Optimized Truck Router

James Currier

1 Project Summary

1.1 Current Problem

The current process for determining truck routes is extremely manual. One person, who has great knowledge and experience operating in the area, prints a page for each delivery that arrives and then groups them into routes based on previous knowledge and intuition. This person has many variables to consider when planning. These include delivery proximity, truck capacity, parking/ yard restrictions, driver restrictions, dangerous goods restrictions, and timing constraints.

Although this current process is working and we are meeting our goals, it is not sustainable. When the planner is absent, the terminal is left in chaos as there is no one else who can adequately do his job. In the event that a planner leaves the company, the learning curve for a new planner is extremely steep, which causes inefficiencies in the planning.

1.2 This Solution

Given all customers and restrictions, this software finds the best way to dispatch the lowest number of trucks, to the most efficient routes to make the deliveries. The program has access to a list of all of the delivery locations and their associated information, as well as a list of drivers and their restrictions (e.g. no heavy lifting).

Every day, the terminal manager is able to enter an Excel file containing the stops that need to be made throughout the day. Each of these stops would include consignee name, address, time windows, size, delivery method, and service type among other variables. The program finds efficient routes to complete all the deliveries.

The output of the program is an Excel file and an interactive map. The Excel file contains a separate sheet for each of the routes, and each of these sheets contains the route code, number of stops, truck type, and each of the stops of the route. The map shows all of the routes and gives information about each of the stops on each of the routes.

2 Inputs and Outputs

2.1 Inputs

The inputs to the program are all contained in one Excel File, which has sheets named "Delivery Locations", "Delivery Requirements", and "Drivers".

2.1.1 Delivery Locations

This sheet contains all the delivery locations and their restrictions. Ideally, this sheet should only be changed when a new location is added or a location changes its requirements. Each line represents a different location with "LOCATION NAME", "ADDRESS", "POSTAL CODE", "OPENING HOURS", "CUSTOMER ACCESSORIALS", "YARD/PARKING RESTRICTIONS", "AVERAGE DELIVERY TIME" and "MINIMUM PALLET COUNT".

LOCATION NAME: The name given to the location to deliver to.

ADDRESS: The address to deliver to.

POSTAL CODE: The postal code of the address to deliver to.

OPENING HOURS: The time window in which deliveries can usually take place.

CUSTOMER ACCESSORIALS: Additional items the location requires to complete their deliveries (e.g. a jack).

YARD/PARKING RESTRICTIONS: Any restriction that affects a certain truck's ability to access a delivery location.

AVERAGE DELIVERY TIME: The average amount of time it takes to make the deliveries. This can be filled in predictively or can be estimated using historical data.

MINIMUM PALLET COUNT: The minimum number of pallets needed to deliver. If this is not met, then don't make the delivery.

2.1.2 Delivery Requirements

This is the sheet that contains each of the stops. It must be completed every day.

Each line represents a delivery with fields "LOCATION NAME", "TRUCK TYPE", "REQUIRED EQUIPMENT", "SPECIFIC DELIVERY TIME", "DELIVERY METHOD", "PALLETS TO BE DELIVERED", "DANGEROUS GOODS", "SERVICE", and "PRO #".

LOCATION NAME: The name of the location to deliver to. Must match the name of a location on the "Delivery Locations" sheet.

TRUCK TYPE: The type of truck needed for this delivery.

REQUIRED EQUIPMENT: Any additional equipment needed for this particular delivery.

SPECIFIC DELIVERY TIME: A specific delivery window to deliver within. This overrides the "OPENING HOURS" column on the "Delivery Locations" sheet.

DELIVERY METHOD: The method of delivery. This impacts the type of driver that can make this delivery.

PALLETS TO BE DELIVERED: The number of pallets to be delivered.

DANGEROUS GOODS: "YES" if these are dangerous goods, "NO" otherwise.

SERVICE: The type of service. One of "REGULAR", "FRESH", "FROZEN", "HAZMAT".

PRO #: The PRO number of this delivery.

2.1.3 Drivers

This sheet contains a list of all the drivers and their restrictions. It must be updated when a new driver is added, or a driver gets a new restriction.

Each line represents a driver with fields “DRIVER ID”, “NAME”, “START TIME”, and “RESTRICTIONS”.

DRIVER ID: The driver’s unique ID number.

NAME: The driver’s name.

START TIME: The driver’s start time.

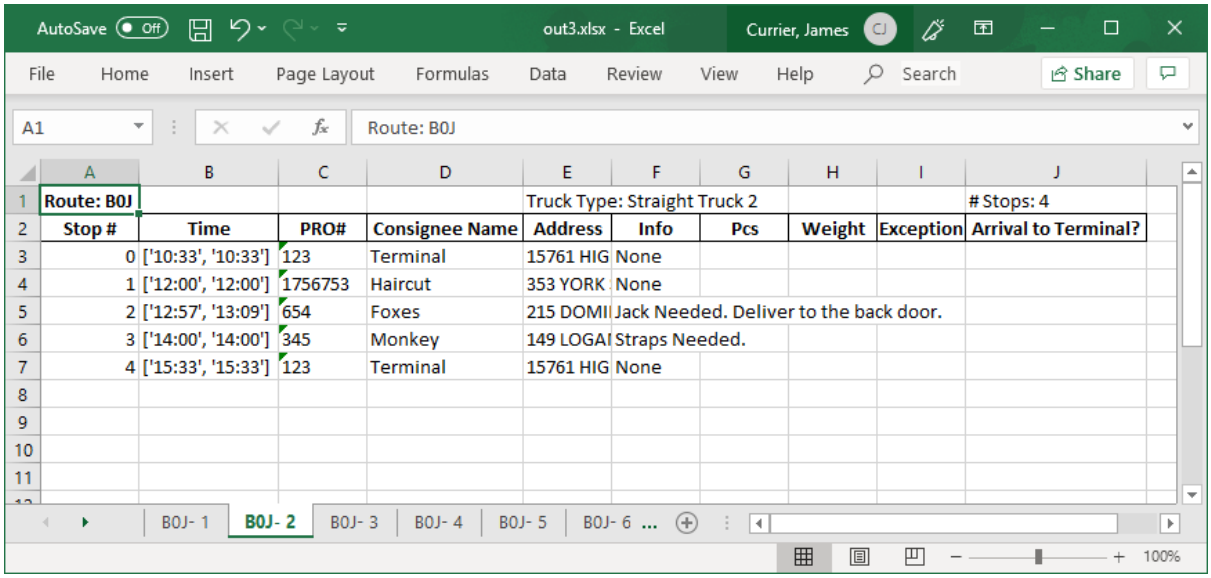
RESTRICTIONS: Any restrictions the driver has (e.g. no heavy lifting).

2.2 Output

The output is in two parts, an Excel file and an interactive map.

2.2.1 Excel File

The Excel file is the main form of output that will be used, and contains all the information needed to deploy a truck. This file has one sheet for each route and each sheet has a header section and a body.



Route: B0J				Truck Type: Straight Truck 2					# Stops: 4
Stop #	Time	PRO#	Consignee Name	Address	Info	Pcs	Weight	Exception	Arrival to Terminal?
0	['10:33', '10:33']	123	Terminal	15761 HIG	None				
1	['12:00', '12:00']	1756753	Haircut	353 YORK	None				
2	['12:57', '13:09']	654	Foxes	215 DOMI	Jack Needed. Deliver to the back door.				
3	['14:00', '14:00']	345	Monkey	149 LOGA	Straps Needed.				
4	['15:33', '15:33']	123	Terminal	15761 HIG	None				

The first line of the outputted file contains three parts: Route, Truck Type, and # Stops.

Route is a 4-letter code which represents the area in which the last delivery takes place. It is found using the first 3 characters of the postal code and is useful for conducting pickups after the truck is done deliveries.

Truck Type is the type of truck to be used for the delivery.

Stops is the total number of separate stops on the route (not just different PRO#s.

The body contains a line for each PRO#, and has the following attributes

Stop #: Which stop this is. By default, is in the order produced by the optimizer. The person using the tool can reorder the stops as they see necessary.

Time: The time window in which the driver should depart for this delivery.

PRO#: The PRO# of this delivery.

Consignee Name: Name of the organization that requested this delivery.

Address: The address to deliver to.

Info: Any additional info about the stop that the driver should know.

Pcs: Number of pieces to deliver.

Weight: The weight of the delivery.

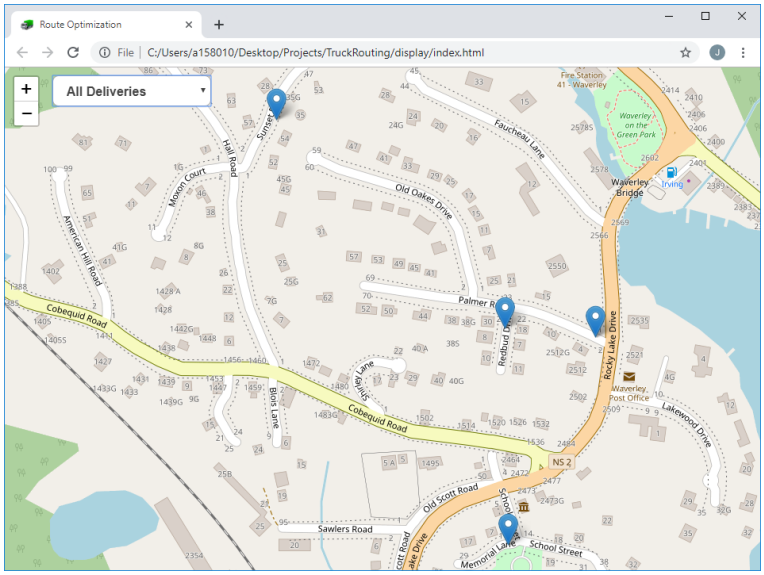
Exception: Blank column. Can be filled in by a worker to alert driver of any exceptions that occurred (e.g. An item is damaged).

Arrival to Terminal?: Whether the items to be delivered have arrived at the terminal.

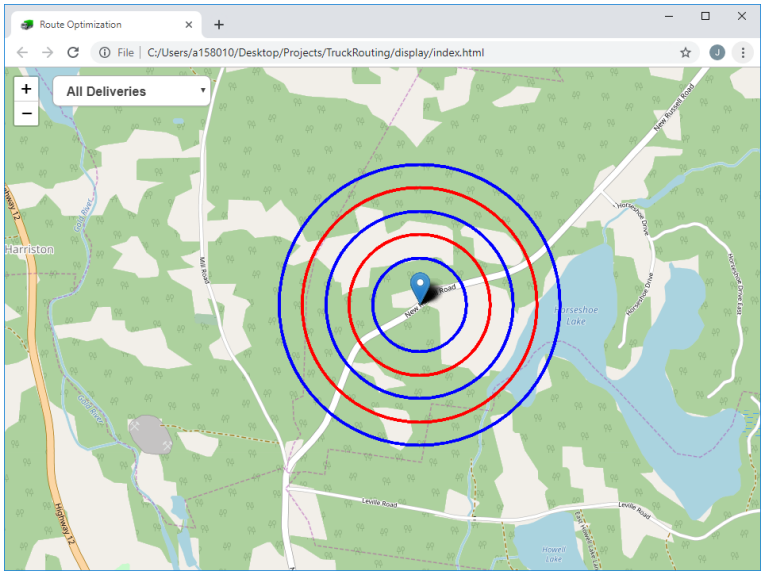
2.2.2 Interactive Map

The interactive map is a supplementary output which allows the user to quickly visualize where the routes are.

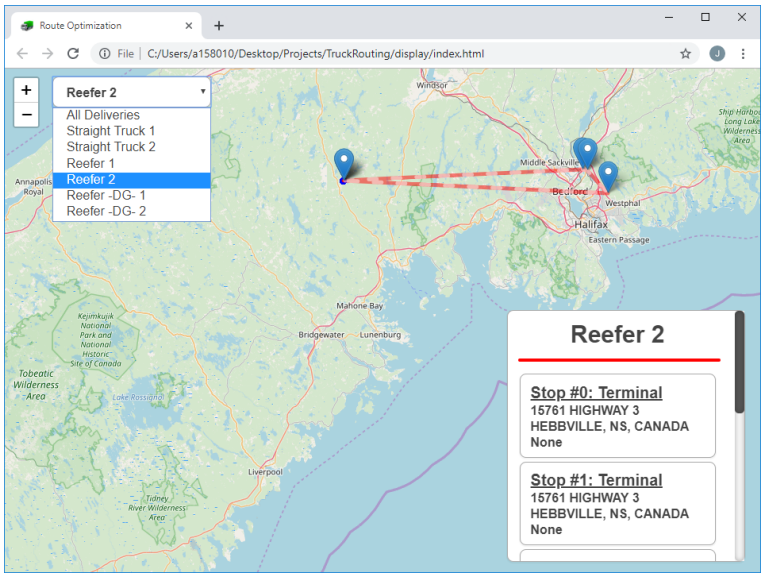
Each stop on the map is shown with a blue marker. Clicking on one of these markers causes a pop up to display with the stop name and address. When the route selector in the top left corner is set to "All Deliveries", all of the stops are displayed on the map. When the route selector is set to a specific route, only the deliveries on that route will be shown.



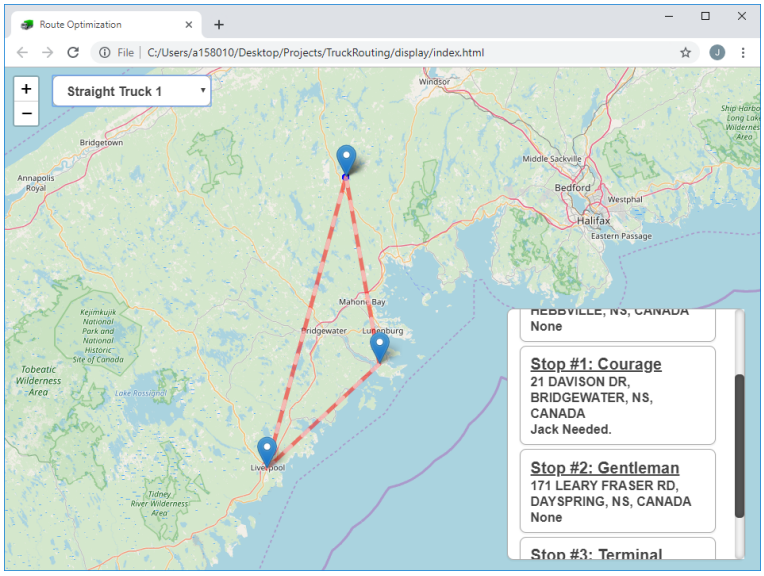
A terminal is represented by multiple red and blue circles around a marker.



Each route is represented by a dashing line. The direction of these lines represents the direction of the route. The corresponding line is shown when the route selector is set.



The additional information box sits in the bottom right corner of the display. When a route is selected, the window is populated with all the information about each stop on the route. If the amount of information causes the window to overflow, it can be scrolled independently of the rest of the display.



3 How It Works

The program can be separated into 5 different parts: Input Processing, Restriction Grouping, Time Calculating, Optimizing, and Outputting.

3.1 Input Processing

This part of the program takes in all required inputs. These are all inputted through a GUI.

3.1.1 CSV Data

There is a file upload section on the GUI which allows the user to upload an Excel file. After this file is uploaded, each sheet is read.

The deliveries that do not meet the minimum pallet count for their location are removed. A CSV file is created, which contains the augmented data of this file.

The fields in the CSV file are as follows

NAME: The stop name from the “Delivery Requirements” sheet.

LOCATIONINFO: The address and other location info for the stop; for display purposes.

TRUCKTYPE: The kind of truck that must be used.

POSTALCODE: The postal code of the stop.

COORDINATES: The coordinates of the stop. Found by a postal code lookup on a local file. If it is not found, the user is prompted to enter the coordinates.

WINDOW: The time window to deliver within. This is taken from the "Delivery Locations" sheet by default, but is overwritten by the value in the "Delivery Requirements" sheet if it is present.

ESTIMATE: The estimated amount of time for the delivery.

PALLETS: Number of pallets to be delivered

ADDITIONALINFO: Any additional info to display for this stop; combination of fields from the file.

PRO #: The PRO number for this delivery.

DANGEROUSGOODS: Whether these are dangerous goods.

3.1.2 Checkbox

There is a checkbox on the GUI which is checked if dangerous goods can be stored with fresh items. A Boolean value of this is passed directly through to the next part of the program as an optional argument.

3.2 Restriction Grouping

In this part of the program, stops are grouped together based on their restrictions. These groups are then all optimized separately.

If the dangerous goods are to be separate from the rest of the goods, then all of the stops

with the dangerous goods flag are removed and put in a separate group.

For each different kind of truck, a new group is made. All stops are then split up based on their truck type.

This is a very simple but effective method of dealing with many of the constraints.

3.3 Time Calculating

This step produces a time matrix representing the travel times between the stops. This is done for each group of trucks and will be used by the optimizer. Firstly, a base time matrix is found by one of the following methods. To account for the time it takes to complete the deliveries at each stop, each stop's delivery time is then added to the proper position on the time matrix.

There are a few different methods that can be used for obtaining the base time matrix

3.3.1 Heuristic Based

The direct distance in miles between two points is approximated by the Euclidean distance between the two coordinates multiplied by a constant. We can find this Euclidean distance for each pair of points and, by comparison with real travel times, we can estimate the constant.

This method has advantages and disadvantages. It is extremely fast, doesn't require internet, and can deal with a very large number of deliveries (up to 5000 in a reasonable amount of time). However, it is not very accurate, as it does not take into account the road lengths or speed limits between stops. Also, it is greatly affected by the curve of the earth; routes that are more north than others will be evaluated as having a much longer travel distance than in reality.

3.3.2 OSRM

OSRM or Open Source Routing Machine is a open source tool built on Open Street Maps which can be used to perform routing calculations.

One of the features of this is the Matrix API. It takes a URL with a list of coordinates and returns a matrix with travel times.

3.3.3 PC Miler

PC Miler is an analytics tool for trucking which we already have a license for. It has a Route Matrix feature which provides the same functionality as OSRM but is even more accurate, as it is optimized for trucking routes.

3.4 Optimizing

This step is where the optimization actually takes place. This step uses a time matrix and list of time windows to produce efficient routes.

The optimizer used is Google OR Tools' VRP solver. This uses a combination of different optimization techniques, namely Local Search methods, Dynamic Programming, and in some cases Exhaustive Tree Search.

The optimizer is given the time matrix, time windows, number of vehicles and maximum wait time by stop, and produces optimized routes. Empty routes are removed, and this is then repeated for all other groups of stops.

The results of this optimization are written to a file in JSON format, as well as a JavaScript file to be used as a source.

3.5 Outputting

The final step involves outputting the results to an Excel File and an interactive map.

3.5.1 Excel File

All of the information in this file is derived from the JSON file, and is written to Excel by pandas' ExcelWriter. A separate sheet is made for each route in the JSON file.

The Route code is generated by comparing with the first three characters of the route's last stop's postal code. These characters are mapped to a route code using a given relation.

The Truck Type is read directly from the JSON file.

The # Stops is calculated by counting the number of stops with distinct addresses. This is different from the total number of stops if any locations have multiple deliveries.

The body of this file is pulled directly from the JSON file, and then formatted as necessary.

3.5.2 Interactive Map

The interactive map is made using Open Street Maps and Leaflet. The results of the optimization are read through a JavaScript file as a source which creates a RESULTS variable with all of the needed information.

The base map of the tool is from Open Street Maps.

Each of the stops on the map are drawn on using a Leaflet Marker object.

The lines between the stops are drawn using the ant-path library for Leaflet.

The rest of the interactive tool, including the information window, are created using vanilla HTML/CSS/JavaScript.

4 Development Process

4.1 Model

When looking for a model to use for the optimizer, we were looking for a model with good performance that was easy to use.

Google OR Tools seemed to have the perfect tool for us. Their Routing Optimizer uses state of the art techniques for optimization, it's free, open source, and has detailed examples of how to use it.

4.2 Prototype

The first prototype was simply a Python script that ran a base example from OR Tools, with a toy time matrix. A JavaScript program reads the file, and displays the results on a simple map using Open Street Maps and Leaflet. This first prototype was very basic, and didn't really have much functionality, but provided an outline of what the end product could be if we continued.

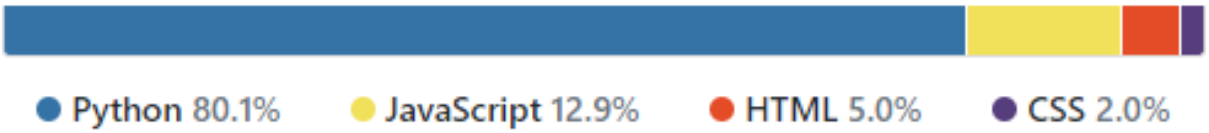
4.3 Business Requests

After presenting the prototype to Midland, everyone seemed on board with the idea, which pushed the project forward. There were some concerns raised about the project, however, and certain requirements that we would need to have for the tool to be useful. These requirements included multiple truck types, dangerous goods separation, Excel output, ability for user intervention, and time window restrictions.

4.4 Programming Tools

The programming languages used for this project were primarily Python and JavaScript.

The following represents the usage of languages in the project



Many non-standard libraries were also used.

Google's OR Tools (Python): Used to perform the optimization.

Pandas (Python): Provides an Excel interface for Python.

PyQt5 (Python): GUI library used in our program's GUI.

OSRM (API): Provides an API to calculate travel times between points.

Open Street Map (API): Used to display the map on our tool.

Leaflet (JavaScript and CSS): Used for customizing and adding visuals to interactive map.

Leaflet Ant Path (JavaScript): Used to create the dashing route lines between stops.

5 Future Improvements

Although many of the requirements for this tool are already implemented, there are some additional features that must be added for the tool to really be effective.

5.1 Truck Capacity

We currently have no perfect way of finding exactly how much we can fill any particular truck. This makes it difficult to optimize, as there could be some trucks that have extra space, or some trucks that don't have enough space for the deliveries it was assigned.

We are currently using a combination of the following methods, but they all have some

issues.

Capacity by Weight: Give each truck weight range that it can be filled to and optimize to meet this range. This method could work, but with more dense or less dense deliveries, this could fail.

Capacity by Cubic Size: Fill the truck based on the cubic size of each of the deliveries. This would be a perfect metric if the cubic size was accurate and available for all deliveries, however, currently it is not.

Capacity by Number of Items: Assign each truck type a maximum number of items it can carry. This is the best metric we currently have, and it works very well when the items are the same size (e.g. pallets), but it fails when the deliveries are different sizes.

5.2 Ease of Deployment

Currently, there is no great way to deploy our solution. We need to put it somewhere online (likely GitHub or BitBucket) and have the user download it to use it.

A server would be the ideal way to deploy our solution. This would allow it to run the software we need to use the tool, as well as give us the ability to give permissions to use the tool as we see fit.

5.3 Optimizing Pickups

In Midland's process, they complete deliveries in the morning, and pickups in the afternoon. After the trucks do their deliveries, they begin doing pickups near their last delivery. Midland deploys trucks with final stops near pickups to cut down on driving cost.

With a list of these pickups, we could optimize the deliveries and pickups together to reduce total driving time.

5.4 Predicting Pickups

Many of the pickup requests that Midland receives are spontaneous. They receive the request for pickup within the same day, sometimes the same hour, that the pickup will occur. They are currently dealing with this by placing trucks near where they predict these pickups will happen.

Based on previous data, we could build tool that predicts when and where these requests will happen. This would provide more accuracy, and require less time than their current solution for prediction, which is essentially just intuition.

5.5 Automation

In its current state, to run this program the user needs to enter all of the data in an Excel file, click the run button, and save the Excel sheet.

Automating this process even further could be beneficial by saving even more time and effort. We could pull the delivery information out of Midland's system as requests come in, run the tool automatically, and save the Excel file in a local location or distribute it as necessary (e.g. email).