Open-Route Documentation

Release

Sean Kelley

Aug 18, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

This documentation explains the code base and mathematical models powering Open-Route. If you haven't checked out the website yet, I welcome you to go to http://www.open-route.website and take a look. Clicking through the website takes less than a minute and will make the following documentation much more coherent!

Originally a package that I created for modeling the movement of identical sets of large construction equipment ('thus identical items that fill the truck's capacity exactly' on the website), the software behind Open-Route has been repurposed for a general application of scheduling and vehicle routing in a delivery network. Open-Route hopes to benefit anyone moving large and identical items through a network by providing the following solutions:

- What days should items be dropped-off/picked-up
- How many trucks are needed each day
- What routes does each truck take each day

1.1 Github

If you'd like to follow along in the code for which this documentation was created, I invite you to check out the github repository. The remaining chapters have a "Documentation" section detailing where specifically in the code base you can find what is explained here.

1.2 Topics

I'll cover a few different topics throughout this documentation in order to describe the entire approach to the solutions Open-Route provides.

- *Set-Up Horizon* covers handling input data and adjusting drop-off/pick-up dates. This part is where everything is set up that will remain constant for the daily routing calculations.
- *Daily Routing* explains how the daily routing calculations are made - how many trucks are needed and what routes does each take.
- *Reporting Results* outlines how the graphs are generated for returned requests from the web app.
- In *Conclusion*, I'll give my closing remarks on the project and a brief bit about myself.

I hope you enjoy reading, and thank you for taking the time to check out my work!

Sean Kelley

Operations Research Analyst and Software Developer

CHAPTER

TWO

SET-UP HORIZON

Each request that Open-Route processes follows three steps: collecting data that will be fixed for each day, making calculations specific to each day, and reporting the results of all calculations made. This section covers capturing the fixed data - the set-up that is required for being able to solve for all days.

2.1 Fixed Inputs

Regardless of which day the process is routing trucks, there are a few inputs that remain constant from the time the user sets them. They are as follows:

- the average rate of travel of the trucks
- the length of a working day for a truck driver
- the average time it takes an item to be unloaded from or loaded onto the truck
- the number of days (time window) in which a site is elgible to have items dropped-off or picked-up for each instance of demand
- the geographical coordinates of each site in our network - sites having items picked-up/dropped-off and the hub where trucks start/end their days

Demands - how many items are requested for drop-off or pick-up on a day at a site - will also remain fixed throughout the request, with the exception of the smoothing operation that may adjust those days prior to the daily calculations. The rest of this section will be devoted to explaining that.

2.2 Importance of Time Windows

In order to make sure the dates items are dropped-off/picked-up are feasible, we need a sufficient number of trucks and items available in our network. To give a preference to having more items or more trucks in our network, I implemented time windows into the calculations. For example, prescribing that site must have its items dropped-off and picked-up the day they're demanded will result in a couple extremes. It will minimize the number of items needed to meet all sites' demands (as it minimizes the amount of items sitting unused at sites prior to/after being needed) but will maximize the number of trucks needed to satify the network as some days may require many more item movements than others. Adding just a day to the time window (increasing it from 1 day to 2) can dramatically decrease the number of trucks required without having much effect on the number items. Having (hopefully) assigned the time window as optimally as possible, let's move on to smoothing the demand in the item network.

2.3 Demand Smoothing Model

I approached demand smoothing by formulating an integer program. It's parameters are sites, i, days, l, and sets, $S_{i,l}$, of alternative days that drop-offs/pick-ups can be made for site i with original demand on day l. Each site, i, for each day, l, has a demand, $d_{i,l}$, stating how many drop-offs or pick-ups are required, with $d_{i,l} < 0$ for drop-offs and $d_{i,l} > 0$ for pick-ups. A set, $S_{i,l}$, is empty unless $d_{i,l} \neq 0$. In that case $S_{i,l}$ is the set of the n days before and including the drop-off date or the n days after and including the pick-up date, where n is the number of days in our time window. (If there are fewer than n days until the start or the end of our days, the set is just the remaining days.)

I used two variables in this problem. The first, $w_{i,l}$, represents the number of items going to or from site i on day l. The second, z, is the largest sum of items going to or from all sites on any single day, l. We can now form the following integer program:

$$\begin{aligned}
&\text{minimize} && z && && (1)\\
&\text{s.t.:} && && && \\
&\sum_{l' \in S_{i,l}} && w_{i,l'} = |d_{i,l}| && \forall\, i,l : S_{i,l} \neq \{\} && (2)\\
&\sum_{i} && w_{i,l} \leq z && \forall\, l && (3)\\
& && w_{i,l}, z \in Z \geq 0 && \forall\, i,l && (4)
\end{aligned}$$

(1) tells us that we are minimizing z, the maximum number of pick-ups and drop-offs to be done on any single day. This is enforced by (3), stating that our objective, z, must be greater than or equal to any single day's sum of drop-offs and pick-ups. (2) adds that for any site, i, on any day, l, with a demand for drop-offs or pick-ups, the number of items going to or from that site within the time window must equal the number of items originally demanded by site i on day l. Our last constraint, (4), ensures all our variables are non-negative integers.

Put in plain English, (2) allows us to assign pick-ups and drop-offs to different days as long as its sufficiently close to the original day they were requested, and (1) and (3) ensure the number of trips being made each day are as few as possible. (4) simply prevents any half- drop-offs or half-pick-ups from being made.

With the number of drop-offs and pick-ups to be made to each site each day now smoothed, the inputs for each day are fixed and ready to be passed to day-specific calculations

Continue to *Daily Routing*

2.4 Documentation

Adjusting/gathering fixed input:

`iterate.`**`solve_horizon`**(*fixed_parameters*, *demand_df*)

Find truck, hauler, and equipment usages for all days in our range.

Finds and smoothes delivery demand for all days. Determines day by day usage of all assets. Creates report detailing usage over whole range.

Parameters

- **`fixed_parameters`** (*`dict`*) – Parameters that are constant for any variation and region (as defined in the main function)
- **`demand_df`** (*`pandas.core.frame.DataFrame`*) – demand for number of drop-offs or pick-ups that each site has for each day. This df has already been smoothed for the web edition

Creating the set, S, and calling the smoothing integer program:

smoothing.**iterate**(*period_inputs*, *current_start_index*)

Smoothes 'period' days of demand data

Pulls the original demand data for the next 'period' days to be smoothed. Creates a matrix of possible days each site can have drop-offs or pick-ups, constrained by the 'window' and the 'period' of days being considered. Smoothes 'period' days of demand and records the new number of drop-offs and pick-ups to be made to each site each day in this period.

Parameters

- **period_inputs** (*dict*) – The variables recording how 'smooth' the demand being adjusted 'period' days at a time can be. Includes how many days are in a period, how many days are in our window, the dataframe recording demand, an array storing how much demand each day has after smoothing, the largest demand seen for any one day, and a flag for if this period length returns a feasible solution for every 'period' days smoothed.
- **current_start_index** (*int*) – The column of the demand dataframe from which the next 'period' days of demand will begin to be smoothed

Returns **period_inputs** – Inputs recording statistics on the smoothed demand and the smoothed demand itself are saved to their respective variables and returned.

Return type dict

Demand smoothing integer program:

smoothing.**smoothing_model**(*d*, *s*)

The integer program responsible for smoothing 'period' days of demand

Minimizes the total number of demand for any one day, while ensuring all demand for each site is met within the time window.

Parameters

- **d** (*numpy.ndarray*) – The number of drop-offs or pick-ups each site needs each day
- **s** (*list*) – Three dimensional list describing the alternative delivery dates each site has for each day it has demand. If a site has no demand on a given day, its list of alternatives is empty.

Returns **results** – Whether or not the IP solved to optimality, what the largest demand for the period was after smoothing, and the newly assigned demands to each site each day.

Return type dict

Wrapper for demand smoothing functions:

smoothing.**smooth_demand**(*demand_df*, *window*, *start_date*, *end_date*)

Smooth the demand for drop-offs and pick-ups for a given variation as much as possible constrained to the time window

Defines a list of period lengths. For each period length, demand is smoothed 'period length' days at a time. Returns the smoothed demand for the entire time range corresponding to the length of period which results in the lowest variance in total daily demand.

Parameters

- **demand_df** (*pandas.core.fame.DataFrame*) – dataframe of demands to be smoothed
- **window** (*str*) – The number of different days to allow a job site to have equipment dropped-off or picked-up

- **start_date** (*str*) – The first day in our range of time we are considering ('yyyy-mm-dd' format)
- **end_date** (*str*) – The last day in our range of time we are considering ('yyyy-mm-dd' format)

Returns **mv_demand_df** – The minimum variance demand dataframe is the demand for each job site each day, spread as smoothly as possible, constrained to the size of our window

Return type pandas.core.frame.DataFrame

Continue to *Daily Routing*

CHAPTER

THREE

DAILY ROUTING

Knowing how many drop-offs and pick-ups are required each day at this point, I tried to route all trucks throughout the time range in one integer program. This proved quite unsuccessful, however, as the model was so large it could not run in a feasible amount of time. After some trial and error, I discovered that truck routing could be done very efficiently one day at a time, thus the daily calculations.

3.1 Creating Parameters

Before solving for the routes each truck will take each day, we need to create a few parameters our routing integer program will need in order to run. They are the following:

- **locations,** i A list of all n sites with demand for drop-offs or pick-ups on a given day. Sites with no demand are excluded from this list. Prepended and appended to this list is the hub, where the trucks start and end their day. The hub has multiple indices, 0 and $n + 1$ but represents the same physical location.
- **demand,** d_i The number of drop-offs or pick-ups each location, i, needs on a given day, ignoring the sites with no demand.
- **customers,** i A list of locations minus the hub. This list has length n.
- **route constraints,** $D_{i,j}$ A matrix of the number of times the route from site i to site j can be traveled, indexed in the same order as the locations parameter. Forces the model to adhere to these real world constraints:
  - A truck must alternate between dropping-off items and picking-up items. This prevents item pick-ups being satisfied by a truck that is already carrying an item and drop-offs being satisfied by a truck that is empty.
  - A truck can only use the demand from one site to satisfy an opposite demand at another as many times as is the minimum absolute value of demand between the two sites. This prevents sites of lesser demand from sourcing sites with greater demand more than they are capable, forcing trucks to return to the hub or less convenient sites for replenishment once the lesser demanded site has been satisfied.

  This assignment is very particular. Please see the documentation below (make_route_constraints) and its definition in the parameters.py module in the github repository for more details.
- **travel,** $t_{i,j}$ A matrix stating how many miles apart site i is from site j indexed in the same order as the locations list. The distances are calculated by converting the differences in geographical coordinates listed for each site.
- **subsets,** S_m A list of the even-sized subsets, m, of sites with demand on a given day, necessary for forcing continuity in our trucks' routes.
- **M** An arbitrarily large number.
- **trucks,** k A list of the trucks we have available.

Also recall a few parameters fixed upon input.

- rate of travel, r
- length of working day, l
- average time to load/unload a truck (handling time), h

3.2 Developing the Routing Model

Lastly, the problem makes use of two variables. The first variable is $x_{i,j,k}$, the number of times truck k takes the route from site i to site j. The other needed variable is $y_{m,k}$, whether or not a truck, k, enters the subset of points m.

Now that we have declared all of our parameters we will need to solve our truck routing for each day, we can define our model[1] .

$$\min \sum_i \sum_j \sum_k t_{i,j} x_{i,j,k} \tag{1}$$

s.t.:

$$\sum_{j \setminus \{0\}} x_{0,j,k} \geq 1 \quad \forall\, k \in \text{trucks} \tag{2}$$

$$\sum_i x_{i,h,k} - \sum_j x_{h,j,k} = 0 \quad \forall\, h \in \text{customers}, k \in \text{trucks} \tag{3}$$

$$\sum_i x_{i,n+1,k} = 1 \quad \forall\, k \in \text{trucks} \tag{4}$$

$$\sum_i \sum_j x_{i,j,k}(h + \frac{t_{i,j}}{r}) \leq l + h \quad \forall\, k \in \text{trucks} \tag{5}$$

$$\sum_j \sum_k x_{i,j,k} =\mid d_i \mid \quad \forall\, i \in \text{customers} \tag{6}$$

$$\sum_k x_{i,j,k} \leq D_{i,j} \quad \forall\, i,j \in \text{locations} \tag{7}$$

$$\sum_{i' \in S_m} \sum_{j' \in S_m} x_{i',j',k} \leq M(y_{m,k}) \quad \forall\, k \in \text{trucks}, m \in \text{subsets} \tag{8}$$

$$\sum_{i' \in S_m} \sum_{j \setminus S_m} x_{i',j,k} \geq y_{m,k} \quad \forall\, k \in \text{trucks}, m \in \text{subsets} \tag{9}$$

$$x_{i,j,k} \in Z \geq 0 \quad \forall\, i,j \in \text{locations}, k \in \text{trucks} \tag{10}$$

$$y_{m,k} \in \{0,1\} \quad \forall\, k \in \text{trucks}, m \in \text{subsets} \tag{11}$$

(1) tells us our objective is to minimize the total amount of distance that our fleet of trucks cover each day. $(2) - (4)$ add constraints for modeling the travel from one site to the next. (2) says that each truck must leave the hub each day, even if just to return directly to it at no cost (equivalent to not being used). If a truck arrives at a site h to make a drop-off or a pick-up, (3) ensures that truck also leaves that site for another. Once a truck has made all drop-offs and pick-ups it needed to make for a day, (4) requires that it returns to the hub. (5) brings our time constraints into play. We required that all driving and unloading/loading (un/loading) of trucks must be less than the length of the allowable work day. Time for an extra handle was added under the assumption that each truck needed no adjustments at neither the start nor end of its day by its driver. (6) covers our demand constraint, the requirement that each site, i, must be visited exactly as many times as it needs items dropped-off or picked-up each day. To ensure that (6) only counts the visits that correspond to the type of demand the site has, (7) applies the route constraints that force a truck's visit to a

[1] (2) - (4), as well as inspiration for the rest of the constraints, originate from the "Vehichle Routing Problem with Time Windows" chapter of *Column Generation* by Desaulniers, Desrosiers, and Solomon.

site be one where it is able to satisfy one unit of that site's demand. $(8) - (9)$ are additions to $(2) - (4)$, enforcing that trucks take routes that are only possible in the real world. (8) monitors whether or not a truck enters a given subset of sites, while (9) requires that a truck leave that set if it entered it. These constraints remove the possibility that the routes a truck takes in a day are disconnected. Lastly, $(10) - (11)$ define the spaces for our variables, specifically that trucks can take no partial routes and either enter or do not enter any given set of sites.

3.3 Running the Routing Model

One final piece of information remains to fully define our integer program for truck routing: how many trucks are needed on a given day. Trying to use as few trucks as possible each day, I started by running the model with as few trucks as possible, one, and rerun it, adding one truck per rerun, until the program is feasible. Once feasible, the following is recorded:

- how many hours each truck driver worked
- what routes each truck covered

Once this process has been repeated for each day given in our range of time, we can then summarize the data we've recorded to understand the capital required to meet all sites' demands over the horizon.

Continue on to *Reporting Results*

3.4 Documentation

Daily routing wrapper function:

`iterate.`**`solve_day`**(*fixed_parameters*, *daily_inputs*)

Determine the usage of semi-trucks and equipment haulers for a given day.

Creates parameters specific to solving a given day's truck and hauler usage. Uses extension of a Vehicle Routing problem to determine how many semi-trucks/haulers were needed, how many miles were driven, how long each hauler works, and what routes each hauler took each day.

Parameters

- **`fixed_parameters`** (*dict*) – Parameters that are constant for the whole horizon (as defined in the main function)
- **`daily_inputs`** (*dict*) – inputs needed each day to make remaining parameters and record the outputs of our routing model

Returns

- **fleet_mileage** (*numpy.ndarray*) – How many miles a fleet of a given size runs on a given day
- **hauler_hours** (*numpy.ndarray*) – How many minutes each hauler works each day
- **hauler_routes** (*OrderedDict*) – What routes each hauler took each day

Creating parameters wrapper function:

`parameters.`**`make_parameters`**(*fixed_parameters*, *daily_inputs*)

Create the remaining parameters (all of which vary by day) to solve our daily routing problem

Parameters

- **`fixed_parameters`** (*dict*) – Parameters that are constant for any variation and region (as defined in the main function)

- **daily_inputs** (*dict*) – inputs needed each day to make remaining parameters and record the outputs of our routing model

Returns **variable_parameters** – The parameters that vary by day but are still needed for our model to run

Return type dict

Demand:

parameters.**make_demand_list**(*daily_demand*)

Converts our daily demand from a pandas series to a list and adds the demands (0 demand) for where haulers start and end their days

Parameters **daily_demand** (*pandas.core.series.Series*) – How much demand each site has on a given day, given that the site needs at least one drop-off or one pick-up

Returns **demand_list** – Demands for all locations (job sites with demands and the hub) to be included on our graph for the day's hauler routing

Return type list

Route constraints:

parameters.**make_route_constraints**(*demand_list*)

Makes a matrix detailing how many times the route from site i to site j can be travelled by all haulers in one day

The following constraints apply to the number of times the route from i to j can be travelled. Travelling the route from i to j never happens if they both need drop-offs, both need pick-ups, or i is where a hauler ends his day. If i and j have opposite demand types (one with drop-offs, the other pick-ups), i to j can be travelled as many times as the minimum absolute value of demand had by the two sites. We place no constraint on how many times a hauler can return to the hub to reload his trailer.

Parameters **demand_list** (*list*) – Demands for all locations (job sites with demands and the hub) to be included on our graph for the day's hauler routing

Returns **route_constraints** – The number of times the route between any two locations can be travelled by all haulers

Return type numpy.ndarray

Travel:

parameters.**make_travel_matrix**(*daily_demand*, *site_df*, *travel_rate*, *day_length*, *handle*)

Makes a matrix describing how long the route from location i to location j is

Takes the difference in coordinates for each location to be visited on a given day and converts them to mileage. Rounds routes that are greater than one day's worth of miles to the max miles that can be done in one day.

Parameters

- **daily_demand** (*pandas.core.series.Series*) – How much demand each site has on a given day, given that the site needs at least one drop-off or one pick-up
- **site_df** (*pandas.core.frame.DataFrame*) – The latitude and longitude for each of our sites
- **travel_rate** (*float*) – How many miles per minute a hauler can drive on average
- **day_length** (*int*) – How many minutes per day a hauler can work
- **handle** (*int*) – How long it takes on average for a hauler to unload or reload his trailer

Returns **travel_matrix** – How many miles the route from location i to location j is

Return type numpy.ndarray

Subsets:

parameters.**make_subsets**(*customers*, *demand_list*)

Make all even sized subsets of customers (job sites) to be visited each day, excluding those where all customers have the same kind of demand (all drop-offs or all pick-ups)

Needed for ensuring the routes each hauler makes this day are all connected

Parameters

- **customers** (*list*) – A list of the indices corresponding to each job site with a demand on a given day
- **demand_list** (*list*) – Demands for all locations (job sites with demands and the hub) to be included on our graph for the day's hauler routing

Returns **subsets** – The list of all even sized subsets of customers where both types of demand are present

Return type list

Model implementation:

hauler_routing.**route_fleet**(*fixed_parameters*, *variable_parameters*, *haulers*)

An Integer Program for determining if a given sized fleet of equipment haulers can feasibly meet the demand for drop-offs and pick-ups in a given day

Parameters

- **fixed_parameters** (*dict*) – Parameters that are constant for any variation and region (as defined in the main function)
- **variable_parameters** (*dict*) – The parameters that vary by day but are still needed for our model to run
- **haulers** (*list*) – A list of indices for all available equipment haulers in the fleet

Returns **results** – Returns whether or not the IP solved to optimality, the total number of miles run by the fleet, and the number of times each hauler ran each route available to be travelled this day.

Return type dict

Continue on to *Reporting Results*

CHAPTER

FOUR

REPORTING RESULTS

Upon completion of the truck routing for every day in our data set, we are left with three complete pieces of information as our results.

- how much demand each site had each day (smoothed)
- what routes the trucks took each day
- how many hours each hauler worked each day

We saw earlier in the docs how to come up with the smoothed version of demand. All that is left with demand is to put the input side by side with the smoothed so the changes are easy to see, as is done exactly on the top of the return page of Open-Route. We've yet to cover how we interpreted the hours each driver works and routes each truck takes each day. I'd like to take a second to do that now.

4.1 Interpreting Daily Hours and Routes

Recall our variable for number of times a route from site i to j was covered by truck k, $x_{i,j,k}$. Also remember $t_{i,j}$ was our travel distance between two sites and r was our travel rate. With those three piece of information, the following gives us how long each truck driver works each day:

$$\sum_i \sum_j x_{i,j,k} * t_{i,j} * r \; \forall \; k \in \text{trucks}$$

Recording the time each driver works each day, we can then plot those hours into the "Truck Utilization" graphs seen in the middle of the response page. Worth noting, the highest indexed truck driver is assigned the longest set of routes to run each day, in order to have to use extra drivers as little as possible. Once we've decided which truck drivers match up to which of our k drivers in our model, we then record the routes they ran for that day by which $x_{i,j,k} > 0$.

Lastly, summary stats on the utilization of each truck driver are printed out to help add a little more detail to their workload throughout the week. With that we now have all the data and graphs we need to generate the return page. That's it for Open-Route's documentation, but I welcome you to read on to my comments on the project or brief biography on myself.

Continue to *Conclusion*

4.2 Documentation

Summary of driver mileage accumulated and routes taken on a given day:

`recording.`**`record_fleet_mileage`**(*fleet_size*, *date_index*, *fleet_mileage*, *objective*, *fleet_upper_bound*)

Records the total mileage run by the fleet each day

Assigns the mileage to all possible fleet sizes greater than or equal to the minimum fleet size.

Parameters

- **fleet_size** (*int*) – The number of haulers in the fleet for this iteration
- **date_index** (*int*) – The index of the date for which is currently being solved
- **fleet_mileage** (*numpy.ndarray*) – The total number of miles that a fleet of a given size would need to run each day to meet all of the demand.
- **objective** (*int*) – The total number of miles the fleet ran to meet all demand on this day
- **fleet_upper_bound** (*int*) – The maximum number of haulers that can be available on one day

Returns **fleet_mileage** – The total number of miles that a fleet of a given size would need to run each day to meet all of the demand.

Return type numpy.ndarray

Summary of time truck driver work each day and the routes they take:

recording.**record_hauler_hours**(*hauler_hours*, *hauler_routes*, *variables*, *handle*, *travel_rate*, *fleet_size*, *date_index*, *locations*, *travel_matrix*, *daily_demand*)

Records the number of minutes each hauler works each day

Sums the time taken by each hauler to run all of his assigned routes in a day. Gives the greatest amount of work to the lowest indexed haulers. Restricted to there being 10 or fewer locations on a graph for each day.

Parameters

- **hauler_hours** (*numpy.ndarray*) – How many minutes each hauler works each day
- **hauler_routes** (*dict*) – Running list of the routes that each equipment hauler runs each day
- **variables** (*dict*) – The number of times each hauler travels each route on a given day
- **handle** (*int*) – How long it takes on average for a hauler to unload or reload his trailer
- **travel_rate** (*float*) – How many miles per minute a hauler can drive on average
- **fleet_size** (*int*) – The number of haulers in the fleet for this iteration
- **date_index** (*int*) – The index of the date for which is currently being solved
- **locations** (*list*) – The indices of all sites the haulers will visit on a given day (including the hub)
- **travel_matrix** (*numpy.ndarray*) – How many miles the route from location i to location j is
- **daily_demand** (*pandas.core.series.Series*) – The sites with demand on a given day and their corresponding demand

Returns **hauler_hours** – How many minutes each hauler works each day

Return type numpy.ndarray

Bar graph for daily hours of truck driver .. autofunction:: reporting.hauler_graph_maker

Report generator:

reporting.**make_report**(*data*, *fixed_parameters*)

Creates input for a report of truck and equipment usage over the time range

Details miles traveled by the fleet of trucks, utilization rates for each equipment hauler, and utilization rates for each set of equipment.

Parameters

- **data** (*dict*) – A dictionary containing daily site demands, truck mileage totals, and hours worked by each hauler
- **fixed_parameters** (*dict*) – Parameters that are constant for any variation and region (as defined in the main function)

Summarization of truck driver utilization:

reporting.**summarize**(*df*)

Computes summary statistics for each equipment hauler

Statistics include the following: total hours worked by each hauler in the time range, number of days each hauler recorded time working, proportion of working days each hauler actually worked, average hours each hauler records on days he works

Parameters **df** (*DataFrame*) – How many minutes each hauler works each day (from hours_df)

Returns **summary** – A dataframe with the summary statistics outlined above

Return type pandas.core.frame.DataFrame

Continue to *Conclusion*

CHAPTER

FIVE

CONCLUSION

5.1 Possible Improvements

Looking back on the work I did creating Open-Route's software package with the construction company, there were a couple avenues I wish I would have had the time to explore that I think would have led to better results for the construction company. At the time of making Open-Route, almost a year later, the first two remain to be improved, as they're why I want to go to graduate school, but I have made progress on the third.

The first had to do with the smoothing integer program. The goal of the model was to minimize the variation amongst the days. Given that variation is a second-order function, I would have liked to have had the time to research approaches to smoothing demand with an Quadratic Program. If the objective of our smoothing function could have been quadratic, I could have been able to minimize the overall variation, resulting in a tighter smoothing of all the days' demands.

I also would have liked to have cleaned up the routing integer program as well. Constraints $(8) - (9)$ despite solving a key issue with the model, were very computationally expensive as they grew exponentially relative to the number of sites that had demand for a given day. Given how small the problem size was for each day (only 5 locations), dealing with exponential constraint growth was not an issue. However, finding a way to model the problem to the same effect but without the exponential constraints would be necessary for scaling the number of locations for which we could solve at one time.

The one area for improvement for that project I had wanted to make that Open-Route solved was another issue in scalability - scaling so that more people can use it more easily. The last project read from fragile excel files and had to be run from the command line. I think providing a structured form and GUI via a web app makes the project much more user friendly and useful to anyone who would want to try it out.

5.2 Biography

A brief bit about me. At time of writing, I'm a studying Industrial Engineer at the University of Illinois. My interest areas lie in Operations Research and Computer Science. I really enjoy Operations Research for the art of modeling mathematically the world around me and having a means of making a decision on what to do with whatever I have modeled. I developed an interest in computer science out of my endeavors to make operations research more valuable by means of developing time saving heuristics and embedding models into software that the average person can use. This work represents the joining of both of my interests which I have been developing throughout my education and serves as the capstone for my collegiate career.

Moving forward, I'd like to continue developing products, like this one, that aid people in making decisions or automate them for networks of machines. I see two bottlenecks in such products becoming reality, and they are the motivation for my work and study plans over the next few years.

The first is being able to develop scalable software, related to my third area of improvement highlighted above. In order for people or machines to be able to use such a product, intelligent data transmission and user interface, as well

as cloud and parallelized computing are bodies of knowledge over which I'll need a firm grasp. It is for that reason I took time between my undergrad and graduate studies to build data pipelines for a software company and intend to take time during my graduate studies to continue developing internet technologies.

The second is being able to develop scalable models, those that as they grow can still reach a good enough solution in a feasible amount of time. As noted in my second area for improvement, this is what stands in the way of me being able to build models, and therefore software, that can accomodate the future's toughest problems. For that reason, I view attending a graduate operations research program with a strong focus on algorithms as a necessity for my future. I intend to learn how optimization algorithms work, as well as equip myself with the abilty to come up with new methods for their approximations, so that no matter the complexity of my environment, I can solve the model that I made to describe it.

Thank you for taking the time to read my work.

Sean Kelley

217-722-2228

seanpkelley94@gmail.com

INDEX