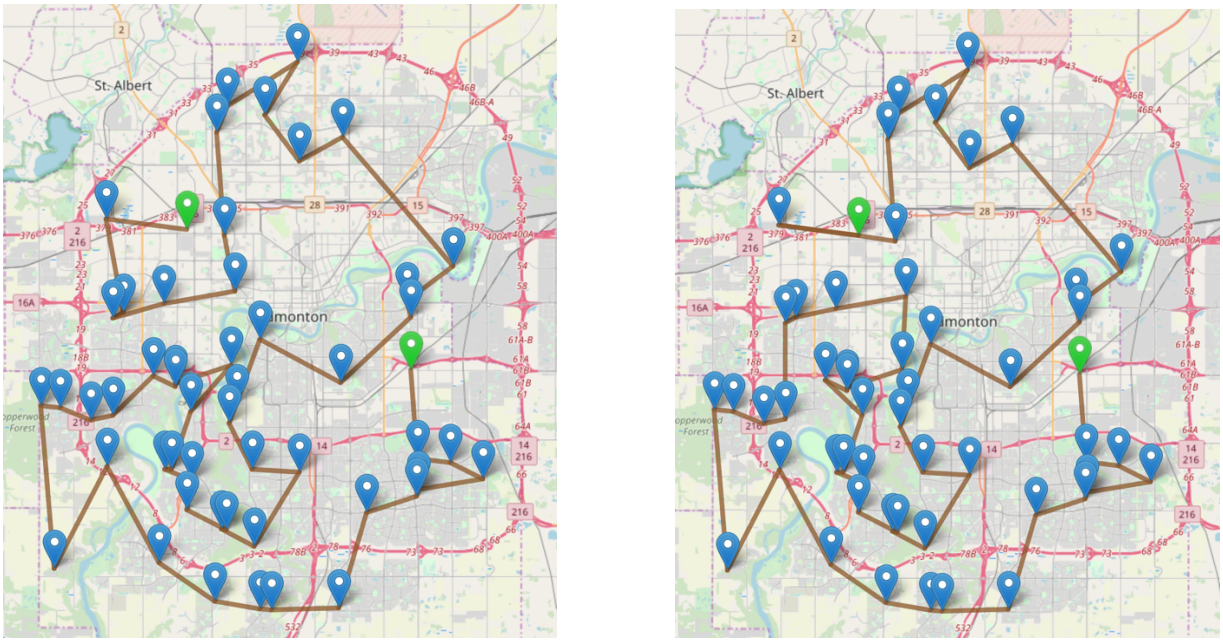


Figure 8 – Problem solution using Google Maps D, LEFT: n_c=50, c=50, t=1, wh=2; RIGHT: n_c=50, c=50, t=2, wh=2

Warehouses = 2, t = 2

In this case, we require exactly 2 paths, to ensure that two trucks are used, but leave the solver to determine the length of each route taken by the truck (from 1 to 49). We obtain the following solution:

Table 11: Optimal route distance and cost for two warehouses and two trucks

Parameters	$n_c = 50$ $n_w = 2$ $t = 2$ $c = 50$
Optimal route distance	**221.103 km**
Optimal route cost	**$ 253.273**

Compared to the version with w=2 and t=1, this route is **1.316 km** longer and **$1.507** more expensive. See figure 8 for a visual view of the paths taken. The optimal routes involve one of the trucks going to a single stop, while the other truck has a route with 49 customers. This may not be ideal, in terms of labour scheduling, and we present the case with a limit on a single truck's capacity of 30 next.

Warehouses = 2, c = 30

In this case, we also require exactly 2 paths, to ensure that two trucks are used, but place a stricter constraint on the length of each route (from 20 to 30). We obtain the following solution:

Table 12: Optimal route distance and cost for two warehouses and capacity = 30

Parameters	$n_c = 50$ $n_w = 2$ $t = 1$ $c = 30$
Optimal route distance	**222.163 km**
Optimal route cost	**$ 246.16**

Compared to the version with w=2, t=1 and c=50, this route is **2.376 km** longer and **$ 5.61**

cheaper. It is also longer and less expensive than the case with w = 2, t = 2 and c = 50. See figure 9 below for a visualization of this solution:

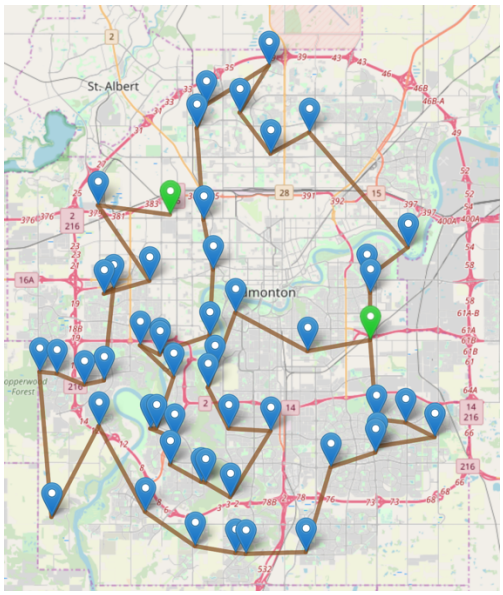


Figure 9 – Problem solution using Google Maps D, with n_c=50, c=30, t=1, wh=2

Here, we see that one of the paths begins and ends at warehouse #2, while the other utilizes both warehouses.

Summary

A summary of the solutions we obtained is provided below:

Table 13: Summary of optimal route distances and costs for all scenarios under consideration

n	w	t	c	Optimal route distance (km)	Optimal route cost ($)
50	1	1	50	220.240	252.280
50	1	2	50	222.469	254.838
50	1	1	30	226.147	250.570
50	2	1	50	219.787	251.766
50	2	2	50	221.103	253.273
50	2	1	30	222.163	246.156

The results are somewhat surprising. Due to having a fairly large number of customers (50), the impact of adding an additional warehouse on the objective function value are insignificant (around 0.39% reduction in the optimal route cost). We see a similar situation for the truck capacity and number of trucks. Reducing truck capacity to 30 has the largest impact on

the optimal objective function value (decreasing it by around 0.68% - while the optimal distance is slightly higher, the optimal cost is actually lower, as the increase in distance is offset by the reduction in fuel cost due to having a lighter load).

This result logically makes sense. Due to the large number of customers per route (50 or 30), and the fact that the warehouse is fairly close to some customers (and not outside the city), the additional trip to the warehouse required for the case of n_c=30 or t=2, as it can be done in an optimal way (ie. between a pair of customers). We would expect, though, that as the number of customers per trip is decreased (say to 25 or 20), we would see a greater impact resulting from an additional warehouse.

Conclusions

Integer programming provides a valuable management tools for making complicated decisions. In this report, we explored the optimization problem associated with finding an optimal delivery route, which revolved around find a sequence of visiting customers that minimizes the total distance travelled. We expressed our objective function as a cost minimization problem by making an assumption about how labour and fuel costs are related to the distance travelled.

We found that the way we formulated our problem and the solver we used had a huge impact on our ability to find an optimal solution. For instance, a simplified version of the formulation can be used if the distance matrix is symmetric, and if only a single path is required to fulfil customer demand (if a single truck has sufficient capacity for all products). We found that the only solvers that were able to reliably find an optimal solution for an asymmetric distance matrix, with more than one path were the SCIP and Gurobi solvers.

We expressed our solution as an optimal distance and cost, and compared it to several scenarios which used (i) a smaller truck with lower truck capacity, (ii) an additional truck, and (iii) an additional warehouse. We found that lowering truck capacity increased the total distance travelled, but decreased the total cost incurred to make the deliveries (due to the lower fuel cost associated with a small truck).

As we've shown, for a business where daily operating expenses are composed primarily of logistics costs, this type of analysis is invaluable: it can help the business to better manage its delivery expenses, and understand the monetary impact of strategic business decisions (such as truck size, number of warehouses, and number of trucks).

Some areas that could further be improved upon in our analysis are:

- Utilizing a time matrix in addition to a distance matrix, which would find an optimal route which minimizes trip time (these are provided by MapBox and Google Maps, in addition to the distance matrix D that we utilized)
- Incorporate truck rental costs into the objective function
- Explore overtime cost of labour for long routes

References

[1] https://developers.google.com/maps/documentation/distance-matrix/

[2] https://data.edmonton.ca/City-Administration/Property-Assessment-Data-2012-2017-/qi6a-xuwt/data

[3] https://leafletjs.com/

[4] https://wiki.openstreetmap.org/wiki/Standard_tile_layer

[5] https://www.mapbox.com/help/define-matrix-api/

[6] https://developers.google.com/maps/documentation/distance-matrix/start

[7] https://www.python.org/

[8] https://projects.coin-or.org/Cbc

[9] https://www.gnu.org/software/glpk/

[10] http://www.gurobi.com/

[11] https://scip.zib.de/

[12] https://pythonhosted.org/PuLP/

[13] https://github.com/SCIP-Interfaces/PySCIPOpt

[14] https://github.com/SCIP-Interfaces/PySCIPOpt/tree/master/examples/finished

[15] http://www.math.uwaterloo.ca/tsp/methods/dfj/index.html

[16] https://github.com/CarlEkerot/held-karp

Appendix 1: Dataset

Location	Address	lat	lng
Warehouse 1	11921 152 St NW EDMONTON AB T5V 1E3	53.572226	-113.58381
Warehouse 2	5810 76 Ave NW EDMONTON AB T6B 0A6	53.512735	-113.423675
Customer 1	1912 TOMLINSON WAY NW EDMONTON AB T6R 2R5	53.45332914	-113.5832276
Customer 2	4922 27A AVENUE NW EDMONTON AB T6L 6B3	53.45903088	-113.4195745
Customer 3	13121 104 AVENUE NW EDMONTON AB	53.5459037	-113.5493732
Customer 4	18219 98A AVENUE NW EDMONTON AB T5T 3L5	53.534394	-113.6363747
Customer 5	11716 135A STREET NW EDMONTON AB T5M 1L5	53.56959423	-113.5568043
Customer 6	4416 MCCLUNG COURT NW EDMONTON AB T6R0M9	53.44479748	-113.5553688
Customer 7	9011 23 AVENUE SW EDMONTON AB T6X0Z9	53.41155502	-113.4750256
Customer 8	3528 105B STREET NW EDMONTON AB T6J 2L1	53.46937759	-113.5031153
Customer 9	747 HETU LANE NW EDMONTON AB T6R 2W9	53.4703923	-113.59925
Customer 10	6007 101A AVENUE NW EDMONTON AB T6A 0L9	53.54195069	-113.4261944
Customer 11	7020 95 STREET NW EDMONTON AB T6E 3E5	53.5072696	-113.4736968
Customer 12	18019 105A STREET NW EDMONTON AB T5X6J8	53.64312767	-113.5045585
Customer 13	14923 87 STREET NW EDMONTON AB T5E 5T4	53.61122946	-113.472363
Customer 14	6220 127 STREET NW EDMONTON AB T6H 3W9	53.49884402	-113.5474969
Customer 15	821 FORBES CLOSE NW EDMONTON AB T6R 2P4	53.4702743	-113.5944371
Customer 16	4906 31 AVENUE NW EDMONTON AB T6L 5H6	53.46191648	-113.4179429
Customer 17	14020 151 AVENUE NW EDMONTON AB T6V1T6	53.61308148	-113.5622736
Customer 18	16255 135 STREET NW EDMONTON AB T6V 0G3	53.624123	-113.5549965
Customer 19	815 WILDWOOD CRESCENT NW EDMONTON AB T6T0M2	53.46654863	-113.3717636
Customer 20	16126 100A AVENUE NW EDMONTON AB	53.54055386	-113.6001564
Customer 21	8928 116 STREET NW EDMONTON AB T6G 1P8	53.5256586	-113.531231
Customer 22	828 CHAHLEY WAY NW EDMONTON AB T6M0C7	53.4718784	-113.6405512
Customer 23	11939 20 AVENUE SW EDMONTON AB T6W 0E1	53.41135954	-113.5311719
Customer 24	21330 61 AVENUE NW EDMONTON AB T6M0K1	53.49736349	-113.6879737
Customer 25	12608 52B AVENUE NW EDMONTON AB T6H 0R4	53.49015719	-113.5535897
Customer 26	3523 38A AVENUE NW EDMONTON AB T6L 6N9	53.47320515	-113.3952815
Customer 27	5808 181 STREET NW EDMONTON AB T6M 1V7	53.49346792	-113.6362668
Customer 28	2218 STAN WATERS AVENUE NW EDMONTON AB T5E5Y8	53.60113978	-113.5031571
Customer 29	2003 AINSLIE LINK SW EDMONTON AB T6W 2M1	53.43066709	-113.6033058
Customer 30	7328 157 STREET NW EDMONTON AB T5R 1Z9	53.50564961	-113.5920701
Customer 31	103 CARLSON CLOSE NW EDMONTON AB T6R 2J8	53.465949	-113.5799035
Customer 32	525 CALLAGHAN POINTE SW EDMONTON AB T6W0G4	53.4107573	-113.5228052
Customer 33	3134 PAISLEY ROAD SW EDMONTON AB	53.41437039	-113.5639131
Customer 34	39 PATRICIA CRESCENT NW EDMONTON AB T5R 5N7	53.51022434	-113.6077606
Customer 35	925 GOODWIN WAY NW EDMONTON AB T5T6X8	53.496531	-113.674332
Customer 36	15104 57 AVENUE NW EDMONTON AB T6H 5B9	53.49497049	-113.5806122
Customer 37	911 117 STREET NW EDMONTON AB T6J 6Z7	53.43789767	-113.5351194
Customer 38	15820 119 STREET NW EDMONTON AB T5X 2P5	53.62070042	-113.5277118
Customer 39	20721 5 AVENUE SW EDMONTON AB T6M 2P4	53.42815457	-113.678483
Customer 40	8012 22 AVENUE NW EDMONTON AB T6K 1Z3	53.45194556	-113.4554235
Customer 41	17916 99A AVENUE NW EDMONTON AB T5T 3R1	53.5372012	-113.6291644
Customer 42	13312 79 AVENUE NW EDMONTON AB T5R 3G5	53.51450263	-113.5517854
Customer 43	3616 117B STREET NW EDMONTON AB T6J 1W2	53.47057706	-113.5366202
Customer 44	5708 95 AVENUE NW EDMONTON AB T6B 1A5	53.53500161	-113.4232647
Customer 45	18445 122 AVENUE NW EDMONTON AB T5V 1R4	53.576407	-113.6409895
Customer 46	3211 103 AVENUE NW EDMONTON AB T5W 0A4	53.55671498	-113.3930178
Customer 47	19011 56 AVENUE NW EDMONTON AB T6M 2L4	53.4913377	-113.6519364
Customer 48	7412 157 STREET NW EDMONTON AB T5R 1Z9	53.50701963	-113.5919666
Customer 49	3909 MACNEIL BAY NW EDMONTON AB T6R0H5	53.4449942	-113.558756
Customer 50	5309 39B AVENUE NW EDMONTON AB T6L 1R9	53.4747778	-113.4188405

Appendix 2: Location Visualization Code

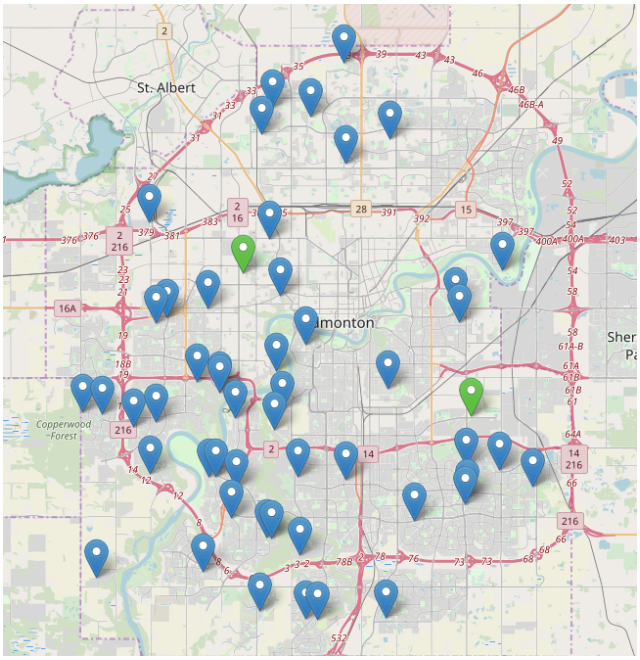
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.4/dist/leaflet.css" />
   <style>
#mapid { height: 700px; }
</style>
</head>
<body>
<div id="mapid"></div>
<script src="https://unpkg.com/leaflet@1.3.4/dist/leaflet.js"></script>
<script>
    map = new L.Map('mapid');
    var osmUrl='https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';
    var osmAttrib='Map data © <a href="https://openstreetmap.org">OpenStreetMap</a> contributors';
    var osm = new L.TileLayer(osmUrl, {minZoom: 8, maxZoom: 12, attribution: osmAttrib});

    map.setView(new L.LatLng(53.51670635,-113.372475),11);
    map.addLayer(osm);

    var whIcon = L.icon({ iconUrl: 'marker-icon-wh.png', shadowUrl: 'marker-shadow.png' });

    // warehouses
    L.marker([53.512735,-113.423675], {icon: whIcon}).addTo(map);
    L.marker([53.572226,-113.583810], {icon: whIcon}).addTo(map);

    // customers
    L.marker([53.45332914,-113.5832276]).addTo(map);
    L.marker([53.45903088,-113.4195745]).addTo(map);
    L.marker([53.5459037,-113.5493732]).addTo(map);
    L.marker([53.534394,-113.6363747]).addTo(map);
    L.marker([53.56959423,-113.5568043]).addTo(map);
    L.marker([53.44479748,-113.5553688]).addTo(map);
    L.marker([53.41155502,-113.4750256]).addTo(map);
    L.marker([53.46937759,-113.5031153]).addTo(map);
    L.marker([53.4703923,-113.59925]).addTo(map);
    L.marker([53.54195069,-113.4261944]).addTo(map);
    L.marker([53.5072696,-113.4736968]).addTo(map);
    L.marker([53.64312767,-113.5045585]).addTo(map);
    L.marker([53.61122946,-113.472363]).addTo(map);
    L.marker([53.49884402,-113.5474969]).addTo(map);
    L.marker([53.4702743,-113.5944371]).addTo(map);
    L.marker([53.46191648,-113.4179429]).addTo(map);
    L.marker([53.61308148,-113.5622736]).addTo(map);
    L.marker([53.624123,-113.5549965]).addTo(map);
    L.marker([53.46654863,-113.3717636]).addTo(map);
    L.marker([53.54055386,-113.6001564]).addTo(map);
    L.marker([53.5256586,-113.531231]).addTo(map);
    L.marker([53.4718784,-113.6405512]).addTo(map);
    L.marker([53.41135954,-113.5311719]).addTo(map);
    L.marker([53.49736349,-113.6879737]).addTo(map);
    L.marker([53.49015719,-113.5535897]).addTo(map);
    L.marker([53.47320515,-113.3952815]).addTo(map);
    L.marker([53.49346792,-113.6362668]).addTo(map);
    L.marker([53.60113978,-113.5031571]).addTo(map);
    L.marker([53.43066709,-113.6033058]).addTo(map);
    L.marker([53.50564961,-113.5920701]).addTo(map);
    L.marker([53.465949,-113.5799035]).addTo(map);
    L.marker([53.4107573,-113.5228052]).addTo(map);
    L.marker([53.41437039,-113.5639131]).addTo(map);
    L.marker([53.51022434,-113.6077606]).addTo(map);
    L.marker([53.496531,-113.674332]).addTo(map);
    L.marker([53.49497049,-113.5806122]).addTo(map);
    L.marker([53.43789767,-113.5351194]).addTo(map);
    L.marker([53.62070042,-113.5277118]).addTo(map);
    L.marker([53.42815457,-113.678483]).addTo(map);
    L.marker([53.45194556,-113.4554235]).addTo(map);
    L.marker([53.5372012,-113.6291644]).addTo(map);
    L.marker([53.51450263,-113.5517854]).addTo(map);
    L.marker([53.47057706,-113.5366202]).addTo(map);
    L.marker([53.53500161,-113.4232647]).addTo(map);
    L.marker([53.576407,-113.6409895]).addTo(map);
    L.marker([53.55671498,-113.3930178]).addTo(map);
    L.marker([53.4913377,-113.6519364]).addTo(map);
    L.marker([53.50701963,-113.5919666]).addTo(map);
    L.marker([53.4449942,-113.558756]).addTo(map);
    L.marker([53.4747778,-113.4188405]).addTo(map);
   </script>
</body>
</html>
```



Appendix 3: Approximated Distance Matrix via Euclidean Distance (m)

	Warehouse 2	Warehouse 1	Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Customer 6	Customer 7	Customer 8	Customer 9	Customer 10	Customer 11
Warehouse 2	0	12469.2	13147	16578.3	3695.46	5439.08	1808.77	14215.5	19171.3	12564	11306.4	10947	10229.9
Warehouse 1	12469.2	0	12444.1	5944.49	9090.16	14278.5	10820.6	11513	11694	7117.98	12535.3	3234.79	3366.14
Customer 1	13147	12444.1	0	10865.9	10478.9	9629.93	12974.3	2073.89	8534.2	5598.22	2165.04	14288.3	9392.77
Customer 2	16578.3	5944.49	10865.9	0	12889.3	16599.9	15230.7	9138.45	6409.45	5653.35	11973.3	9179.25	6426.85
Customer 3	3695.46	9090.16	10478.9	12889.3	0	5894.44	2665.25	11186.8	15650.7	8998.92	8978.8	8159.63	6582.37
Customer 4	5439.08	14278.5	9629.93	16599.9	5894.44	0	6545.36	11266.5	17285.3	11381.4	7491.67	13929.8	11176.5
Customer 5	1808.77	10820.6	12974.3	15230.7	2665.25	6545.36	0	13799.6	18295.1	11637.1	11323.2	9161.97	8815.94
Customer 6	14215.5	11513	2073.89	9138.45	11186.8	11266.5	13799.6	0	6473.74	4402.61	4058.22	13732.7	8774.78
Customer 7	19171.3	11694	8534.2	6409.45	15650.7	17285.3	18295.1	6473.74	0	6659.45	10496.4	14776.9	10583.9
Customer 8	12564	7117.98	5598.22	5653.35	8998.92	11381.4	11637.1	4402.61	6659.45	0	6371.13	9504.28	4620.79
Customer 9	11306.4	12535.3	2165.04	11973.3	8978.8	7491.67	11323.2	4058.22	10496.4	6371.13	0	13924	9261.73
Customer 10	10947	3234.79	14288.3	9179.25	8159.63	13929.8	9161.97	13732.7	14776.9	9504.28	13924	0	4958.63
Customer 11	10229.9	3366.14	9392.77	6426.85	6582.37	11176.5	8815.94	8774.78	10583.9	4620.79	9261.73	4958.63	0
Customer 12	9426.68	15377.2	21622.2	21118	11150.7	14846.5	8832.86	22186.2	25680.4	19212.5	20100.2	12327.6	15160.2
Customer 13	8533.58	11356.7	18938	17187.8	8836.45	13773.9	7233.57	19205.2	22079.5	15816.5	17691	8245.86	11495.6
Customer 14	8462.34	8340.48	5561.71	9549.89	5205.06	7075.26	7847.33	5998.86	10781.3	4388.48	4652.89	9336.61	4974.39
Customer 15	11295.1	12245.2	2015.6	11654.8	8878.87	7614.34	11261.1	3826.43	10239.6	6052.05	319.179	13670.6	8982.55
Customer 16	16410.5	5632.02	10996.4	336.899	12726.7	16536	15041.4	9304.34	6733.45	5704.29	12051.5	8866.55	6228.01
Customer 17	4736.35	14390.6	17718.8	19475.3	7476.89	9984.75	4822.12	18613.4	23019.9	16364.7	15966.2	11948	13084.1
Customer 18	6045.92	15068.8	18977.5	20334.5	8657	11285.7	6030.65	19828.8	24093.1	17451.5	17248.8	12450.4	13994.3
Customer 19	18262.2	6156.52	14091.5	3275.83	14671.9	19059.7	16728	12405.6	9157.71	8709.73	15080.4	9083.03	8115.36
Customer 20	3665.14	12076.8	9709.72	14974.6	3410.9	2491.03	4304.66	10995.8	16498.1	10159.7	7758.28	11508.8	9142.39
Customer 21	6213.45	7260.68	8707.71	10438.4	2540.06	7023.96	5144.17	9082.97	13155.1	6495.75	7591.56	7179.22	4317.03
Customer 22	11714.1	15056.9	4317.38	14712.8	10170.2	6918.13	12143.5	6390.89	12843.4	9110.91	2741.58	16168.9	11723.5
Customer 23	18126	13281.9	5784.08	9086.11	14925.9	15286.2	17578.8	4030.67	3725.59	6679.6	7936.33	16027.6	11268.6
Customer 24	10771.1	17581	8477.58	18279.6	10627.7	5332.04	11794.4	10535	17008.1	12630.3	6590.51	18013.7	14229.9
Customer 25	9292.37	8957.05	4521.01	9522.94	6170.43	7345.57	8786.25	5016.99	10132.4	4057.13	3731.75	10193.3	5618.53
Customer 26	16599.9	4758.29	12648.2	2246.8	12987.6	17337.4	15095.4	11065.7	8627.14	7157.65	13518.5	7872.15	6415.62
Customer 27	9374.75	14236.2	5661.13	14852.9	8166.5	4525.37	9924.7	7595.97	14010.2	9214.02	3538.85	14902	10872.2
Customer 28	6214.96	11098.7	17181.4	16658.4	6829.07	11490.5	4973.42	17629.3	21045.7	14569.5	15793.4	8289.58	10560.9
Customer 29	15705.9	14967.1	2837.53	12579	13233.4	11676.8	15667.3	3542.07	8768.43	7901.76	4400.8	17001.1	12062.6
Customer 30	7381.87	11175.5	5814.87	12535.6	5272.12	4324.39	7445.59	7154.56	12978.2	7127.48	3927.44	11688.3	7839.02
Customer 31	11754.3	11568.6	1412.71	10653.2	9069.03	8441.58	11562	2848.66	9194.63	5102.56	1372.92	13198.3	8388.3
Customer 32	18305.7	13050.4	6182.04	8680.99	15047	15605.2	17707	4339.65	3171.61	6612.07	8317.21	15855.9	11157.1
Customer 33	17504.4	14306.5	4494.23	10770.6	14576	14113.2	17170.2	3411.84	5906.03	7297.02	6622.87	16800.4	11885.7
Customer 34	7036.46	12189.2	6497.65	13690.5	5521.96	3275.45	7379.66	8024.09	14015.2	8272.6	4440.31	12516.8	8881.3
Customer 35	10292	16692.1	7697.68	17379.7	9909.53	4882.68	11213.2	9739.58	16212.1	11732.5	5752.37	17173.9	13337.6
Customer 36	8545.07	10574.8	4607.71	11385.1	5999.4	5711.99	8400.46	5794.51	11578.6	5861.99	2985	11464.9	7208.73
Customer 37	15199	11090.3	3617.33	8008.68	11979.9	12602.8	14632.8	1544.3	4937.04	4076.41	5566.1	13580.3	8683.5
Customer 38	6516.52	13777.8	18867.8	19254.7	8393.54	11943.8	5968.94	19536.2	23387.5	16811.3	17280.6	10992.5	13041
Customer 39	17119.6	19301.2	6902.41	17502.8	15577.3	12074	17593	8369.13	13621.8	12487.7	7028.79	20918.9	16145.7
Customer 40	15783.4	7043.07	8473.47	2502.08	12109.5	15057.3	14638.1	6672.98	4651.51	3702.23	9748.08	10138.7	6236.03
Customer 41	4898.43	13865.1	9760.26	16350.3	5365.25	569.13	5977.37	11326.2	17242.9	11220.3	7648.23	13437.5	10808.2
Customer 42	6725.05	8482.44	7077.59	10691.6	3475.81	6014.5	6100.74	7711.23	12468.3	5940.25	5802.66	8847.42	5230.87
Customer 43	11665.6	8813.84	3630.24	7861.06	8371.85	9665.36	11029.9	3109.65	7698.91	2224.08	4150.01	10757.9	5816.37
Customer 44	11387.1	2462.26	13920.4	8403.94	8429.04	14099.8	9624.26	13267.9	14074.2	8978.16	13668	792.454	4532.49
Customer 45	3807.71	16006.1	14136.1	19575.9	6933.38	4655.56	5615.53	15617.4	21285.5	14943.8	12043.5	14705.1	13450.3
Customer 46	12730.3	5269.09	17008.2	10943.5	10409.5	16284.8	10922.3	16392.3	16945.2	12098.6	16657.4	2734.95	7641.23
Customer 47	10015.8	15297.8	6195.89	15804	9082.14	4871.08	10700.5	8211.65	14675.3	10153.5	4188.76	15955.5	11933.2
Customer 48	7230.29	11158.9	5964.93	12592.3	5141.08	4219.03	7299.69	7294.91	13095.9	7208.14	4078.66	11630.5	7830.04
Customer 49	14166	11670	1865.85	9356.06	11175.3	11142.1	13778.1	225.628	6671.92	4568.45	3884.7	13856.7	8898.06
Customer 50	15339.4	4209.27	11149.4	1741.88	11684.3	15839.5	13902.8	9637	7921.44	5615.82	11963.5	7443.51	5109.54

	Customer 12	Customer 13	Customer 14	Customer 15	Customer 16	Customer 17	Customer 18	Customer 19	Customer 20	Customer 21	Customer 22	Customer 23	Customer 24
Warehouse 2	9426.68	8533.58	8462.34	11295.1	16410.5	4736.35	6045.92	18262.2	3665.14	6213.45	11714.1	18126	10771.1
Warehouse 1	15377.2	11356.7	8340.48	12245.2	5632.02	14390.6	15068.8	6156.52	12076.8	7260.68	15056.9	13281.9	17581
Customer 1	21622.2	18938	5561.71	2015.6	10996.4	17718.8	18977.5	14091.5	9709.72	8707.71	4317.38	5784.08	8477.58
Customer 2	21118	17187.8	9549.89	11654.8	336.899	19475.3	20334.5	3275.83	14974.6	10438.4	14712.8	9086.11	18279.6
Customer 3	11150.7	8836.45	5205.06	8878.87	12726.7	7476.89	8657	14671.9	3410.9	2540.06	10170.2	14925.9	10627.7
Customer 4	14846.5	13773.9	7075.26	7614.34	16536	9984.75	11285.7	19059.7	2491.03	7023.96	6918.13	15286.2	5332.04
Customer 5	8832.86	7233.57	7847.33	11261.1	15041.4	4822.12	6030.65	16728	4304.66	5144.17	12143.5	17578.8	11794.4
Customer 6	22186.2	19205.2	5998.86	3826.43	9304.34	18613.4	19828.8	12405.6	10995.8	9082.97	6390.89	4030.67	10535
Customer 7	25680.4	22079.5	10781.3	10239.6	6733.45	23019.9	24093.1	9157.71	16498.1	13155.1	12843.4	3725.59	17008.1
Customer 8	19212.5	15816.5	4388.48	6052.05	5704.29	16364.7	17451.5	8709.73	10159.7	6495.75	9110.91	6679.6	12630.3
Customer 9	20100.2	17691	4652.89	319.179	12051.5	15966.2	17248.8	15080.4	7758.28	7591.56	2741.58	7936.33	6590.51
Customer 10	12327.6	8245.86	9336.61	13670.6	8866.55	11948	12450.4	9083.03	11508.8	7179.22	16168.9	16027.6	18013.7
Customer 11	15160.2	11495.6	4974.39	8982.55	6228.01	13084.1	13994.3	8115.36	9142.39	4317.03	11723.5	11268.6	14229.9
Customer 12	0	4118.01	16204.6	20015.9	20840	5055.16	3936.99	21409	12982.2	13108.1	20962.6	25688	20169.1
Customer 13	4118.01	0	13383.4	17553.8	16898.1	5941.26	5639.62	17327	11507.4	10231	19005.5	22440.7	19021.8
Customer 14	16204.6	13383.4	0	4432.55	9504.29	12669.4	13861.5	12176.5	5780.82	3154.45	6847.09	9733.85	9303.62
Customer 15	20015.9	17553.8	4432.55	0	11732.5	15933.5	17210.5	14761.2	7780.31	7417.68	3060.7	7748.28	6882.01
Customer 16	20840	16898.1	9504.29	11732.5	0	19250	20096.9	3102.94	14871.9	10294	14792.7	9361.22	18313.2
Customer 17	5055.16	5941.26	12669.4	15933.5	19250	0	1312.06	20527.1	8401.46	9882.12	16449.6	22400	15258.4
Customer 18	3936.99	5639.62	13861.5	17210.5	20096.9	1312.06	0	21224.4	9710.58	11000.3	17759.9	23578.9	16545.9
Customer 19	21409	17327	12176.5	14761.2	3102.94	20527.1	21224.4	0	17194	12419.2	17820.6	12205.6	21222.3
Customer 20	12982.2	11507.4	5780.82	7780.31	14871.9	8401.46	9710.58	17194	0	4848.68	8050.89	14998.8	7522.65
Customer 21	13108.1	10231	3154.45	7417.68	10294	9882.12	11000.3	12419.2	4848.68	0	9368.3	12638.5	10837.4
Customer 22	20962.6	19005.5	6847.09	3060.7	14792.7	16449.6	17759.9	17820.6	8050.89	9368.3	0	9868.15	4220.01
Customer 23	25688	22440.7	9733.85	7748.28	9361.22	22400	23578.9	12205.6	14998.8	12638.5	9868.15	0	14087.9
Customer 24	20169.1	19021.8	9303.62	6882.01	18313.2	15258.4	16545.9	21222.3	7522.65	10837.4	4220.01	14087.9	0
Customer 25	17222.4	14425.1	1041.84	3486.54	9514.08	13604.4	14813.4	12325.5	6368.22	4195.34	6105.06	8838.8	8935.24
Customer 26	20130.6	16091.1	10473.4	13200	1952.73	19006.3	19750	1723.44	15473.6	10708.8	16251.9	11311.6	19570.7
Customer 27	18699.3	16942.6	5908.47	3775.64	14878.9	14102.3	15414.2	17773.5	5728.88	7811.54	2404.05	11444	3451.16
Customer 28	4643.69	2319.83	11685.2	15679.9	16394.5	4121.77	4263.62	17235.3	9273.36	8550.16	16938.5	21066.6	16765.7
Customer 29	24384.1	21765.2	8396.98	4418.82	12766	20352.3	21628.7	15854.8	12152.4	11538	5182.82	5239.93	9266.79
Customer 30	16264.9	14104.9	3045.72	3914.73	12507	12041.5	13327.4	15219.6	3896.43	4594.77	4924.97	11180.1	6415.85
Customer 31	20215	17569.1	4223.58	1075.29	10742.7	16310.8	17567.4	13793.3	8357.64	7347.09	4071.9	6846.73	7957.25
Customer 32	25722.5	22417.1	9876.66	8116.07	8964.24	22523.8	23688.7	11763.2	15239.7	12717.4	10326.3	559.148	14545.8
Customer 33	25597.7	22595.4	9403.75	6504.41	11015.2	21972.5	23200.7	13986.6	14157.7	12494.7	8139.82	2197.8	12322.3
Customer 34	16201.6	14313.8	4183.7	4504.71	13660.7	11764.3	13068.4	16360	3391.2	5345.09	4763.87	12052.9	5497.99
Customer 35	19715.9	18418.6	8402.82	6036.42	17411.2	14866.1	16165	20315.8	6913.6	10005.6	3526.73	13369.7	908.037
Customer 36	17136.5	14714.1	2234.35	2880.23	11379.6	13116.2	14381	14187.8	5203.69	4711.79	4720.62	9808.9	7114.58
Customer 37	22783	19610.8	6788.81	5317.63	8209.55	19453.9	20633.6	11282.8	12141	9707.5	7934.69	2946.09	12076.2
Customer 38	2913	3802	13537.5	17208.8	18999.7	2432.75	1840.78	19923.6	10072.8	10511.7	18069.3	23148.8	17271
Customer 39	26409.1	24411	11681.3	7261.96	17673	21846.7	23158.6	20773.3	13467.9	14540	5449.58	9947.64	7678.51
Customer 40	21388.1	17648.3	8006.56	9433.59	2717.98	19168.8	20145.8	5775.25	13706.1	9573.13	12465.8	6736.37	16205.5
Customer 41	14316.9	13207.1	6870.71	7749.34	16276.2	9484.04	10789.1	18748.4	1954.5	6604.42	7262.24	15355	5878.42
Customer 42	14561.1	11915	1754.56	5647.66	10601.8	10922.3	12123	13048.8	4306.18	1836.38	7534.71	11486.5	9213.55
Customer 43	19197	16122.9	3207.55	3831.2	7922.64	15848.4	17021.6	10933.3	8807.14	6101.03	6888.03	6557.88	10454.1
Customer 44	13107.2	9032.06	9143.29	13404.3	8088.99	12608.2	13149.6	8301.84	11718.7	7218.26	15990.4	15428.2	18008.7
Customer 45	11645.5	11787.6	10574.1	12133.2	19447.3	6594.77	7752.88	21564.8	4796.56	9175.03	11558.2	19645.7	9276.44
Customer 46	12066.1	7989.34	12059.9	16405	10611.3	12804.3	13042.3	10068.8	13816.1	9766.61	18880.4	18495.3	20592
Customer 47	19407.1	17798.6	6965.96	4464.71	15840.2	14709.8	16020.9	18762.3	6431.36	8846.35	2280.05	11928.9	2477.82
Customer 48	16120.9	13975.8	3080.04	4066.37	12559.9	11890.9	13177.3	15256.5	3747.42	4517.62	5045.13	11319	6445.89
Customer 49	22199.8	19249.8	6000.94	3661.57	9519.97	18587.5	19808.5	12621.8	10916.3	9103.82	6182.95	4144.79	10336.4
Customer 50	19458.9	15497.8	8927.55	11645.4	1423.37	17996.8	18809.4	3249.39	14035.3	9329.3	14693.5	10229.8	18000.8

	Customer 25	Customer 26	Customer 27	Customer 28	Customer 29	Customer 30	Customer 31	Customer 32	Customer 33	Customer 34	Customer 35	Customer 36	Customer 37
Warehouse 2	9292.37	16599.9	9374.75	6214.96	15705.9	7381.87	11754.3	18305.7	17504.4	7036.46	10292	8545.07	15199
Warehouse 1	8957.05	4758.29	14236.2	11098.7	14967.1	11175.5	11568.6	13050.4	14306.5	12189.2	16692.1	10574.8	11090.3
Customer 1	4521.01	12648.2	5661.13	17181.4	2837.53	5814.87	1412.71	6182.04	4494.23	6497.65	7697.68	4607.71	3617.33
Customer 2	9522.94	2246.8	14852.9	16658.4	12579	12535.6	10653.2	8680.99	10770.6	13690.5	17379.7	11385.1	8008.68
Customer 3	6170.43	12987.6	8166.5	6829.07	13233.4	5272.12	9069.03	15047	14576	5521.96	9909.53	5999.4	11979.9
Customer 4	7345.57	17337.4	4525.37	11490.5	11676.8	4324.39	8441.58	15605.2	14113.2	3275.45	4882.68	5711.99	12602.8
Customer 5	8786.25	15095.4	9924.7	4973.42	15667.3	7445.59	11562	17707	17170.2	7379.66	11213.2	8400.46	14632.8
Customer 6	5016.99	11065.7	7595.97	17629.3	3542.07	7154.56	2848.66	4339.65	3411.84	8024.09	9739.58	5794.51	1544.3
Customer 7	10132.4	8627.14	14010.2	21045.7	8768.43	12978.2	9194.63	3171.61	5906.03	14015.2	16212.1	11578.6	4937.04
Customer 8	4057.13	7157.65	9214.02	14569.5	7901.76	7127.48	5102.56	6612.07	7297.02	8272.6	11732.5	5861.99	4076.41
Customer 9	3731.75	13518.5	3538.85	15793.4	4400.8	3927.44	1372.92	8317.21	6622.87	4440.31	5752.37	2985	5566.1
Customer 10	10193.3	7872.15	14902	8289.58	17001.1	11688.3	13198.3	15855.9	16800.4	12516.8	17173.9	11464.9	13580.3
Customer 11	5618.53	6415.62	10872.2	10560.9	12062.6	7839.02	8388.3	11157.1	11885.7	8881.3	13337.6	7208.73	8683.5
Customer 12	17222.4	20130.6	18699.3	4643.69	24384.1	16264.9	20215	25722.5	25597.7	16201.6	19715.9	17136.5	22783
Customer 13	14425.1	16091.1	16942.6	2319.83	21765.2	14104.9	17569.1	22417.1	22595.4	14313.8	18418.6	14714.1	19610.8
Customer 14	1041.84	10473.4	5908.47	11685.2	8396.98	3045.72	4223.58	9876.66	9403.75	4183.7	8402.82	2234.35	6788.81
Customer 15	3486.54	13200	3775.64	15679.9	4418.82	3914.73	1075.29	8116.07	6504.41	4504.71	6036.42	2880.23	5317.63
Customer 16	9514.08	1952.73	14878.9	16394.5	12766	12507	10742.7	8964.24	11015.2	13660.7	17411.2	11379.6	8209.55
Customer 17	13604.4	19006.3	14102.3	4121.77	20352.3	12041.5	16310.8	22523.8	21972.5	11764.3	14866.1	13116.2	19453.9
Customer 18	14813.4	19750	15414.2	4263.62	21628.7	13327.4	17567.4	23688.7	23200.7	13068.4	16165	14381	20633.6
Customer 19	12325.5	1723.44	17773.5	17235.3	15854.8	15219.6	13793.3	11763.2	13986.6	16360	20315.8	14187.8	11282.8
Customer 20	6368.22	15473.6	5728.88	9273.36	12152.4	3896.43	8357.64	15239.7	14157.7	3391.2	6913.6	5203.69	12141
Customer 21	4195.34	10708.8	7811.54	8550.16	11538	4594.77	7347.09	12717.4	12494.7	5345.09	10005.6	4711.79	9707.5
Customer 22	6105.06	16251.9	2404.05	16938.5	5182.82	4924.97	4071.9	10326.3	8139.82	4763.87	3526.73	4720.62	7934.69
Customer 23	8838.8	11311.6	11444	21066.6	5239.93	11180.1	6846.73	559.148	2197.8	12052.9	13369.7	9808.9	2946.09
Customer 24	8935.24	19570.7	3451.16	16765.7	9266.79	6415.85	7957.25	14545.8	12322.3	5497.99	908.037	7114.58	12076.2
Customer 25	0	10653.2	5487.8	12717.1	7357.19	3070.43	3194.42	9013.63	8407.95	4217.79	8027.29	1867.09	5906.77
Customer 26	10653.2	0	16119.7	15844.6	14570.2	13518.5	12259.9	10916.9	12935.8	14655.5	18663.2	12510.1	10057.8
Customer 27	5487.8	16119.7	0	14807.4	7279.63	3221.65	4816.77	11841.2	9974.86	2644.9	2543.46	3689.45	9093.19
Customer 28	12717.1	15844.6	14807.4	0	19981.5	12085.6	15787.5	21091.5	21039.9	12202.8	16185.5	12808.7	18173.9
Customer 29	7357.19	14570.2	7279.63	19981.5	0	8324.48	4198.45	5776.34	3174.25	8801.91	8671.58	7267.61	4592.14
Customer 30	3070.43	13518.5	3221.65	12085.6	8324.48	0	4463.21	11453	10264.2	1155.38	5539.42	1403.56	8388.31
Customer 31	3194.42	12259.9	4816.77	15787.5	4198.45	4463.21	0	7181.91	5800.98	5231.85	7110.92	3209.37	4293.53
Customer 32	9013.63	10916.9	11841.2	21091.5	5776.34	11453	7181.91	0	2756.7	12356	13814.6	10069.4	3110.2
Customer 33	8407.95	12935.8	9974.86	21039.9	3174.25	10264.2	5800.98	2756.7	0	10990.1	11666.4	8980.75	3227.32
Customer 34	4217.79	14655.5	2644.9	12202.8	8801.91	1155.38	5231.85	12356	10990.1	0	4660.53	2464.96	9334
Customer 35	8027.29	18663.2	2543.46	16185.5	8671.58	5539.42	7110.92	13814.6	11666.4	4660.53	0	6208.74	11275.5
Customer 36	1867.09	12510.1	3689.45	12808.7	7267.61	1403.56	3209.37	10069.4	8980.75	2464.96	6208.74	0	6993.88
Customer 37	5906.77	10057.8	9093.19	18173.9	4592.14	8388.31	4293.53	3110.2	3227.32	9334	11275.5	6993.88	0
Customer 38	14535.8	18512.5	15794.1	2703.31	21600	13414.4	17456.3	23216.5	22940.2	13312.9	16807.9	14335.8	20219.2
Customer 39	10748.3	19423.8	7745.01	22373	4993.93	10306.2	7757.55	10505.5	7751.73	10213.4	7565.66	9831.49	9568.75
Customer 40	7756.41	4627.43	12831.9	16796.7	10084	10826.5	8394.55	6380.72	8308.5	11973.6	15319	9562.13	5507.56
Customer 41	7217.02	17025.1	4858.56	10925.8	11903.8	4265.98	8527.12	15656.9	14253.3	3302.17	5400.13	5668.61	12624.2
Customer 42	2694.62	11325.9	6057.78	10105.1	9878.59	2840.83	5682.59	11631.2	11101.2	3735.43	8353.26	2882.21	8542.18
Customer 43	2439.51	9369.55	7069.61	14605.6	6246.43	5341.19	2913.47	6677.64	6473.1	6435.76	9562.99	3970.84	3614.86
Customer 44	9950.53	7079.81	14828.4	9020.97	16593.2	11634.1	12879.2	15239.3	16271.4	12513.6	17154.2	11316.8	13045.3
Customer 45	11153.2	19864.9	9176.23	9509.29	16307.1	8466.77	12865.5	19918.7	18629.9	7641.05	9103.51	9851	16843.2
Customer 46	12926.2	9235.23	17550.9	8781.75	19702.9	14330	15930.5	18286.2	19387.5	15109.5	19769.4	14167.7	16160.4
Customer 47	6514.82	17119.5	1064.16	15628.5	7442.67	4268.43	5536.62	12356	10318.8	3594.07	1590.44	4740.59	9739.05
Customer 48	3151.89	13552.7	3293.93	11950.1	8475.98	151.643	4611.1	11589.2	10412	1104.01	5575.61	1529.81	8520.81
Customer 49	5005.57	11275.1	7423.54	17653.4	3352.18	7060.76	2708.08	4474.12	3403.43	7909.75	9545.82	5712.7	1752.74
Customer 50	9086.74	1570.59	14549.9	15044.7	13165	11970.2	10716.9	9880.84	11710.4	13111.2	17093.3	10945.8	8719.76

	Customer 38	Customer 39	Customer 40	Customer 41	Customer 42	Customer 43	Customer 44	Customer 45	Customer 46	Customer 47	Customer 48	Customer 49	Customer 50
Warehouse 2	6516.52	17119.6	15783.4	4898.43	6725.05	11665.6	11387.1	3807.71	12730.3	10015.8	7230.29	14166	15339.4
Warehouse 1	13777.8	19301.2	7043.07	13865.1	8482.44	8813.84	2462.26	16006.1	5269.09	15297.8	11158.9	11670	4209.27
Customer 1	18867.8	6902.41	8473.47	9760.26	7077.59	3630.24	13920.4	14136.1	17008.2	6195.89	5964.93	1865.85	11149.4
Customer 2	19254.7	17502.8	2502.08	16350.3	10691.6	7861.06	8403.94	19575.9	10943.5	15804	12592.3	9356.06	1741.88
Customer 3	8393.54	15577.3	12109.5	5365.25	3475.81	8371.85	8429.04	6933.38	10409.5	9082.14	5141.08	11175.3	11684.3
Customer 4	11943.8	12074	15057.3	569.13	6014.5	9665.36	14099.8	4655.56	16284.8	4871.08	4219.03	11142.1	15839.5
Customer 5	5968.94	17593	14638.1	5977.37	6100.74	11029.9	9624.26	5615.53	10922.3	10700.5	7299.69	13778.1	13902.8
Customer 6	19536.2	8369.13	6672.98	11326.2	7711.23	3109.65	13267.9	15617.4	16392.3	8211.65	7294.91	225.628	9637
Customer 7	23387.5	13621.8	4651.51	17242.9	12468.3	7698.91	14074.2	21285.5	16945.2	14675.3	13095.9	6671.92	7921.44
Customer 8	16811.3	12487.7	3702.23	11220.3	5940.25	2224.08	8978.16	14943.8	12098.6	10153.5	7208.14	4568.45	5615.82
Customer 9	17280.6	7028.79	9748.08	7648.23	5802.66	4150.01	13668	12043.5	16657.4	4188.76	4078.66	3884.7	11963.5
Customer 10	10992.5	20918.9	10138.7	13437.5	8847.42	10757.9	792.454	14705.1	2734.95	15955.5	11630.5	13856.7	7443.51
Customer 11	13041	16145.7	6236.03	10808.2	5230.87	5816.37	4532.49	13450.3	7641.23	11933.2	7830.04	8898.06	5109.54
Customer 12	2913	26409.1	21388.1	14316.9	14561.1	19197	13107.2	11645.5	12066.1	19407.1	16120.9	22199.8	19458.9
Customer 13	3802	24411	17648.3	13207.1	11915	16122.9	9032.06	11787.6	7989.34	17798.6	13975.8	19249.8	15497.8
Customer 14	13537.5	11681.3	8006.56	6870.71	1754.56	3207.55	9143.29	10574.1	12059.9	6965.96	3080.04	6000.94	8927.55
Customer 15	17208.8	7261.96	9433.59	7749.34	5647.66	3831.2	13404.3	12133.2	16405	4464.71	4066.37	3661.57	11645.4
Customer 16	18999.7	17673	2717.98	16276.2	10601.8	7922.64	8088.99	19447.3	10611.3	15840.2	12559.9	9519.97	1423.37
Customer 17	2432.75	21846.7	19168.8	9484.04	10922.3	15848.4	12608.2	6594.77	12804.3	14709.8	11890.9	18587.5	17996.8
Customer 18	1840.78	23158.6	20145.8	10789.1	12123	17021.6	13149.6	7752.88	13042.3	16020.9	13177.3	19808.5	18809.4
Customer 19	19923.6	20773.3	5775.25	18748.4	13048.8	10933.3	8301.84	21564.8	10068.8	18762.3	15256.5	12621.8	3249.39
Customer 20	10072.8	13467.9	13706.1	1954.5	4306.18	8807.14	11718.7	4796.56	13816.1	6431.36	3747.42	10916.3	14035.3
Customer 21	10511.7	14540	9573.13	6604.42	1836.38	6101.03	7218.26	9175.03	9766.61	8846.35	4517.62	9103.82	9329.3
Customer 22	18069.3	5449.58	12465.8	7262.24	7534.71	6888.03	15990.4	11558.2	18880.4	2280.05	5045.13	6182.95	14693.5
Customer 23	23148.8	9947.64	6736.37	15355	11486.5	6557.88	15428.2	19645.7	18495.3	11928.9	11319	4144.79	10229.8
Customer 24	17271	7678.51	16205.5	5878.42	9213.55	10454.1	18008.7	9276.44	20592	2477.82	6445.89	10336.4	18000.8
Customer 25	14535.8	10748.3	7756.41	7217.02	2694.62	2439.51	9950.53	11153.2	12926.2	6514.82	3151.89	5005.57	9086.74
Customer 26	18512.5	19423.8	4627.43	17025.1	11325.9	9369.55	7079.81	19864.9	9235.23	17119.5	13552.7	11275.1	1570.59
Customer 27	15794.1	7745.01	12831.9	4858.56	6057.78	7069.61	14828.4	9176.23	17550.9	1064.16	3293.93	7423.54	14549.9
Customer 28	2703.31	22373	16796.7	10925.8	10105.1	14605.6	9020.97	9509.29	8781.75	15628.5	11950.1	17653.4	15044.7
Customer 29	21600	4993.93	10084	11903.8	9878.59	6246.43	16593.2	16307.1	19702.9	7442.67	8475.98	3352.18	13165
Customer 30	13414.4	10306.2	10826.5	4265.98	2840.83	5341.19	11634.1	8466.77	14330	4268.43	151.643	7060.76	11970.2
Customer 31	17456.3	7757.55	8394.55	8527.12	5682.59	2913.47	12879.2	12865.5	15930.5	5536.62	4611.1	2708.08	10716.9
Customer 32	23216.5	10505.5	6380.72	15656.9	11631.2	6677.64	15239.3	19918.7	18286.2	12356	11589.2	4474.12	9880.84
Customer 33	22940.2	7751.73	8308.5	14253.3	11101.2	6473.1	16271.4	18629.9	19387.5	10318.8	10412	3403.43	11710.4
Customer 34	13312.9	10213.4	11973.6	3302.17	3735.43	6435.76	12513.6	7641.05	15109.5	3594.07	1104.01	7909.75	13111.2
Customer 35	16807.9	7565.66	15319	5400.13	8353.26	9562.99	17154.2	9103.51	19769.4	1590.44	5575.61	9545.82	17093.3
Customer 36	14335.8	9831.49	9562.13	5668.61	2882.21	3970.84	11316.8	9851	14167.7	4740.59	1529.81	5712.7	10945.8
Customer 37	20219.2	9568.75	5507.56	12624.2	8542.18	3614.86	13045.3	16843.2	16160.4	9739.05	8520.81	1752.74	8719.76
Customer 38	0	23512.6	19263	11410.8	11850.1	16610.2	11724	8943.57	11369.7	16495.2	13268.6	19536.8	17669.1
Customer 39	23512.6	0	15023	12492.5	12713.7	10509.7	20625.8	16579.8	23653.8	7204.56	10436.2	8155.08	17967.6
Customer 40	19263	15023	0	14874.6	9412.36	5762.27	9427.55	18446.6	12299.5	13730.2	10904.6	6893.53	3500.27
Customer 41	11410.8	12492.5	14874.6	0	5702.61	9582.02	13624.3	4405.1	15767.6	5290.59	4147.04	11211.5	15541.6
Customer 42	11850.1	12713.7	9412.36	5702.61	0	4959.78	8801.97	9037.07	11494.4	7108.63	2785.67	7699.69	9839.07
Customer 43	16610.2	10509.7	5762.27	9582.02	4959.78	0	10347.9	13588.4	13456.4	7976.63	5447.53	3186.65	7817.7
Customer 44	11724	20625.8	9427.55	13624.3	8801.97	10347.9	0	15108.3	3125.23	15888.3	11586	13400.8	6665.63
Customer 45	8943.57	16579.8	18446.6	4405.1	9037.07	13588.4	15108.3	0	16537.8	9434.3	8329.63	15517.1	18504
Customer 46	11369.7	23653.8	12299.5	15767.6	11494.4	13456.4	3125.23	16537.8	0	18597.3	14264.5	16523.9	9219.94
Customer 47	16495.2	7204.56	13730.2	5290.59	7108.63	7976.63	15888.3	9434.3	18597.3	0	4333.1	8024.05	15549
Customer 48	13268.6	10436.2	10904.6	4147.04	2785.67	5447.53	11586	8329.63	14264.5	4333.1	0	7202.71	12007.6
Customer 49	19536.8	8155.08	6893.53	11211.5	7699.69	3186.65	13400.8	15517.1	16523.9	8024.05	7202.71	0	9840.78
Customer 50	17669.1	17967.6	3500.27	15541.6	9839.07	7817.7	6665.63	18504	9219.94	15549	12007.6	9840.78	0

Appendix 4: MapBox Distance Matrix (m)

	Warehouse 2	Warehouse 1	Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Customer 6	Customer 7	Customer 8	Customer 9	Customer 10	Customer 11	Customer 12	Customer 13	Customer 14	Customer 15	Customer 16	Customer 17	Customer 18	Customer 19	Customer 20	Customer 21	Customer 22	Customer 23	Customer 24	Customer 25	Customer 26	Customer 27	Customer 28	Customer 29	Customer 30	Customer 31	Customer 32	Customer 33	Customer 34	Customer 35	Customer 36	Customer 37	Customer 38	Customer 39	Customer 40	Customer 41	Customer 42	Customer 43	Customer 44	Customer 45	Customer 46	Customer 47	Customer 48	Customer 49	Customer 50
Warehouse 2	0	19826.7	17580.6	6918.1	14316.6	23615.8	16166.7	17972.3	17397	10516.6	17481.8	4141.6	5354.2	30884.2	14112.2	13396.3	16466.2	6458.8	35738.8	34266.3	8733.4	22225.3	13723	35324.2	18384	27009.4	12792.7	6842.4	23265.3	15378.8	21434.9	19399.4	15995.2	17824	23702.2	19445.9	25827.4	15207.4	27034.2	32977.7	41367.7	10466	23794.2	19959	13016	3209.9	32762.6	12446.4	24351.2	19232.4	31109.5	5016.6
Warehouse 1	16964.8	0	14898.4	24981	5797	8052.1	2030.3	28719.3	34994.4	18195.1	14799.5	15780.2	14536.9	16258.9	11756	11960.2	13784	24521.6	7853.6	9220	26533.9	4670.3	8382.2	21452.1	31608.8	18280.4	13657.3	24902	18066.3	8548.8	26062.2	8355.5	13313	32508.9	29415	8402	17098.5	11146.7	20619.7	10113.4	24984.7	36588	8230.4	8416.4	14832.9	15051.6	5480	16399.7	17207.1	7772.8	28580.5	23079.5
Customer 1	17358.4	23668.3	0	18467.5	14107.5	17104.8	16595.7	2730.4	12411.9	11681.6	5096.7	20649.9	15193.5	35313.2	38268.5	9961.6	3272.7	18008.2	29957.1	31294.5	20020.4	13739.3	13270	6466.7	10429.2	11959.8	8112.2	18388.5	11033.5	31897	4906.9	10913.3	3308.6	11329.3	10006.7	10959.8	10777.9	6586.4	8004.3	31845.3	12510.2	15408.4	17283.2	11473	8319.4	20515	21843.3	36718	10174.3	10746.4	2591.5	16566
Customer 2	6941	37824.4	18283	0	22473	31260.9	24961.2	16546.9	9501	7119.1	17361.2	10361	8597.2	37684.5	31413.2	13275.7	19337	1002.1	44113.2	41066.6	4550.4	22104.8	17726.3	20622.7	13300.8	26115.9	12672.2	3846.6	25189.6	32679.9	19038.5	19278.8	19373	12740.8	18608.9	19325.3	24933.9	15086.9	12332.8	39778	26666.3	4102.2	31439.2	19838.5	9618.5	9429.3	35999.3	25612.6	24330.3	19111.9	16408	2838.1
Customer 3	13714.3	5147.6	13379.2	23461.9	0	7124.7	3915.7	15338	23556.4	16675.9	13280.4	11841.6	11286.4	19721.7	12268.5	9415.5	12264.8	23002.5	9902.1	11268.5	25014.8	3703.2	5131.7	18403.3	28560	15231.7	11112.7	23382.8	11487.6	9693.2	17233.5	7115.9	11793.8	20970.8	26366.2	7428.9	14049.7	9627.6	19100.6	10160.5	21935.9	20943	7303	5254.6	13313.8	11801.1	10482.6	12863.5	12573.5	6930.3	15038.9	21560.3
Customer 4	23429.4	8050.2	14455.8	31836.5	7001.1	0	9677.5	20104.3	26379.4	17752.5	14357	18225	21264.4	24515.9	27471.3	11517.7	13341.4	24079.1	19159.8	20497.2	26091.4	3255.9	11515.2	12837.1	22993.8	9665.5	13214.8	24459.5	5845.8	17331.8	17447.2	6376	12870.5	23893.9	20800	6422.5	8483.5	10704.2	23192.4	21048	16369.7	27973	724.6	7774.8	14390.4	18184.6	6680.6	25182.7	6931.6	6209.1	19965.5	22636.9
Customer 5	15043.3	1977.1	15787.7	25870.3	3873.3	9626.1	0	17746.4	36841.2	19084.4	15688.8	13170.5	12615.4	17056.9	11382.2	10744.5	14673.3	25410.9	7237.3	8603.7	30016.7	6244.3	6460.7	23298.9	33455.6	20127.2	12441.6	25791.3	19913.1	8175	27909	9524.3	14202.3	34355.7	31261.8	9976	18945.3	12036	21509	8642.4	26831.5	23351.5	9804.5	7663	15722.2	13130.1	7326.8	16025.9	19053.9	9338.7	30427.3	23968.7
Customer 6	18209.1	17914.4	2883.8	19318.2	14958.2	15980.5	17446.4	0	11970.4	7412.2	5974.3	21500.5	16044.1	38597.2	41552.6	10812.3	5037.4	18858.9	33241.2	34578.5	20871.1	14590	14120.7	8397.1	9993	13890.2	8962.9	19239.2	12963.9	24387	6837.3	11764	4458.3	9433.1	11937.1	11810.5	12708.3	7437.1	7562.8	35129.4	14440.6	8868.3	16158.9	12323.7	6542.7	21365.7	25127.3	37568.7	12104.7	11597.1	604.8	17416.7
Customer 7	24889.6	37978.7	18437.2	10320.9	30326.2	31415.2	40451.7	16701.2	0	12749	21315.3	27458.9	17472.9	43579.4	37308.1	18926.4	19491.3	11117.8	44267.5	45604.8	13953.9	32062.1	21629.9	20777	9184.1	26270.2	18322.9	12978.4	25343.9	38574.8	15801.2	30344.8	19527.3	8600.4	9616.3	30391.3	25088.2	22805.1	12487.1	46155.7	26820.6	6008.2	31593.5	27691.7	15248.4	26857.5	36153.6	31507.5	24484.6	30177.8	16562.3	10436.5
Customer 8	11296.3	18177.8	8191.8	7493.5	15221.6	16243.9	17709.8	7831.5	8919.2	0	10109.9	14587.8	6562.2	38860.7	36456.5	5884.2	9094.3	7079.4	33504.6	34842	13958.3	14853.4	8982.5	17194.7	9906.1	19637.5	5280.6	12326.4	21761.5	24650.5	14063	12027.5	8623.3	9346.1	15224.3	12074	18455.5	7835.5	8904.8	35392.8	23238.3	5647.3	16422.3	12587.2	3441.4	14452.9	25390.7	30655.9	20902.3	11860.5	8436.3	10503.9
Customer 9	16485.2	16190.5	3607.2	17594.4	13234.3	14256.6	15722.5	5997.4	17688.8	10808.4	0	19776.7	14320.3	37850	40805.4	9088.4	1537.4	17135	32493.9	33831.3	19147.2	12866.1	12396.8	9003.5	12966	14496.7	7239	17515.3	13570.3	34433.8	7443.7	10040.2	2453.3	13866.1	12543.6	10086.7	13314.7	4238	10622.2	34382.2	15047.1	15075.5	14435	10599.8	7446.3	19641.8	24380.1	35844.8	12711.1	9873.2	5698.3	15692.8
Customer 10	4086.5	16204.7	20304.9	10365.4	11309.7	28384.6	14972.7	34382.7	28119.5	13240.9	20206.1	0	7464.4	27948.1	10490.1	16120.5	19190.5	9906	18339.6	19706	12173.3	14938.3	10716.2	38458.6	31136.6	31957.3	15517	10282.3	31743.1	11756.8	24159.2	22123.6	18719.5	30576.7	36444.7	22170.1	30775.3	17931.7	30168.7	15337	38661.5	13278.9	28562.9	16395.9	15740.2	1464.3	19156.8	8824.4	30883.9	21956.7	34243.9	8463.8
Customer 11	5354.2	30884.2	14112.2	13396.3	16466.2	6458.8	35738.8	34266.3	8733.4	22225.3	13723	35324.2	0	35662.5	15188.9	10121.9	13191.9	8416.3	17856.3	19222.7	11768.4	18951	9426.3	19720.4	15109.6	23735	9518.4	10136.4	19991	16455.5	18160.6	16125	12720.9	14549.7	20427.8	16171.5	22553.1	11933.1	14108.3	37756	25763.9	7715.4	20519.8	16684.7	9741.6	7194.2	29488.3	13523.1	21076.8	15958.1	15965.9	7449.6
Customer 12	14536.9	16258.9	11756	11960.2	13784	24521.6	7853.6	9220	26533.9	4670.3	8382.2	21452.1	33405	0	6431.2	35390.6	37214.3	35310	7580.6	5608.4	30506.3	25885.3	18346.2	31982.7	42139.4	28811.1	37087.7	32449.8	28596.9	7074.9	36592.8	30248.9	36743.3	43039.5	39945.6	30295.4	27629.1	34577.1	45469.7	4381.9	35515.3	41766.9	25416.7	20343.6	38263.3	26707.1	17501.3	23452.1	27737.7	30082	39111.1	33867.8
Customer 13	15193.5	35313.2	38268.5	9961.6	3272.7	18008.2	29957.1	31294.5	20020.4	13739.3	13270	6466.7	15569.3	6581.5	0	21076.5	40228.1	30021.8	6779.2	7895.4	25218.1	16801	16792.7	34996.5	45153.2	31824.8	36376.6	27161.6	31610.7	4189	39606.5	33262.7	39757.1	40589.5	42959.4	33309.2	30642.9	37590.8	40181.5	4803.2	38529.1	36478.7	28430.5	18790.1	36599.9	11473.5	15570.9	11227.3	30751.5	33095.7	42124.9	28579.6
Customer 14	8597.2	37684.5	31413.2	13275.7	19337	1002.1	44113.2	41066.6	4550.4	22104.8	17726.3	20622.7	7424.9	33570.4	36525.7	0	7573.5	14079.3	28214.3	29551.7	16091.5	9563.1	5283	14102	12638.7	14347.2	2881.4	14459.6	10603.1	19360.2	12542.2	6737.2	7102.6	12078.8	17642.1	6783.7	13165.2	5739.5	10208.5	30102.5	21051.5	12019.8	11132	7296.9	4421.7	11816.9	20100.4	18145.7	11689	6570.2	9704.3	12637.1
Customer 15	11286.4	19721.7	12268.5	9415.5	12264.8	23002.5	9902.1	11268.5	25014.8	3703.2	5131.7	18403.3	13723.9	36612.1	39567.5	8492.1	0	16538.6	31256.1	32593.4	18550.9	12269.8	11800.5	7765.6	11728.1	13258.8	6642.7	16919	12332.5	33196	6205.8	9443.8	1856.9	12628.3	11305.7	9490.3	12076.8	3671.6	9808.3	33144.3	13809.2	14479.2	18582.1	10003.5	6849.9	19045.5	23142.2	35248.4	11473.2	9276.9	5101.9	15096.4
Customer 16	21264.4	24515.9	27471.3	11517.7	13341.4	24079.1	19159.8	20497.2	26091.4	3255.9	11515.2	12837.1	8117	37204.3	30933.1	12795.6	15865.5	0	42058.9	40586.5	4070.2	21624.6	17246.2	21206.3	13884.3	26699.5	12192.1	3366.5	25773.1	32199.7	19622	18798.7	15394.6	13324.4	19192.4	18845.2	25517.5	14606.7	12916.4	39297.9	27249.9	5354.1	23193.5	19358.4	9138.4	8949.1	36582.9	25132.5	24913.9	18631.7	16991.6	2358
Customer 17	12615.4	17056.9	11382.2	10744.5	14673.3	25410.9	7237.3	8603.7	30016.7	6244.3	6460.7	23298.9	18826.6	7342.1	7456.7	30310.5	32134.2	40778.5	0	3179.3	35974.8	11265.9	12671.9	26902.6	37059.3	23731	32007.6	37918.3	23516.8	6027.9	31512.7	25168.8	31663.2	37959.4	34865.5	25215.3	22549	29497	37257.9	2847.6	30435.2	42038.5	20336.6	14738.9	33183.2	18012.5	10357.2	17766.3	22657.6	25001.9	34031	39336.3
Customer 18	16044.1	38597.2	41552.6	10812.3	5037.4	18858.9	33241.2	34578.5	20871.1	14590	14120.7	8397.1	37406.1	5898.1	7819.6	30029.1	31852.8	39311.1	2045.1	0	34507.4	12565.1	13971.1	26621.2	36778	23449.6	31726.2	36451	23235.5	7408.3	31231.3	24887.4	31381.8	37678.1	34584.1	24933.9	22267.6	29215.6	36976.5	3028.4	30153.9	45768	20055.2	16038.1	32901.8	18375.4	12139.8	27453.2	22376.2	24720.5	33749.6	37869
Customer 19	17472.9	43579.4	37308.1	18926.4	19491.3	11117.8	44267.5	45604.8	13953.9	32062.1	21629.9	20777	12354.8	32470.1	26198.9	15955.7	23159.2	4075.3	37324.8	35852.3	0	24784.8	20406.3	24445	17123	29938.1	15352.2	2821.7	29011.8	27465.5	22860.7	21958.8	23195.2	16563	22431.1	22005.3	28756.1	17766.9	16155	34563.7	30488.5	12452.2	35261.4	22518.5	18413.3	11214.8	39821.6	20398.3	28152.6	21791.9	20230.3	5104.9
Customer 20	6562.2	38860.7	36456.5	5884.2	9094.3	7079.4	33504.6	34842	13958.3	14853.4	8982.5	17194.7	14453.8	24683.4	16538.3	9530.6	11354.3	22092	11351.9	12718.3	24104.2	0	8299.1	15434.3	25591	12262.6	11227.7	22472.3	8847.8	13331.1	20044.4	4993.4	10883.3	26491.1	23397.2	5039.9	11080.7	8717.1	18190	13611.7	18966.9	20032.5	3822.5	6496.3	12403.3	14968.5	6848.1	21182	9933.7	4410.7	14128.3	20649.8
Customer 21	14320.3	37850	40805.4	9088.4	1537.4	17135	32493.9	33831.3	19147.2	12866.1	12396.8	9003.5	9974.8	22320.4	16101.4	5429.5	11677.7	18324.5	12500.7	13867.2	20336.7	8292.9	0	18206.2	16883.9	17648.2	7126.6	18704.8	13904.1	12291.9	16646.4	10038.2	11206.7	16324	21746.2	10084.7	16466.2	9040.5	14453.7	12759.2	24352.5	16265	14433	9750.5	8666.9	10489.5	13320.3	16818.4	14990	9871.2	14451.8	16882.3
Customer 22	7464.4	27948.1	10490.1	16120.5	19190.5	9906	18339.6	19706	12173.3	14938.3	10716.2	38458.6	23050.9	32004.2	34959.5	17867.1	8431.3	22716.8	26648.1	27985.5	25552.9	14442.8	21175.5	0	13077.2	8650.8	19564.2	27181.1	7724.5	28588	7530.5	12725.4	8467.3	13977.3	10883.4	12771.9	7468.9	17053.6	13275.7	28536.3	8384.4	18056.4	13974.2	16019.6	20739.8	38456.5	18534.3	43106.5	6865.3	12558.5	10048.9	23659
Customer 23	18384	27009.4	12792.7	6842.4	23265.3	15378.8	21434.9	19399.4	15995.2	17824	23702.2	19445.9	15109.6	23735	9518.4	10136.4	19991	16455.5	18160.6	16125	12720.9	14549.7	20427.8	16171.5	0	19048.5	11681.6	20831.4	18122.2	38985.7	8215.9	23123.1	12305.6	1371.8	5327.4	23169.6	17866.5	15583.4	5316.3	38934	19598.9	11706.7	24371.8	20470	8918	32106.8	28932	36756.8	17263	22956.2	9059.6	17309.3
Customer 24	31608.8	18280.4	13657.3	24902	18066.3	8548.8	26062.2	8355.5	13313	32508.9	29415	8402	42139.4	28811.1	37087.7	32449.8	28596.9	7074.9	36592.8	30248.9	36743.3	43039.5	39945.6	30295.4	17920.6	0	16608.7	27853.4	4360.6	25632.5	12373.9	9770	13310.7	18820.7	15726.8	9816.5	1427.1	14098.1	18119.1	25580.8	9822.2	22899.8	11018.7	13064.1	17784.3	43299.9	15578.8	32320.5	3501.4	9603	14892.3	26030.8
Customer 25	10429.2	11959.8	8112.2	18388.5	11033.5	31897	4906.9	10913.3	3308.6	11329.3	10006.7	10959.8	45153.2	31824.8	36376.6	27161.6	31610.7	4189	39606.5	33262.7	39757.1	40589.5	42959.4	33309.2	11646.7	17124.7	0	13467.6	13380.7	22137.7	11550.2	9514.7	6110.6	11086.8	16650.1	9561.2	15942.8	5322.8	9216.5	32880	19153.6	11027.8	13909.5	10074.4	3429.7	15594.1	22878	31797.1	14466.5	9347.7	8712.3	11645.1
Customer 26	13300.8	26115.9	12672.2	3846.6	25189.6	32679.9	19038.5	19278.8	19373	12740.8	18608.9	19325.3	12638.7	14347.2	2881.4	14459.6	10603.1	19360.2	12542.2	6737.2	7102.6	12078.8	17642.1	6783.7	16570.6	27663.5	13446.8	0	23919.4	29269.9	22089	20053.4	16649.3	16010.7	21878.7	20099.9	26481.5	15861.5	15602.6	36368	29936.1	7745.4	24448.3	20613.1	13670.1	8621.9	33416.7	22202.6	27600.2	19886.5	19677.9	4277.9
Customer 27	28560	15231.7	11112.7	23382.8	11487.6	9693.2	17233.5	7115.9	11793.8	20970.8	26366.2	7428.9	11728.1	13258.8	6642.7	16919	12332.5	33196	6205.8	9443.8	1856.9	12628.3	11305.7	9490.3	17013.1	4654.9	12595.6	23840.2	0	26440.9	11466.5	5756.8	12251.2	17913.3	14819.3	5803.3	3473	10085	17211.7	26389.2	10008.7	21992.3	6030	9051	13771.2	25966.8	16387.1	33128.8	1470.7	5589.9	13984.8	22017.7
Customer 28	22993.8	9665.5	13214.8	24459.5	5845.8	17331.8	17447.2	6376	12870.5	23893.9	20800	6422.5	13884.3	26699.5	12192.1	3366.5	25773.1	32199.7	19622	18798.7	15394.6	13324.4	19192.4	18845.2	39104	25775.6	19198.9	28342.5	25561.5	0	33557.4	27213.5	33707.9	40004.1	36910.2	27260	24593.7	31541.6	41362.4	4140.5	32479.9	37659.6	22381.3	15215.4	35227.9	12654.4	11996.2	12408.2	24702.3	27046.6	36075.7	29760.5
Customer 29	33455.6	20127.2	12441.6	25791.3	19913.1	8175	27909	9524.3	14202.3	34355.7	31261.8	9976	37059.3	23731	32007.6	37918.3	23516.8	6027.9	31512.7	25168.8	31663.2	37959.4	34865.5	25215.3	8375	14819.8	11137.8	22478.9	13893.5	34757	0	18894.5	6334.2	9275.1	4643.1	18941	13637.9	9611.9	8573.6	34705.3	15370.2	13354.2	20143.2	14498.6	11345	33754.3	24703.3	38404.4	13034.3	18727.5	4830.9	18956.8
Customer 30	9993	13890.2	8962.9	19239.2	12963.9	24387	6837.3	11764	4458.3	9433.1	11937.1	11810.5	36778	23449.6	31726.2	36451	23235.5	7408.3	31231.3	24887.4	31381.8	37678.1	34584.1	24933.9	23094.6	9766.2	8733.1	19977.8	6022.2	16594.1	13828.5	0	8388.8	17565.8	20900.8	1627.9	8584.3	6222.5	15695.5	25521.5	16470.5	17538	6551	4530.8	9908.7	18231.5	15519.5	24444.9	7108	141.7	11633.8	18155.3
Customer 31	9184.1	26270.2	18322.9	12978.4	25343.9	38574.8	15801.2	30344.8	19527.3	8600.4	9616.3	30391.3	17123	29938.1	15352.2	2821.7	29011.8	27465.5	22860.7	21958.8	23195.2	16563	22431.1	22005.3	12507.8	16456.6	6045.5	16321.7	13112.1	21469.6	6985.5	8846.6	0	10500.1	12085.3	8893.1	15274.7	4355.7	8629.9	32211.9	14588.8	13881.9	13241.4	9406.3	6252.7	18448.2	22209.9	34651.2	12252.9	8679.6	3705.9	14499.2
Customer 32	9906.1	19637.5	5280.6	12326.4	21761.5	24650.5	14063	12027.5	8623.3	9346.1	15224.3	12074	25591	12262.6	11227.7	22472.3	8847.8	13331.1	20044.4	4993.4	10883.3	26491.1	23397.2	5039.9	1395.3	19434.2	11121.4	18120.1	18507.8	39371.3	9422.9	23508.8	12691.3	0	6721.1	23555.3	18252.2	15969	4756.1	39319.7	19984.6	10113	24757.5	20855.7	8357.8	30513.1	29317.6	35163.2	17648.6	23341.8	9726.3	13898.9
Customer 33	12966	14496.7	7239	17515.3	13570.3	34433.8	7443.7	10040.2	2453.3	13866.1	12543.6	10086.7	16883.9	17648.2	7126.6	18704.8	13904.1	12291.9	16646.4	10038.2	11206.7	16324	21746.2	10084.7	5312.3	16439.3	14500	21286.8	15513	36376.5	5606.7	20513.9	9696.4	5988.7	0	20560.4	15257.3	12974.2	7381.5	36324.8	16989.7	12162.2	21762.6	17860.8	14932.2	32562.3	26322.7	37212.3	14653.7	20346.9	6450.3	17764.7
Customer 34	31136.6	31957.3	15517	10282.3	31743.1	11756.8	24159.2	22123.6	18719.5	30576.7	36444.7	22170.1	13077.2	8650.8	19564.2	27181.1	7724.5	28588	7530.5	12725.4	8467.3	13977.3	10883.4	12771.9	23099.5	9771.1	8738	19982.7	6027.1	25578.1	13833.4	1561.5	8393.7	17570.7	20905.7	0	8589.2	6227.4	15700.4	25526.4	16475.4	17542.9	6555.9	5202.3	9913.6	22109.2	15524.4	32266.1	7112.9	1394.6	11638.7	18160.2
Customer 35	25827.4	15207.4	27034.2	32977.7	41367.7	10466	23794.2	19959	13016	3209.9	32762.6	12446.4	22553.1	11933.1	14108.3	37756	25763.9	7715.4	20519.8	16684.7	9741.6	7194.2	29488.3	13523.1	17866.5	15583.4	5316.3	38934	19598.9	11706.7	24371.8	20470	8918	32106.8	28932	36756.8	0	12880.8	16901.8	24363.5	10521.1	21682.4	9801.3	11846.8	16567	42082.5	14361.5	31103.1	2284	8385.7	13674.9	24813.5
Customer 36	17098.5	11146.7	20619.7	10113.4	24984.7	36588	8230.4	8416.4	14832.9	15051.6	5480	16399.7	27629.1	34577.1	45469.7	4381.9	35515.3	41766.9	25416.7	20343.6	38263.3	26707.1	17501.3	23452.1	1427.1	14098.1	18119.1	25580.8	9822.2	22899.8	11018.7	13064.1	17784.3	43299.9	15578.8	32320.5	13008.1	0	11540.8	29945.4	17194.4	13383.3	10974.9	7139.7	5754.1	17949.6	19943.3	34152.6	14858.5	6413.1	7845.6	14000.6
Customer 37	10777.9	6586.4	8004.3	31845.3	12510.2	15408.4	17283.2	11473	8319.4	20515	21843.3	36718	30642.9	37590.8	40181.5	4803.2	38529.1	36478.7	28430.5	18790.1	36599.9	11473.5	15570.9	11227.3	15942.8	5322.8	9216.5	32880	19153.6	11027.8	13909.5	10074.4	3429.7	15594.1	22878	31797.1	15059.3	12776.1	0	36126.7	16791.6	9321.9	21564.6	17662.7	6239.4	29722	26124.7	34372.1	14455.7	20148.9	6533.4	14924.5
Customer 38	24933.9	15086.9	12332.8	39778	26666.3	4102.2	31439.2	19838.5	9618.5	9429.3	35999.3	25612.6	13165.2	5739.5	10208.5	30102.5	21051.5	12019.8	11132	7296.9	4421.7	11816.9	20100.4	18145.7	26481.5	15861.5	15602.6	36368	29936.1	7745.4	24448.3	20613.1	13670.1	8621.9	33416.7	22202.6	24527	31475	39235.9	0	32413.3	44462.3	22314.7	15608.1	35161.2	16100.4	14399.2	15854.3	24635.7	26979.9	36009.1	36563.3
Customer 39	14049.7	9627.6	19100.6	10160.5	21935.9	20943	7303	5254.6	13313.8	11801.1	10482.6	12863.5	12076.8	3671.6	9808.3	33144.3	13809.2	14479.2	18582.1	10003.5	6849.9	19045.5	23142.2	35248.4	3473	10085	17211.7	26389.2	10008.7	21992.3	6030	9051	13771.2	25966.8	16387.1	33128.8	10560.8	20956.8	18696.8	32439.5	0	23477.5	17877.4	19922.8	24643	43877.6	22437.5	39179.2	10768.5	16461.7	15470	29080.1
Customer 40	8483.5	10704.2	23192.4	21048	16369.7	27973	724.6	7774.8	14390.4	18184.6	6680.6	25182.7	25517.5	14606.7	12916.4	39297.9	27249.9	5354.1	23193.5	19358.4	9138.4	8949.1	36582.9	25132.5	24593.7	31541.6	41362.4	4140.5	32479.9	37659.6	22381.3	15215.4	35227.9	12654.4	11996.2	12408.2	21812.2	13529.1	9211.1	42879.7	23544.6	0	28317.5	18280.7	7889.1	13262.7	32877.6	31477.7	21208.6	17554.1	13286.3	5690.7
Customer 41	18945.3	12036	21509	8642.4	26831.5	23351.5	9804.5	7663	15722.2	13130.1	7326.8	16025.9	22549	29497	37257.9	2847.6	30435.2	42038.5	20336.6	14738.9	33183.2	18012.5	10357.2	17766.3	13637.9	9611.9	8573.6	34705.3	15370.2	13354.2	20143.2	14498.6	11345	33754.3	24703.3	38404.4	8927.3	10566.2	23636.2	20910.1	16813.6	28416.8	0	7636.8	14252.4	18046.6	6542.6	25044.7	6793.7	6071.1	20409.3	22499
Customer 42	12708.3	7437.1	7562.8	35129.4	14440.6	8868.3	16158.9	12323.7	6542.7	21365.7	25127.3	37568.7	22267.6	29215.6	36976.5	3028.4	30153.9	45768	20055.2	16038.1	32901.8	18375.4	12139.8	27453.2	8584.3	6222.5	15695.5	25521.5	16470.5	17538	6551	4530.8	9908.7	18231.5	15519.5	24444.9	11280.9	6858.7	16331.7	14798.3	19167.1	18174.2	7710.9	0	10545	15968.4	18216.1	22181.8	9804.6	4161.5	12270	18791.5
Customer 43	25088.2	22805.1	12487.1	46155.7	26820.6	6008.2	31593.5	27691.7	15248.4	26857.5	36153.6	31507.5	28756.1	17766.9	16155	34563.7	30488.5	12452.2	35261.4	22518.5	18413.3	11214.8	39821.6	20398.3	15274.7	4355.7	8629.9	32211.9	14588.8	13881.9	13241.4	9406.3	6252.7	18448.2	22209.9	34651.2	16108.5	5488.5	6410.1	33045.8	22203.6	7693.7	14075.3	10240.1	0	15466.5	23043.7	31669.5	14632.3	9513.5	7173.3	11517.5
Customer 44	18455.5	7835.5	8904.8	35392.8	23238.3	5647.3	16422.3	12587.2	3441.4	14452.9	25390.7	30655.9	11080.7	8717.1	18190	13611.7	18966.9	20032.5	3822.5	6496.3	12403.3	14968.5	6848.1	21182	18252.2	15969	4756.1	39319.7	19984.6	10113	24757.5	20855.7	8357.8	30513.1	29317.6	35163.2	28341.6	17721.6	29226.7	16473.7	43560.2	12577.1	26308.4	22473.2	15530.2	0	20293.5	9961.1	41224.2	21746.6	33301.9	7530.8
Customer 45	13314.7	4238	10622.2	34382.2	15047.1	15075.5	14435	10599.8	7446.3	19641.8	24380.1	35844.8	16466.2	9040.5	14453.7	12759.2	24352.5	16265	14433	9750.5	8666.9	10489.5	13320.3	16818.4	15257.3	12974.2	7381.5	36324.8	16989.7	12162.2	21762.6	17860.8	14932.2	32562.3	26322.7	37212.3	12882.7	19830.7	27591.6	13146	20769	32372.2	6842.4	14268.9	23516.9	20553.4	0	20307.2	12991.3	15335.6	24364.7	31763.4
Customer 46	30775.3	17931.7	30168.7	15337	38661.5	13278.9	28562.9	16395.9	15740.2	1464.3	19156.8	8824.4	7468.9	17053.6	13275.7	28536.3	8384.4	18056.4	13974.2	16019.6	20739.8	38456.5	18534.3	43106.5	8589.2	6227.4	15700.4	25526.4	16475.4	17542.9	6555.9	5202.3	9913.6	22109.2	15524.4	32266.1	46075.8	32033.5	33423.7	26798.5	47757.2	29720.9	43863.4	36785.2	29842.1	9830.9	21298.9	0	45421.2	36058.5	37498.9	21821.8
Customer 47	24484.6	30177.8	16562.3	10436.5	24889.6	37978.7	18437.2	10320.9	30326.2	31415.2	40451.7	16701.2	28152.6	21791.9	20230.3	5104.9	12354.8	32470.1	26198.9	15955.7	23159.2	4075.3	37324.8	35852.3	12252.9	8679.6	3705.9	14499.2	12507.8	16456.6	6045.5	16321.7	13112.1	21469.6	6985.5	8846.6	14632.3	9513.5	7173.3	11517.5	16108.5	5488.5	6410.1	33045.8	22203.6	7693.7	14075.3	10240.1	0	6655.7	13751.8	23083.5
Customer 48	20902.3	11860.5	8436.3	10503.9	11296.3	18177.8	8191.8	7493.5	15221.6	16243.9	17709.8	7831.5	9933.7	4410.7	14128.3	20649.8	14453.8	24683.4	16538.3	9530.6	11354.3	22092	11351.9	12718.3	17648.6	23341.8	9726.3	13898.9	1395.3	19434.2	11121.4	18120.1	18507.8	39371.3	9422.9	23508.8	41224.2	21746.6	33301.9	7530.8	28341.6	17721.6	29226.7	16473.7	43560.2	12577.1	26308.4	22473.2	6966.3	0	11492.1	18013.5
Customer 49	12711.1	9873.2	5698.3	15692.8	16485.2	16190.5	3607.2	17594.4	13234.3	14256.6	15722.5	5997.4	14990	9871.2	14451.8	16882.3	9974.8	22320.4	16101.4	5429.5	11677.7	18324.5	12500.7	13867.2	14653.7	20346.9	6450.3	17764.7	5312.3	16439.3	14500	21286.8	15513	36376.5	5606.7	20513.9	12991.3	15335.6	24364.7	31763.4	12882.7	19830.7	27591.6	13146	20769	32372.2	6842.4	14268.9	11965.9	11458.3	0	17277.8
Customer 50	30883.9	21956.7	34243.9	8463.8	4086.5	16204.7	20304.9	10365.4	11309.7	28384.6	14972.7	34382.7	6865.3	12558.5	10048.9	23659	23050.9	32004.2	34959.5	17867.1	8431.3	22716.8	26648.1	27985.5	7112.9	1394.6	11638.7	18160.2	23099.5	9771.1	8738	19982.7	6027.1	25578.1	13833.4	1561.5	45421.2	36058.5	37498.9	21821.8	46075.8	32033.5	33423.7	26798.5	47757.2	29720.9	43863.4	36785.2	26486.2	17607.6	18563.9	0

Appendix 5: Google Maps Distance Matrix (m)

	Warehouse 2	Warehouse 1	Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Customer 6	Customer 7	Customer 8	Customer 9	Customer 10	Customer 11	Customer 12	Customer 13	Customer 14	Customer 15	Customer 16	Customer 17	Customer 18	Customer 19	Customer 20	Customer 21	Customer 22	Customer 23	Customer 24	Customer 25	Customer 26	Customer 27	Customer 28	Customer 29	Customer 30	Customer 31	Customer 32	Customer 33	Customer 34	Customer 35	Customer 36	Customer 37	Customer 38	Customer 39	Customer 40	Customer 41	Customer 42	Customer 43	Customer 44	Customer 45	Customer 46	Customer 47	Customer 48	Customer 49	Customer 50
Warehouse 2	0	19438	17585	6922	14110	23570	15954	17952	14861	10489	17087	4123	4800	30427	14175	13340	16419	6480	35697	34113	8481	21707	11963	25615	18085	26928	12752	6431	23166	15381	21258	19354	15961	17801	20466	19398	25785	15172	16459	32914	31585	9968	23442	19918	12400	3146	22736	12444	24267	19228	19213	5021
Warehouse 1	19281	0	14942	24955	5503	7217	2027	16699	36473	18169	14444	15699	14363	15956	12234	11909	13776	24513	7520	9169	26536	5016	8221	21050	31065	17861	14607	24464	17649	8484	25768	7958	13319	32088	28969	8391	16718	11155	31426	9012	24522	22469	7089	8412	14842	16644	5100	17749	16799	7806	16570	23054
Customer 1	17150	14803	0	19088	13274	16942	15757	2677	15550	7224	3136	20562	13861	34717	38131	7843	2451	17162	32623	31040	19185	12429	11816	6316	10142	11776	7255	17112	10334	31690	4864	10075	2495	11165	7089	10120	10633	5819	7693	31648	12286	9251	16814	10640	6958	19584	20239	39826	10004	9949	2549	15703
Customer 2	6922	24870	18420	0	23341	31334	25824	16650	10064	7068	20119	10334	9308	37213	20386	14128	19434	654	42482	40899	4559	22495	16108	20707	13127	26167	13540	3510	24726	21592	16350	20142	19493	12843	15484	20187	25024	15960	12466	39699	26678	4212	31206	20706	9572	9356	34630	25154	24396	20016	16522	2834
Customer 3	13538	5709	13415	23428	0	7102	3916	15171	23525	16642	12917	11542	11133	19778	12832	10382	12249	22986	9933	11582	25009	3693	4991	18376	22607	15186	13080	22936	11424	9640	17329	7106	11792	20960	20511	7657	14044	9628	18848	10168	21848	20941	6974	5251	13315	11658	11078	12848	12526	6955	15043	21527
Customer 4	24038	7220	17033	24492	6986	0	8846	19803	27962	17706	13981	17822	20749	24066	27481	11446	13313	24050	21972	20389	26073	3262	11271	12538	22553	9349	14144	24000	5787	15785	17256	6885	12856	23576	20457	6929	8206	10692	22914	20998	16010	27663	727	8361	14379	17938	6667	25050	6888	6758	19675	22591
Customer 5	15060	2026	15874	25886	3930	8866	0	17830	25984	19100	15376	14203	12655	17158	10739	10747	14708	25444	7313	8962	28858	6603	6513	23289	33305	20100	12472	25395	19888	7020	28007	9565	14250	34328	31209	10115	18957	12086	18014	7548	26762	23400	8738	7710	15773	13180	7340	16253	19039	9413	17502	23986
Customer 6	18002	16488	2659	18310	14959	20184	17442	0	14772	6836	5961	21413	14713	37959	41373	8721	5029	18365	35865	34282	21945	14114	12595	9558	9364	15018	8133	17964	13576	34932	4805	11760	3851	9416	6663	11805	13875	7441	7304	34890	15528	8863	20056	12324	6570	20436	23480	39048	13246	11634	433	16555
Customer 7	14896	35745	16265	10389	23485	29179	38207	14495	0	8543	17964	27783	11617	43186	37397	14273	17279	10444	44860	43277	14024	29838	16555	18553	6413	24013	13685	12647	22571	38603	14196	20287	17338	6129	9605	20331	22870	16105	10312	43885	24523	6094	29051	20851	11048	26829	32476	31127	22241	20161	14367	10480
Customer 8	10445	16818	7232	7087	15289	16307	17772	6859	8499	0	9981	13857	6542	45143	21235	6507	9067	6676	43048	41465	12480	14444	7944	16741	9211	22201	5919	10407	20760	24657	12384	12091	7870	8927	11592	12135	21058	7908	5424	42074	22712	5385	16179	12655	2650	12879	30664	34240	20430	11964	7158	6863
Customer 9	16868	14521	3365	17321	12992	14009	15475	5969	17953	10535	0	20279	13579	37120	40534	7561	1670	16879	35026	33443	18903	12146	11534	8719	12545	14179	6973	16830	12737	22360	7267	9793	2589	13568	10449	9838	13036	4383	10507	34052	14689	14835	13881	10357	7208	19302	22642	35116	12407	9667	5840	15421
Customer 10	4109	15777	21058	10394	11221	17867	14216	21425	27748	13962	20560	0	7007	16504	10514	11858	19892	9952	16856	18292	12384	15192	9074	38391	30811	31835	16225	9901	26638	11720	34034	22827	19434	30527	33168	22871	30693	18645	30150	15276	44361	13441	17739	16299	15873	1534	19075	8783	30774	22700	34205	8494
Customer 11	4862	13711	14327	9364	9999	16405	11843	14694	11614	7092	13829	7002	0	34344	15147	7545	13161	8953	17861	19510	11781	14542	6030	22217	14687	19763	7617	9708	16001	16353	17860	12189	12703	14403	17068	12233	18620	11914	13061	19909	28188	7736	16277	12753	9142	6309	24091	13416	17102	12063	15955	8299
Customer 12	29157	16695	36638	35941	15646	25408	11860	39409	43224	39526	38337	26128	33031	0	6729	35499	37653	35499	6986	5427	30712	26068	17263	32144	42159	28955	38197	32215	28743	6980	36862	30432	37712	43182	40063	30476	27812	34746	45626	4268	35616	41929	25280	19867	38433	26771	17703	23226	27893	30306	39281	34040
Customer 13	14149	12227	39986	20435	12811	28756	10667	42757	37733	21313	25975	10566	15151	6763	0	16330	41000	19993	6751	7387	25221	17217	16070	35491	40796	32302	23576	26725	32090	4195	40209	20164	24849	40512	43411	20714	31159	22685	40136	4730	38964	36438	28628	18754	23223	11512	15526	12617	31240	20012	42628	18534
Customer 14	13654	11404	8316	14107	9875	10892	12358	8702	14205	6521	7818	11666	7292	33057	16126	0	7150	13666	16935	29379	15689	9030	4622	13682	11948	14250	2442	13616	10488	19243	12230	6676	6692	11664	14305	6721	13108	5681	9552	29988	20912	11621	10765	7240	4018	16088	18578	18080	11590	6550	9002	12207
Customer 15	16162	13815	2417	16615	12286	13303	14769	5007	17005	9829	1688	19573	12873	36173	39587	6855	0	16173	34078	32495	18197	11440	10827	7771	11597	13231	6267	16124	11790	33145	6320	9087	1860	12620	9501	9132	12088	3677	9559	33104	13741	16706	13175	9651	6502	18596	21694	34410	11459	8961	4879	14714
Customer 16	6466	24413	18474	654	22884	31388	25367	16705	10119	6645	20173	9877	8885	36756	19929	13672	19489	0	42025	40442	4108	22039	15651	20762	13182	26222	13084	3058	24781	21135	16405	19686	19548	12898	15539	19730	25079	15504	12521	39243	26732	4266	31260	20250	9149	8900	34685	24698	24450	19559	16576	2378
Customer 17	34409	7095	32171	41193	10111	20941	7291	34942	43100	24567	33870	16859	18836	7154	7552	16928	33186	40751	0	2091	35964	11652	12694	27677	37692	24487	18653	37467	24276	6013	32395	15031	33245	38715	35596	15026	23345	17553	38053	2891	31149	42801	20813	14842	21240	17805	10370	18910	23426	14880	34814	39292
Customer 18	32833	8719	31003	39618	11735	19773	8915	33774	41932	36124	32702	29805	20460	5605	7950	29864	32018	39176	2910	0	34388	20432	14318	26509	36524	23319	32562	35892	23108	6410	31227	24797	32077	37547	34428	24841	22177	29110	36885	2951	29981	41633	19645	16466	32797	30447	12068	26903	22258	24670	33646	37717
Customer 19	8478	26446	22109	4574	24917	35023	29935	20339	13753	12854	23808	12137	11689	32021	26232	15705	23123	4123	37290	35707	0	24071	17684	24397	16816	29856	15117	2462	28415	27438	20039	21718	23182	16532	19173	21763	28714	17536	16156	34508	30367	8353	34895	22282	14764	10913	38319	19962	28085	21592	20211	5595
Customer 20	22058	4623	12499	22512	3946	3643	6351	14255	22609	15726	12001	14818	18769	24246	18103	9465	11333	22070	11270	20568	24093	0	8267	15414	25429	12224	12163	22020	7555	13109	20132	4594	10875	26452	23333	4774	11082	8712	17932	21177	18886	20025	3515	5893	12399	14934	6847	23618	8679	4442	14127	20611
Customer 21	12381	8369	11838	15908	4658	11304	6502	12388	16197	8531	11340	10384	6623	22364	15945	4402	10672	15466	12519	14168	17489	8629	0	17204	14065	16620	6127	15416	12858	12226	15752	9046	10215	13781	18934	9091	15477	8051	11669	12754	23282	13421	11176	9736	7704	10501	13288	16798	13960	8920	13466	14007
Customer 22	23116	20348	7388	21855	18166	13783	22811	10158	18317	18268	9087	39249	22927	31558	34972	17797	8402	21910	29464	27881	25490	14442	20543	0	12908	8617	13222	24784	7175	28531	7611	12729	8461	13931	10812	12773	7474	11722	13269	28489	8282	18018	13655	16014	13457	38295	17079	42593	6845	12603	10030	23596
Customer 23	18173	31558	12079	15416	23931	24992	34021	9080	6424	9697	13778	32810	14356	42768	42424	12159	13093	15471	40673	39090	19051	25652	14450	14366	0	19826	11571	18345	18385	39741	7880	23938	13152	1269	5140	23983	18683	16413	5236	39699	20336	11579	24865	21297	8817	31856	28289	36154	18054	23812	8951	17157
Customer 24	27383	17343	12185	26653	15161	10777	19805	14956	23114	23066	13884	31545	24095	28552	31967	14791	13200	26707	26458	24875	30288	11437	17537	7691	17706	0	17489	27346	4268	25525	12409	8987	13259	18729	15610	9032	1380	14037	18067	25484	10451	22815	10649	12273	17724	29818	14074	33595	3418	8861	14828	28393
Customer 25	13053	14211	7715	13506	12682	13700	15165	8101	13604	5920	7217	16464	9764	35865	17859	2481	6549	13084	20539	32187	15088	11837	6355	13081	11347	17058	0	13015	13296	22051	11629	9484	6091	11062	13704	9528	15915	5302	8951	32796	19051	11020	13572	10048	3417	15487	21386	31301	14397	9358	8401	11606
Customer 26	6141	24836	18339	3765	23307	24325	25790	19838	12967	11244	17841	9553	10079	33916	28127	14095	17173	3314	39185	37602	2764	22462	16074	23896	16315	29355	13507	0	23920	29333	19538	20108	16716	16031	18672	20153	28213	15926	15655	36402	29866	7423	24197	20673	13154	8575	32010	21857	27584	19982	19710	3733
Customer 27	23161	16556	11278	25745	11406	5799	19018	14049	22207	16829	12977	26573	19872	27765	31179	10569	12292	25800	25671	24088	29380	7534	13315	6783	16799	4271	13267	23123	0	24738	11501	5486	12351	17822	14703	5530	3128	9815	17159	24697	9872	21908	5672	8771	13502	25595	13287	32808	1383	5360	13920	21714
Customer 28	15307	8951	33253	21593	9914	22023	7071	36024	38892	22472	22698	11725	16309	7048	3906	16708	34267	21151	5996	6631	26380	13507	12474	28758	38774	25569	18433	27884	25357	0	33476	16887	34326	39797	36678	16882	24426	19408	39134	3998	32231	37597	21895	15078	23095	12671	11849	13775	24508	16736	35895	19693
Customer 29	20947	24685	5219	17041	17071	18120	27148	4828	13503	13454	6918	34435	18113	35895	39309	11641	6233	17096	33801	32218	20676	18779	15613	7494	8095	12954	11053	19970	11512	32868	0	13873	6292	9117	5371	13917	11811	9553	8455	32826	13464	13204	17992	14437	10672	33481	21416	37779	11182	13747	4699	18782
Customer 30	19548	7918	9989	20002	7169	6355	9652	11745	20099	13216	9491	22960	16259	28519	20287	6955	8823	19560	14565	24842	21583	4683	9702	12902	22917	9713	9653	19510	5951	16537	13903	0	8366	17534	20821	1612	8570	6202	15422	25451	16374	17515	6227	4534	9889	21982	14041	25802	7052	147	11617	18101
Customer 31	15498	13151	2749	15952	11622	12640	14105	3836	13024	9166	2624	18910	12209	34804	24741	6191	1956	15510	19479	31127	17533	10777	10164	8115	11941	13575	5604	15460	12133	20990	7193	8424	0	10482	9051	8468	12432	4104	8370	31735	14085	9929	12512	8988	5839	17932	20326	33746	11803	8297	3708	14051
Customer 32	17723	31108	11629	13952	23481	24542	33571	9859	5974	9247	13328	31346	13906	42318	40960	11709	12643	14007	40223	38640	17587	25202	14000	13916	1197	19376	11121	16881	17935	39291	9559	23488	12702	0	6249	23533	18233	15963	4786	39249	19886	10115	24415	20847	8367	20158	27839	34690	17605	23362	9730	15693
Customer 33	21770	28080	6991	15941	20453	21514	30542	6540	8686	12355	10299	33336	17014	39289	42704	14536	9615	15996	37195	35612	19576	22173	18995	10888	5119	16348	13948	18870	14907	36262	5341	20460	8981	6182	0	20505	15205	12935	7462	36221	16858	12104	21386	17819	12385	32381	24811	36679	14576	20334	6412	17682
Customer 34	19612	8360	10053	20066	7871	6419	10354	11810	20163	13280	9555	23024	16323	28584	31998	7020	8887	19624	26489	24906	21647	4811	9766	12966	22982	9777	9718	19574	6015	17239	13968	1604	8430	17598	20886	0	8634	6266	15486	25515	16439	17579	6291	5236	9953	22047	14105	26504	7116	1477	11681	18165
Customer 35	26247	16206	11048	25516	14024	9641	18669	13819	21977	21929	12748	30408	22958	27416	30830	13654	12063	25570	25321	23738	29151	10300	16400	6554	16569	1360	16352	26209	3131	24389	11272	7851	12122	17592	14473	7895	0	12900	16930	24347	10471	21679	9513	11136	16587	28681	12937	32458	2281	7724	13691	27256
Customer 36	15418	11312	5782	15872	9783	10801	12266	7512	15969	9086	4342	18830	12130	32965	22902	5579	3674	15430	17640	29288	17453	8938	8325	11136	14961	14159	5524	15381	10397	19152	9684	6585	4133	13404	12865	6629	13016	0	11293	29897	17106	13386	10673	7149	5759	17853	18487	33667	11498	6458	7384	13971
Customer 37	18999	27923	8444	13170	20297	21358	30386	6674	7258	5364	10143	30565	14243	39133	42547	9592	9458	13225	37039	35455	16805	22017	11883	10732	5038	16191	9004	16099	14750	36106	6374	20304	9517	4754	7396	20348	15049	12779	0	36064	16702	9333	21230	17662	6250	29610	24654	33908	14420	20178	6546	14911
Customer 38	31623	9307	33321	38407	10429	22091	7586	36092	44250	38443	35021	15291	19876	44190	5569	32182	34336	37965	2885	2951	33178	13863	14905	28827	38842	25638	34880	34681	25426	4054	33545	27115	34395	39865	36746	27159	24495	31429	39203	0	32299	44394	21964	15593	35116	16237	14386	17341	24576	26989	35964	36506
Customer 39	28471	24187	12742	27210	22005	17622	26650	15513	23671	23623	14441	44604	28282	35397	38811	21636	13757	27264	33303	31720	30845	18281	24382	8313	18263	10451	18577	30139	10593	32370	12966	14995	13816	19286	16167	15040	10491	17077	18624	32328	0	23372	17494	18281	18812	43650	20918	47948	9694	14869	15385	28950
Customer 40	10017	22426	9291	4189	20897	28136	23380	8918	6207	5427	12039	13429	7692	41517	21363	11685	16237	424	43817	42234	8189	20052	13664	17510	9930	22970	11097	7140	21529	22569	13153	17699	9929	9646	12287	17743	21827	13516	9269	42843	23480	0	28009	18263	7932	12452	31433	29458	21199	17572	13324	5929
Customer 41	23910	7092	14351	24364	6858	727	8718	20529	28687	17578	13853	17694	20621	23938	27353	11318	13185	23922	21844	20261	25945	3134	11143	13264	23279	10074	14016	23872	5659	15657	17982	6757	12728	24302	21183	6801	8932	10564	23640	20870	16736	21877	0	8233	14251	17810	6540	24922	6761	6631	20401	22463
Customer 42	20422	8334	10863	20876	5106	9277	7589	12619	20973	14090	10365	16056	17133	31441	18225	7830	9697	20434	14141	15790	22457	5981	9506	15824	20055	12634	10528	20384	8872	14475	14778	4554	9240	18408	17959	5105	11492	7076	16296	15003	19296	18389	9149	0	10763	16173	16962	23740	9974	4403	12491	18975
Customer 43	13289	14735	6971	13743	13206	14224	15689	6598	10964	3053	7741	16701	10000	36388	24079	4009	7073	13301	21063	32711	15324	12361	7882	16142	8561	17582	3421	13251	13820	22575	10686	10008	7610	8277	10919	10052	16439	5826	6166	33320	22112	7699	14096	10572	0	15723	21910	31537	14921	9881	6898	11842
Customer 44	3090	16603	20039	9375	11297	26024	13141	20406	17315	12943	19752	1450	6298	28084	11340	15794	18873	8934	17682	31770	10935	15269	9151	28069	20539	29382	15206	8866	25620	12547	23712	21808	18415	20255	22920	21852	28239	17626	18913	16103	34039	12422	25896	22372	14854	0	19902	9690	26721	21681	21667	7475
Customer 45	23177	5392	21701	36168	11349	6645	7854	24471	32630	26822	23400	19594	19981	16181	19596	20562	22715	36223	14087	12504	39803	6664	13839	17206	27222	14017	23260	33116	13805	12379	21924	15494	22774	28244	25126	15539	12874	19808	27582	13113	20678	32331	6517	14257	23495	20540	0	21644	12955	15368	24343	31707
Customer 46	16981	18006	39404	23765	12869	25231	16447	37634	31048	27351	33948	8994	13579	24341	12745	30201	33280	23323	19086	28028	18536	22997	15677	41691	34111	34066	29613	20040	45710	13951	37334	36215	32822	33827	36468	36260	32923	32033	34350	17507	47662	29753	25103	18454	29261	9940	21306	0	33005	36089	37506	21868
Customer 47	25744	15703	11118	25585	12527	9138	18166	13888	22047	21998	12817	29906	22455	26913	30327	13151	12132	25640	24819	23235	29220	9797	15898	6623	16638	3419	15849	25706	1383	23886	11341	6607	12191	17661	14542	6652	2276	12398	16999	23844	9686	21748	9010	9893	16085	28178	12434	31956	0	6481	13760	24297
Customer 48	19422	7767	9863	19875	7017	6229	9500	11619	19973	13089	9365	22833	16133	28393	20136	6829	8697	19433	14414	24716	21457	4531	9576	12776	22791	9586	9527	19384	5824	16385	13777	147	8239	17407	20695	1486	8444	6075	15296	25324	16248	17389	6101	4383	9762	21856	13914	25651	6926	0	11491	17975
Customer 49	18706	16360	2531	18182	14830	20056	17313	433	14643	7135	5832	22118	15418	37831	41245	9021	4900	18236	35736	34153	21817	13985	13372	9429	9235	14889	8433	18669	13448	34804	4677	11632	3722	10258	6535	11676	13746	7312	7604	34762	15400	9162	19928	12196	6870	21141	23532	38920	13118	11506	0	17259
Customer 50	5074	23022	20070	2815	21493	22511	23976	18300	10541	6795	16027	8486	8265	35365	18538	12281	15359	237.3	40634	39051	5603	20648	14260	22357	14777	27817	11693	3784	26376	19744	18000	18295	14902	14493	17134	18339	26675	14112	14117	37852	28328	5862	22383	18859	11340	7509	30196	23306	26046	18168	18172	0

Appendix 6: Code for Retrieving MapBox Distance Matrix

```
import json
import requests
import csv
import numpy

coords = numpy.empty([52, 2])

with open('coords.csv') as csv_file:
  csv_reader = csv.reader(csv_file, delimiter=',')
  for idx, row in enumerate(csv_reader):
    coords[idx,:] = row

indexes = numpy.empty([6, 2])
indexes[0,:] = [0,11]
indexes[1,:] = [12,23]
indexes[2,:] = [24,35]
indexes[3,:] = [36,47]
indexes[4,:] = [48,51]
indexes[5,:] = [52,55]

dd = numpy.empty([52,52])
dt = numpy.empty([52,52])

for s in range(5):
  for d in range(5):
    min_s = indexes[s,0]
    min_d = indexes[d,0]
    if s < d:
      min_d = indexes[d,0]-12
    elif s > d:
      min_s = indexes[s,0]-12
    s_l = list(range(int(indexes[s,0]-min_s), int(indexes[s+1,0]-min_s)))
    d_l = list(range(int(indexes[d,0]-min_d), int(indexes[d+1,0]-min_d)))
    s_s = ';'.join([str(i) for i in s_l])
    d_s = ';'.join([str(i) for i in d_l])
    c_s = None
    c_l = None
    if s==d:
      c_l = coords[int(indexes[s,0]):int(indexes[s+1,0]),:]
    else:
      c_l = coords[int(indexes[s,0]):int(indexes[s+1,0]),:]
      c_l = numpy.append(c_l, coords[int(indexes[d,0]):int(indexes[d+1,0]),:], axis=0)

    c_s = ';'.join(','.join(str(x) for x in y) for y in c_l)

    url = "https://api.mapbox.com/directions-matrix/v1/mapbox/driving/" + c_s + "?destinations="
+ d_s + "&sources=" + s_s + "&annotations=duration,distance&access_token=API_KEY"
    print("Calling url: " + url)
    response = requests.get(url)
    json_data = json.loads(response.text)
    if json_data['code'] != "Ok":
      print("Error calling API: " + str(s) + " d: " + str(d))
    dd[int(indexes[s,0]):int(indexes[s+1,0]), int(indexes[d,0]):int(indexes[d+1,0])] =
json_data['distances']
    dt[int(indexes[s,0]):int(indexes[s+1,0]), int(indexes[d,0]):int(indexes[d+1,0])] =
json_data['durations']

numpy.savetxt("d_dist.csv", dd, delimiter=",")
numpy.savetxt("d_time.csv", dt, delimiter=",")
```

Appendix 7: Code for Retrieving Google Maps Distance Matrix

```
import json
import requests
import csv
import numpy

coords = numpy.empty([52, 2])

with open('coords.csv') as csv_file:
  csv_reader = csv.reader(csv_file, delimiter=',')
  for idx, row in enumerate(csv_reader):
    coords[idx,:] = row

indexes = numpy.empty([7, 2])
indexes[0,:] = [0,9]
indexes[1,:] = [10,19]
indexes[2,:] = [20,29]
indexes[3,:] = [30,39]
indexes[4,:] = [40,49]
indexes[5,:] = [50,51]
indexes[6,:] = [52,55]

dd = numpy.empty([52,52])
dt = numpy.empty([52,52])

for s in range(6):
  for d in range(6):
    s_s = None
    s_l = coords[int(indexes[s,0]):int(indexes[s+1,0]),:]
    d_s = None
    d_l = coords[int(indexes[d,0]):int(indexes[d+1,0]),:]

    s_s = '|'.join(','.join(str(x) for x in y) for y in s_l)
    d_s = '|'.join(','.join(str(x) for x in y) for y in d_l)

    url = "https://maps.googleapis.com/maps/api/distancematrix/json?origins=" + s_s +
"&destinations=" + d_s + "&key=API_KEY"
    print("Calling Url: " + url)
    response = requests.get(url)
    json_data = json.loads(response.text)

    distances = numpy.empty([int(indexes[s+1,0]-indexes[s,0]), int(indexes[d+1,0]-indexes[d,0])])
    times = numpy.empty([int(indexes[s+1,0]-indexes[s,0]), int(indexes[d+1,0]-indexes[d,0])])
    rows = json_data['rows']
    for i in range(len(rows)):
      row = rows[i]
      elements = row['elements']
      for j in range(len(elements)):
        element = elements[j]
        if element['status'] != 'OK':
          print('Error: i: ' + str(i) + ' j: ' + str(j))
        distances[i,j] = element['distance']['value']
        times[i,j] = element['duration']['value']
    dd[int(indexes[s,0]):int(indexes[s+1,0]), int(indexes[d,0]):int(indexes[d+1,0])] = distances
    dt[int(indexes[s,0]):int(indexes[s+1,0]), int(indexes[d,0]):int(indexes[d+1,0])] = times

numpy.savetxt("d_dist.csv", dd, delimiter=",")
numpy.savetxt("d_time.csv", dt, delimiter=",")
```

Appendix 8: Solver Interface Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import RadioButtons
import random
from matplotlib.widgets import Button
from matplotlib.text import Text
import pulp
import math
import time

import networkx
from pyscipopt import Model, quicksum, SCIP_PARAMSETTING
import itertools
import csv

loc_x = []
loc_y = []
n = 50
c = 50
loc_size = 1000
marker_size = 5
trucks = 2
solver = "gurobi"
li = [0] * n
warehouses = 1
lines_x = []
lines_y = []
obj_value = ""
exec_value = ""
model = None
x = None

min_loc_x = None
min_loc_y = None
max_loc_x = None
max_loc_y = None

distances = [[0 for x in range(n+warehouses)] for y in range(n+warehouses)]
distances2 = {}

def generate_data(n,whs):
    #global loc_x,loc_y,
    global obj_value,exec_value,distances,distances2
    global min_loc_x,min_loc_y,max_loc_x,max_loc_y
    distances = [[0 for x in range(n+whs)] for y in range(n+whs)]
    distances2 = {}

    with open(‘distances_1wh.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for idx,row in enumerate(csv_reader):
            for i in range(len(row)):
                distances2[idx,i] = float(row[i])
                distances[idx][i] = float(row[i])

    with open('coords_1wh.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            lng = float(row[0])
            lat = float(row[1])
            loc_x.append(lng)
            loc_y.append(lat)

        min_loc_x = min(loc_x)
        min_loc_y = min(loc_y)
        max_loc_x = max(loc_x)
        max_loc_y = max(loc_y)
        for i in range(len(loc_x)):
            loc_x[i] = (loc_x[i] - min_loc_x) / (max_loc_x - min_loc_x) * 1000
```

```
        for i in range(len(loc_y)):
            loc_y[i] = (loc_y[i] - min_loc_y) / (max_loc_y - min_loc_y) * 1000
            print('lng: ' + str(loc_x[i]) + ' lat: ' + str(loc_y[i]))

    obj_value = ""
    exec_value = ""

def plot_data(ax, loc_x, loc_y, warehouses):
    l, = ax.plot(loc_x[warehouses:], loc_y[warehouses:], linestyle='None', marker='o',
markersize=marker_size, antialiased=True, markeredgecolor='#F5A623',
markerfacecolor='#F8E71C',markeredgewidth=2)
    l2, = ax.plot(loc_x[0:warehouses], loc_y[0:warehouses], linestyle='None', marker='s',
markersize=marker_size+2, antialiased=True, markeredgecolor='#7ED321',
markerfacecolor='#B8E986',markeredgewidth=3)


fig, ax = plt.subplots()

generate_data(n,warehouses)
plot_data(ax, loc_x, loc_y, warehouses)

plt.axis([0, 1000, 0, 1000])
plt.subplots_adjust(left=0.3)

# Radio Buttons - Li
rax = plt.axes([0.05, 0.875, 0.15, 0.1])
radioli = RadioButtons(rax, ('Hide Li', 'Show Li'))

def lifunc(label):
    global li, loc_x, loc_y, n, warehouses,fig,ax, obj_value, exec_value

    if label=="Show Li":
        for idx, val in enumerate(li):
            x = loc_x[idx+warehouses]
            y = loc_y[idx+warehouses]
#            ax.text(x+15, y-15, "Li=" + str(int(val)), family="sans-serif",
horizontalalignment='left', verticalalignment='top')
            ax.text(x+15, y-15, "i=" + str(int(val)), family="sans-serif",
horizontalalignment='left', verticalalignment='top')
        fig.canvas.draw()
    if label=="Hide Li":
        ax.clear()
        ax.set_xlim(0,1000)
        ax.set_ylim(0,1000)

        ax.plot(lines_x, lines_y, color='#B8E986', markeredgewidth=50, markersize=50,
linewidth=5)
        ax.plot(lines_x, lines_y, color='#7ED321')


        ax.text(350, 1075, obj_value, family="serif", horizontalalignment='right',
verticalalignment='top')
        ax.text(450, 1075, exec_value, family="serif", horizontalalignment='left',
verticalalignment='top')

        plot_data(ax, loc_x, loc_y, warehouses)
        fig.canvas.draw()


radioli.on_clicked(lifunc)

# Radio Buttons - n
rax = plt.axes([0.05, 0.7, 0.15, 0.175])
radio = RadioButtons(rax, ('n=10', 'n=25', 'n=50', 'n=75', 'n=100', 'n=150'))

def nfunc(label):
    global loc_x, loc_y, n, warehouses, ax, li
    s = label.split('=')
    n = int(s[1])
    generate_data(n,warehouses)
    ax.clear()
```

```
    ax.set_xlim(0,1000)
    ax.set_ylim(0,1000)
    plot_data(ax, loc_x, loc_y, warehouses)
    li = []
    #l, = ax.plot(loc_x[1:], loc_y[1:], linestyle='None', marker='o', markersize=marker_size,
antialiased=True, markeredgecolor='#F5A623', markerfacecolor='#F8E71C',markeredgewidth=2)
    #l2, = ax.plot(loc_x[0], loc_y[0], linestyle='None', marker='s', markersize=marker_size+2,
antialiased=True, markeredgecolor='#7ED321', markerfacecolor='#B8E986',markeredgewidth=3)
    plt.draw()
radio.on_clicked(nfunc)
#

# Radio Buttons - warehouses
rax = plt.axes([0.05, 0.575, 0.15, 0.125])
radiowh = RadioButtons(rax, ('1 WHS', '2 WHS', '3 WHS'))

def whfunc(label):
    global warehouses, n, ax, loc_x, loc_y, fig
    s = label.split(" ")
    warehouses = int(s[0])
    ax.clear()
    ax.set_xlim(0,1000)
    ax.set_ylim(0,1000)
    generate_data(n,warehouses)
    plot_data(ax, loc_x, loc_y, warehouses)
    fig.canvas.draw()
radiowh.on_clicked(whfunc)

# Radio Buttons - trucks
rax = plt.axes([0.05, 0.4, 0.15, 0.175])
radio2 = RadioButtons(rax, ('1 Truck', '2 Trucks', '3 Trucks', '4 Trucks', '5 Trucks'))

def trucksfunc(label):
    global trucks
    s = label.split(" ")
    trucks = int(s[0])
radio2.on_clicked(trucksfunc)

# Radio Buttons - truck capacity
rax = plt.axes([0.05, 0.15, 0.15, 0.25])
radio4 = RadioButtons(rax, ('c=5', 'c=10', 'c=15', 'c=20', 'c=25', 'c=50', 'c=75', 'c=100'))

def cfunc(label):
    global c
    s = label.split('=')
    c = int(s[1])
radio4.on_clicked(cfunc)

# Radio Buttons - solver
rax = plt.axes([0.05, 0.0, 0.15, 0.15])
radio3 = RadioButtons(rax, ('CBC', 'GLPK', 'Held-Karp', 'Cut Plane'))

def solverfunc(label):
    global solver
    solver = label.lower()
radio3.on_clicked(solverfunc)

# Plot connecting lines
def plot_data_lines(lines_x,lines_y):
    ax.plot(lines_x, lines_y, color='#B8E986', markeredgewidth=50, markersize=50, linewidth=5)
    ax.plot(lines_x, lines_y, color='#7ED321')

def addcut(cut_edges,model,x,warehouses):
    invalid = False #ADDED
    G = networkx.Graph()
    G.add_edges_from(cut_edges)
    Components = list(networkx.connected_components(G))
    if len(Components) == 1:
        return False

    for S in Components:
```

```
            invalid_path = True
            for warehouse in range(0,warehouses):
                if warehouse in S:
                    invalid_path = False
                    break
            if invalid_path:
                invalid=True
                break

        if not invalid:
            return False

        model.freeTransform()

        for S in Components:
            invalid_path = True
            for warehouse in range(0,warehouses):
                if warehouse in S:
                    invalid_path = False
                    break
            if invalid_path:
                 model.addCons(quicksum(x[i,j] for i in S for j in S if j!=i) <= len(S)-1)
                 print("cut: len(%s) <= %s" % (S,len(S)-1))

        return True

    def held_karp(dists):
        """
        Implementation of Held-Karp, an algorithm that solves the Traveling
        Salesman Problem using dynamic programming with memoization.

        Parameters:
            dists: distance matrix

        Returns:
            A tuple, (cost, path).
        """
        n = len(dists)

        C = {}

        for k in range(1, n):
            C[(1 << k, k)] = (dists[0][k], 0)

        for subset_size in range(2, n):
            for subset in itertools.combinations(range(1, n), subset_size):
                # Set bits for all nodes in this subset
                bits = 0
                for bit in subset:
                    bits |= 1 << bit

                # Find the lowest cost to get to this subset
                for k in subset:
                    prev = bits & ~(1 << k)

                    res = []
                    for m in subset:
                        if m == 0 or m == k:
                            continue
                        res.append((C[(prev, m)][0] + dists[m][k], m))
                    C[(bits, k)] = min(res)

        bits = (2**n - 1) - 1

        res = []
        for k in range(1, n):
            res.append((C[(bits, k)][0] + dists[k][0], k))
        opt, parent = min(res)

        path = []
        for i in range(n - 1):
```

```
        path.append(parent)
        new_bits = bits & ~(1 << parent)
        _, parent = C[(bits, parent)]
        bits = new_bits

    path.append(0)

    return opt, list(reversed(path))


def solve(event):
    global n, trucks, solver, li, ax, warehouses, loc_x, loc_y, lines_x, lines_y, obj_value,
exec_value
    global distances, distances2
    print("solving...")

    ax.clear()
    ax.set_xlim(0,1000)
    ax.set_ylim(0,1000)
    plot_data(ax, loc_x, loc_y, warehouses)
    ax.text(400, 1075, "solving...", family="serif", horizontalalignment='left',
verticalalignment='top')
    plt.draw()
    fig.canvas.draw()

    start_time = time.time()

    # Calculate constraint for Li
    total_l = 0
    total_c = 0.0
    paths = 0
    slack = False
    while total_c < n or paths < trucks:
        c1 = min(c,n)
        total_c += c1
        total_l += c1*(c1+1)/2
        paths += 1
    if total_c > n:
        slack = True

    if solver=="cbc" or solver=="glpk" or solver=="gurobi":

        # Objective function
        z = pulp.LpProblem('Test', pulp.LpMinimize)

        # Generate decision variables
        x = {}
        y = {}
        variables = []
        l = {}
        s = {}
        for i in range(n+warehouses):
            for j in range(n+warehouses):
                if i==j:
                    continue
                x[i,j] = pulp.LpVariable('x_' + str(i) + '_' + str(j), 0, 1, pulp.LpInteger)
            if i >= warehouses:
                l[i] = pulp.LpVariable('l_' + str(i), 1, min(n,c), pulp.LpInteger)

        # Objective function
        z += pulp.lpSum([distances[i][j] * x[i,j] for i in range(n+warehouses) for j in
list(range(i)) + list(range(i+1,n+warehouses))])

        # Constraints
        constraintSeq = []
        constraintTrucks = []
        for i in range(n+warehouses):
            if i>=warehouses:
                constraintSeq.append(l[i])

            constraintFrom = []
```

```
            constraintTo = []
            for j in range(n+warehouses):
                if i==j:
                    continue
                if i>=warehouses and j>=warehouses:
                    z += pulp.lpSum([l[i], -1*l[j], n*x[i,j], -n+1]) <= 0

                if i>=warehouses:
                    constraintFrom.append(x[i,j])
                    constraintTo.append(x[j,i])
                if i<warehouses:
                    constraintTrucks.append(x[j,i])

            if i>=warehouses:
                z += pulp.lpSum(constraintFrom) == 1 # paths from location
                z += pulp.lpSum(constraintTo) == 1 # paths to location
            if i==warehouses and (paths > 1 or warehouses>1):
                z += pulp.lpSum(constraintTrucks) == paths # paths to warehouse

        if not slack:
            z += pulp.lpSum(constraintSeq) == total_l
        else:
            z += pulp.lpSum(constraintSeq) <= total_l

        # Solve
        if solver=="cbc":
            status = z.solve()
        if solver=="glpk":
            status = z.solve(pulp.GLPK())
        if solver=="gurobi":
            status = z.solve(pulp.GUROBI_CMD())

        # should be 'Optimal'
        if pulp.LpStatus[status]!="Optimal":
            print("RESULT: ".pulp.LpStatus[status])

        print("Objective function value: "+str(z.objective.value()))

        # Print variables & save path
        lines_x = []
        lines_y = []
        li = [0] * n
        for i in range(n+warehouses):
            if i>=warehouses:
                li[i-warehouses] = pulp.value(l[i])
            for j in range(n+warehouses):
                if i==j:
                    continue
                if pulp.value(x[i,j]) == 1:
                    lines_x.append(loc_x[i])
                    lines_x.append(loc_x[j])
                    lines_y.append(loc_y[i])
                    lines_y.append(loc_y[j])
                    lines_x.append(np.nan)
                    lines_y.append(np.nan)

        obj_value = "c=" + str(round(z.objective.value(),2))

    elif solver=="cut plane":
        model = Model("tsp")
        model.hideOutput()
        x = {}
        l = {}
        for i in range(n+warehouses):
            for j in range(n+warehouses):
                if i != j:
                    x[i,j] = model.addVar(ub=1, name="x(%s,%s)"%(i,j))
            if (paths > 1 or warehouses > 1) and i >= warehouses:
                l[i] = model.addVar(ub=min(c,n),lb=1, name="l(%s)"%(i))

        if paths == 1 and warehouses == 1:
```

```
            # SYMMETRIC DISTANCE MATRIX ONLY
            #for i in range(n+warehouses):
                #model.addCons(quicksum(x[j,i] for j in range(n+warehouses) if j != i) + \
                #            quicksum(x[i,j] for j in range(n+warehouses) if j != i) == 2,
"Degree(%s)"%i)

            # ASYMMETRIC DISTANCE MATRIX
            for i in range(n+warehouses):
                model.addCons(quicksum(x[j,i] for j in range(n+warehouses) if j != i) == 1,
"In(%s)"%i)
                model.addCons(quicksum(x[i,j] for j in range(n+warehouses) if j != i) == 1,
"Out(%s)"%i)

        else:
            for i in range(warehouses, n+warehouses):
                model.addCons(quicksum(x[j,i] for j in range(n+warehouses) if j != i) == 1,
"In(%s)"%i)
                model.addCons(quicksum(x[i,j] for j in range(n+warehouses) if j != i) == 1,
"Out(%s)"%i)

            for i in range(warehouses, n+warehouses):
                for j in range(warehouses, n+warehouses):
                    if i!=j:
                        model.addCons(l[i] -l[j] +n*x[i,j] <= n-1, "Li(%s,%s)"%(i,j))

            model.addCons(quicksum(x[j,i] for i in range(warehouses) for j in range(n+warehouses)
if i!=j) == paths, "Paths(%s)"%paths)

            if not slack:
                model.addCons(quicksum(l[i] for i in range(warehouses,n+warehouses)) == total_l,
"TotalL")
            else:
                model.addCons(quicksum(l[i] for i in range(warehouses,n+warehouses)) <= total_l,
"TotalL")

        model.setObjective(quicksum(distances2[i,j]*x[i,j] for (i,j) in x), "minimize")

        EPS = 1.e-6
        isMIP = False
        model.setPresolve(SCIP_PARAMSETTING.OFF)

        while True:
            model.optimize()
            #edges = []
            lines_x = []
            lines_y = []
            edges = []
            li = [0] * n
            for (i,j) in x:
                # i=j already skipped
                if model.getVal(x[i,j]) > EPS:
                    #edges.append( (i,j) )
                    lines_x.append(loc_x[i])
                    lines_x.append(loc_x[j])
                    lines_y.append(loc_y[i])
                    lines_y.append(loc_y[j])
                    lines_x.append(np.nan)
                    lines_y.append(np.nan)
                    edges.append( (i,j) )
            if paths>1 or warehouses>1:
                for i in range(warehouses, n+warehouses):
                    li[i-warehouses] = int(model.getVal(l[i]))

            obj_value = "c=" + str(round(model.getObjVal(),2))

            ax.clear()
            ax.set_xlim(0,1000)
            ax.set_ylim(0,1000)

            plot_data_lines(lines_x,lines_y)
            plot_data(ax, loc_x, loc_y, warehouses)
```

```
            ax.text(400, 1075, "solving...", family="serif", horizontalalignment='left',
verticalalignment='top')

            fig.canvas.draw()

            if addcut(edges,model,x,warehouses) == False:
                if isMIP:     # integer variables, components connected: solution found
                    break
                model.freeTransform()
                for (i,j) in x:     # all components connected, switch to integer model
                    model.chgVarType(x[i,j], "B")
                if paths > 1 or warehouses > 1:
                    for i in range(warehouses,n+warehouses):
                        model.chgVarType(l[i], "I")
                isMIP = True


        sol_li = [0] * (n+warehouses)
        sol_xij = {}

        print('solved.')

    elif solver == 'held-karp':
        li = [0] * n
        opt, path = held_karp(distances)
        print(path)
        obj_value = "c=" + str(round(opt,2))
        x = [[0 for x in range(n+warehouses)] for y in range(n+warehouses)]
        for idx, val in enumerate(path):
            if idx < (len(path)-1):
                x[val][path[idx+1]] = 1;
            elif idx == (len(path)-1):
                x[val][path[0]] = 1;

        for i in range(n+warehouses):
            for j in range(n+warehouses):
                if x[i][j] == 1:
                    #edges.append( (i,j) )
                    lines_x.append(loc_x[i])
                    lines_x.append(loc_x[j])
                    lines_y.append(loc_y[i])
                    lines_y.append(loc_y[j])
                    lines_x.append(np.nan)
                    lines_y.append(np.nan)
                    #edges.append( (i,j) )


    # Print computation time

    time2 = time.time() - start_time
    exec_value = time2
    units = 'secs'
    if time2 > 60:
        time2 /= 60
        units = 'mins'
    if time2 > 60:
        time2 /= 60
        units = 'hours'

    time2 = round(time2,2)

    exec_value = "exec=" + str(time2) + " " + units

    print("--- " + str(time2) + " " + units + " ---")

    # Redraw points
    ax.clear()
    ax.set_xlim(0,1000)
    ax.set_ylim(0,1000)

    plot_data_lines(lines_x,lines_y)
```

```
    plot_data(ax, loc_x, loc_y, warehouses)

    ax.text(350, 1075, obj_value, family="serif", horizontalalignment='right',
verticalalignment='top')
    ax.text(450, 1075, exec_value, family="serif", horizontalalignment='left',
verticalalignment='top')
    fig.canvas.draw()

axsolve = plt.axes([0.87, 0.905, 0.1, 0.075])
bsolve = Button(axsolve, 'Solve')
bsolve.on_clicked(solve)

plt.show()
```