Logistics Optimization for Regionally Located Equipment Documentation

Release

Sean Kelley

Jun 16, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

This documentation explains the math, algorithms, and computer code used to arrive at a solution for how to transport sets of construction equipment kept in various geographical regions for a national construction company. In the general sense, the code explained details the necessary investments of delivery vehicles and commodities to make feasible a single, large commodity delivery network, where large means that one unit of the commodity equals the capacity of our delivery vehicle. The single, large commodity delivery network will be known throughout the rest of the documentation as "the internal delivery system."

The current state of the construction company's delivery of equipment sets (their large commodity) involves the crews working at each job site to take time away from building at the start and end of their projects to move all of the sets of equipment they need. Further adding to costs, the crews have extra licensing and fuel inefficient vehichles they only fully utilize on the days they move equipment. The construction company believed there to be a more efficient means of operating.

In its inception, this problem concerned the construction company inquiring if an internal delivery system using its own drivers for moving sets of construction equipment would be more profitable than the crews moving the sets of equipment on their own for each geographical region in which they do business. Such a delivery system would consist of semi-trucks and flatbed trailers, as well as equipment haulers (truck drivers), and would be responsible for moving equipment sets to construction sites in the region before jobs begin and from them once they end.

With the costs of the crews moving the equipment on their own known to the construction company already, this work will focus on the calculation of costs for the new delivery system, namely:

- the number of semi-trucks and trailers
- the number of equipment haulers (truck drivers)
- the number of construction equipment sets

The output of the code described by this documentation is a report detailing the utilization of semi-trucks/haulers and equipment sets. This report can then be used to estimate the cost of the internal delivery system, leading us to make a comparison of the two systems and a confidently made decision.

1.1 Python Implementation

If you'd like to follow along in the code for which this documentation was created, I invite you to check out its repository at https://github.com/spkelle2/Equipment_Routing. Most following chapters have a "Documentation" section detailing where specifically in the code 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 determining if an internal delivery system is a more viable option to construction crews transporting their own equipment.

- *Main Function* covers the inputs that the code requires to begin calculations. If you're looking for a full understanding of how the math, algorithms, and code interact, start here.
- *Region Variation* details the calculations that only need done once for each variation of a geographic region we consider. If you're only interested in the math, read this part and the next.
- *Daily Routing* explains the data manipulation and calculations that require being solved one day at a time, for each variation of region we consider.
- *Reporting Our Results* outlines how all of the data and calculations from the previous two topics come together into one final report that can be used to make a decision on using the internal delivery system.
- In *Conclusion*, I'll give my closing remarks on the project.

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

Sean Kelley

Industrial Engineer, University of Illinois

CHAPTER

TWO

MAIN FUNCTION

2.1 Overview

This problem takes a historical approach to determining cost, using past data on construction equipment usage to determine if an internal delivery system would have been more cost effective than the construction crews moving their own equipment. Our data file (outlined below) will tell us the number of sets of construction equipment to be moved each day. This allows us to infer the number of semi-trucks and equipment haulers needed to move all equipment each day, as well how much equipment needed on hand to ensure each construction site has their demand for equipment met. Whichever system would have been more cost effective for the data we were provided will be chosen as the system predicted to be more cost effective in the future.

2.2 Inputs

Regardless of which variation of the problem is run, there are a few inputs to the problem that remain constant. They are as follows:

- the start and end date of the range of time to be considered
- the rate of travel of our semi-truck and trailer
- the length of a working day for an equipment hauler
- the amount of time on average it takes an equipment hauler to unload or load equipment from/on his trailer
- upper bound on how many semi-trucks can be used on one day
- the number of days (time window) in which a construction site is elgible for having equipment dropped-off or picked-up before/after its start/end date
- the geographic regions for which our problem will solve
- the different variations of each region (the differences being how many subregions of each region are to be considered)
- the location of the excel spreadsheet that has the data used by our problem. This file must include the following:
  - the geographical coordinates of each construction site worked on in our time range, separated by geographical region. The first and last entries must have indices 9998 and 9999 and be the location of where haulers will start and end their days as well as where equipment will be kept. The first image below shows an example of what this should look like.
  - the number of sets of construction equipment picked-up or dropped- off at each site each day, separated by variation of region. A negative number represents that many sets of equipment needing dropped-off and a positive one sets of equipment needing picked-up. The second image below shows an example.

	A	B	C
1	Project #	Lat	Long
2	9998	40.61	-89.46
3	1	41.00	-87.91
4	2	40.61	-88.18
5	3	39.80	-91.86
6	4	40.46	-90.67
7	5	40.46	-90.67
401	709	42.01	-89.33
402	710	42.41	-89.01
403	711	41.98	-89.22
404	712	42.42	-89.41
405	713	42.05	-89.43
406	714	42.49	-89.04
407	715	42.13	-89.26
408	716	42.10	-89.00
409	9999	40.61	-89.46

	A	E	F	G	H	I	J	K
1		1/5/2	1/6/2	1/7/2	1/8/2	1/9/2	1/12/2	1/13/2
39	86.00	0	0	0	0	0	0	0
40	87.00	0	0	0	0	0	0	0
41	88.00	0	0	0	0	0	0	0
42	89.00	0	0	0	0	0	0	0
43	90.00	0	0	0	0	0	0	0
44	91.00	0	0	0	0	0	0	0
45	92.00	0	0	0	0	0	0	-2
46	93.00	0	0	0	0	0	0	0
47	94.00	0	0	0	0	0	0	0
48	95.00	0	0	0	0	0	0	0
49	96.00	0	0	0	0	0	0	0
50	111.00	0	0	0	0	0	0	0
51	112.00	0	0	0	0	0	0	0
52	113.00	0	0	0	0	0	0	0
53	119.00	0	0	0	0	0	0	0
54	120.00	0	0	0	0	-3	0	0
55	121.00	0	0	0	0	0	0	0
56	122.00	0	0	0	0	0	0	0
57	123.00	0	0	0	0	0	0	0

Note: For a full example of what a proper data sheet cooresponding with the code looks like, check the repository at https://github.com/spkelle2/Equipment_Routing.

2.3 Region

For each different geographical region we test, we will find the cost of the internal delivery system for each different variation of that region. What is meant by variation is the same geographical region, just contracted to include fewer jobs or expanded to include more. The purpose for doing this is to help us find variations where assets have high usage rates, which correspond to a higher chance of being a success.

The best return on investment of all variations for each region will be the variations used for comparing the costs of the internal delivery system to the crews delivering their own equipment. Solving for any variation of any region is formulated in the same way. A reporting tool will print the results (number of semi-trucks, equipment haulers, and extra equipment sets) for each variation of region, and the user may then calculate the respective costs to determine which are optimal. However, since all of these results will differ by variation of region, we'll move next to talking about what parts of the code and calculation take place individually for each variation.

Continue to *Region Variation*

CHAPTER

THREE

REGION VARIATION

3.1 Importance of Time Windows

When calculating the cost of the internal delivery system for each variation of region, it is important to find an optimal trade-off between the number of semi-trucks and equipment haulers and the number of sets of construction equipment needed. For example, prescribing that each construction site must have its equipment dropped-off the day before work begins and picked-up the day after it ends will result in a couple extremes. It will minimize the number of sets of equipment needed to meet all construction sites' demand for equipment (as it minimizes the amount of time equipment sets sit unused waiting for pick-up) but will maximize the number of semi-trucks and equipment haulers needed as some days may require many more equipment movements than others. For that reason, the formulation requires a time window (number of days before a construction site begins or after it ends) in which a site must have its equipment sets dropped-off or picked-up. Adding just a day to the time window (increasing it from 1 day to 2) will dramatically decrease the number of semi-trucks and equipment haulers without having much effect on the number of sets of equipment.

3.2 Set-Up for Daily Computations

Assigning the time window as optimally as possible, the first step for each variation of region is to smooth the demand for the number of drop-offs and pick-ups to be made each day as evenly as possible subject to all drop-offs and pick-ups occuring within the time window before a work at a site begins or after work at a site ends. (The section "Smoothing Demand" below will cover this process more in depth.)

Note: To assign the time window as optimally as possible, it is suggested to try a small range of values in order to see which returns a least total cost in the report output at the end.

We then create two tables, the first recording how many miles all semi-trucks run each day and the second recording how many hours each equipment hauler works each day. We then return to the data sheet with the number of equipment sets to be picked-up or dropped-off at each site each day for this variation of region (which has now been smoothed). For each day in that data sheet between our start and end date, we make one list tracking which sites that day need a drop-off or a pick-up and a second list tracking what the demand for drop-offs or pick-ups at those sites are.

Note: One drop-off or pick-up (1) is equivalent to one set of equipment (2) as that is the maximum capacity for a single truckload (3). These three terms will be used interchangeably moving forward.

Knowing which sites need drop-offs and pick-ups, as well as how many each of those need, we can now formulate how many semi-trucks and equipment haulers we need for each day. Before diving into that, though, let's take a second to understand what's going on in the demand smoothing part I mentioned above.

3.3 Demand Smoothing Model

As mentioned previously, the purpose of smoothing our demand for equipment drop-offs and pick-ups is to greatly reduce the amount of semi-trucks and equipment haulers needed to move all of the equipment while slightly increasing the amount of equipment needed on hand.

We approached this part of the problem by formulating an integer program. It is defined by a set of job sites (indexed by i), a set of days (indexed by l), and a set S of sets of alternative days that drop-offs/pick-ups can be made for a site i with demand on day l. Each site i has a demand, $d_{i,l}$, stating how many drop-offs or pick-ups it needs on day l, with $d_{i,l} < 0$ for drop-offs and $d_{i,l} > 0$ for pick-ups. The sets within S, known as $S_{i,l}$ (indexed by l', as we are indexing a subset of our set of days), are 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 time range, the set is just the remaining days.)

We used two variables in this problem. The first, $w_{i,l}$, represents the number of truckloads (equivalently, sets of equipment) going to or from site i on day l. The second, z, is the largest sum of truckloads 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 truckloads going to or from that site within the time window must equal the number of truckloads originally demanded to or from 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 needed done, 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.

3.4 Demand Smoothing Improvement Algorithm

One issue I saw with the formulation we came up with is that if a couple of days were to have a much higher demand for drop-offs and pick-ups than all of the others, (3) would not be a tightly enforced constraint on many of the other days, and their demands would not be smoothed effectively. To work around this, we took a 'divide and conquer' approach to our final formulation for demand smoothing. Rather than trying to smooth all of our input days of data with the integer program in one go, we broke the data set into equal sized chunks (periods) and had the integer program solve each period individually.

To further optimize results, we repeated this process for periods with lengths varying from a couple days to a couple weeks.

Note: We picked the periods to be at least a couple of days so at minimum some of the day's demands in that period could be reassigned. We capped the periods' lengths at a couple weeks, however, because such large demands would appear for one day that our integer program would no longer smooth very effectively.

After running the integer program over all periods in our data for each different period length, we picked the smoothed data we were going to work with by whichever period length resulted in the least amount of variability over all of its days (thus was the most smooth). Now that we have chosen the smoothest data we can for drop-offs and pick-ups to be made each day, we can formulate how we're going to figure out how many assets (semi-trucks, equipment handlers, and equipment sets) we're going to need to meet each construction site's demand.

With the number of drop-offs and pick-ups to be made to each site each day now smoothed and a way to record our results created, we can move on to calculating the routes that the fleet of haulers must make each day.

Continue to *Daily Routing*

3.5 Documentation

Set-up for daily computation:

iterate.**solve_variation**(*fixed_parameters*, *region*, *variation*)

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)
- **region** (*str*) – The geographical region for which we're finding usage stats
- **variation** (*str*) – The variation of the geographical region for which we solve

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

Improvement algorithm/wrapper for demand smoothing functions:

smoothing.**smooth_demand**(*file_path*, *variation*, *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

- **file_path** (*str*) – location of the excel spreadsheet that has our demand and site location data
- **variation** (*str*) – which variation of a proposed region for which we want to smooth demand
- **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

CHAPTER

FOUR

DAILY ROUTING

Knowing how many drop-offs and pick-ups needed made each day at this point, we tried to route all equipment hauler movements throughout the time range in one integer program. This proved quite unsuccessful, however, as our integer program we modeled after that was so large it could not run in a feasible amount of time. After some trial and error, we discovered that equipment hauler routing (where each equipment hauler goes at each point in time) could be done very efficiently when done one day at a time, thus everything in this section is repeated for each day in our timeframe.

Note: Due to the complexity of trying to route our equipment haulers, it was necessary to smooth the demand beforehand. Smoothing could also take place by routing for multiple days at a time, assigning time windows to deliveries in the same manner as in the smoothing process; however, doing this even for 2 or 3 days resulted in computation times above feasibility.

4.1 Creating Parameters

Before solving for the routes each equipment hauler will take each day, we need to create a few parameters our routing integer program will need to be able to solve. They are the following:

- **demand,** d_i The number of drop-offs and pick-ups each site needs on a given day, ignoring the sites with no demand.
- **locations (indexed by** i**)** A list of all sites with demand for drop-offs and pick-ups or, equivalently, the sites, i, for which d_i exists on a given day. This list also includes the hubs, where the equipment haulers start and end their day. They're represented as different numbers but can be same physical location. This list has length $n + 1$. The hubs are locations$_0$ and locations$_{n+1}$.
- **customers (indexed by** i**)** A list of only just the sites with demand for drop-offs and pick-ups on a given day. This list has length $n - 1$.
- **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 hauler must alternate between dropping-off equipment and picking- up equipment.
  - A hauler 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 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 a site, i, is from a site, j, indexed in the same order as the locations list. The sites represented in this matrix are only the hubs and those which have a demand for drop-offs or pick-ups on this day, and the distances are calculated by converting the differences in geographical coordinates listed for each site.

- **subsets, S_m (indexed by i', a subset of a sites)** A list of the even-sized subsets, m, of sites with demand on a given day, necessary for forcing continuity in our equipment haulers' routes.
- **M** An arbitrarily large number.
- **haulers (indexed by k)** A list of the equipment haulers we have available.

Also recall a few parameters we stated as inputs for each region we'll solve.

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

4.2 Developing the Routing Model

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

Now that we have declared all of our parameters we will need to solve our equipment hauler 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.:

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

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

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

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

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

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

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

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

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

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

(1) tells us our objective is to minimize the total amount of distance that our fleet of equipment haulers cover each day. $(2) - (4)$ add constraints for modeling the travel from one site to the next. (2) says that each hauler must leave the hub each day, even if just to return to it at no cost (equivalent to not being used). If a hauler arrives at a site, h, to make a drop-off or a pick-up, (3) ensures that hauler also leaves that site for another. Once a hauler has made

[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.

all drop-offs and pick-ups it needed to make for a day, (4) requires that he returns to the hub. (5) brings our time constraints into play. We required that all driving and unloading/loading (un/loading) of semi-trucks must be less than the length of the allowable work day. Time for an extra handle was added under the assumption that the semi-truck needed no adjustments at neither the start nor end of its day. (6) covers our demand constraint, the requirement that each site, i, must be visited exactly as many times as it needs equipment 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 hauler's visit to a site be one where his semi-truck is able to satisfy one unit of that site's demand. $(8) - (9)$ are additions to $(2) - (4)$, enforcing that equipment haulers take routes that are only possible in the real world. (8) monitors whether or not an equipment hauler enters a given subset of sites, while (9) requires that an equipment hauler leave that set if he entered it. These constraints remove the possibility that the routes a hauler takes in a day are disconnected. Lastly, $(10) - (11)$ define the spaces for our variables, specifically that equipment haulers can take no partial routes and either enter or do not enter any given set of sites.

4.3 Running the Routing Model

One final piece of information remains to fully define our integer program for hauler routing, how many haulers (and therefore trucks) are available to be used on a given day. Since the purpose of the problem is to use as few assets as possible given a list of drop-offs and pick-ups to make each day, we start by running the model with as few haulers as possible, one, and rerun it, adding one hauler per rerun, until our model is feasible. Once we achieved feasibility, we recorded in the tables created before running the first day's models the total number of miles the haulers drove this given day and how many hours each of them consequently worked.

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 costs associated with this variation's of region delivery system.

Continue on to *Reporting Our Results*

4.4 Documentation

Daily routing wrapper function:

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

Determine the usage of semi-trucks and 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, and how long each hauler works.

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
- **`region`** (*str*) – The geographical region for which we're finding usage stats

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

Creating parameters wrapper function:

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

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
- **region** (*str*) – the geographical region for the variation we are modeling

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 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*, *file_path*, *region*, *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

- **file_path** (*str*) – location of the excel spreadsheet that has our demand and site location data
- **region** (*str*) – The geographical region for which we're finding usage stats
- **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

Summary of fleet mileage accumulated 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 equipment haulers spent working on a given day:

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

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
- **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

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

Return type numpy.ndarray

CHAPTER

FIVE

REPORTING OUR RESULTS

Upon completion of the hauler 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)
- how many miles the fleet of haulers traveled each day
- how many hours each hauler worked each day

All of this information we then compile into a report which can then be used to make a calculation of the cost of this variation's delivery system.

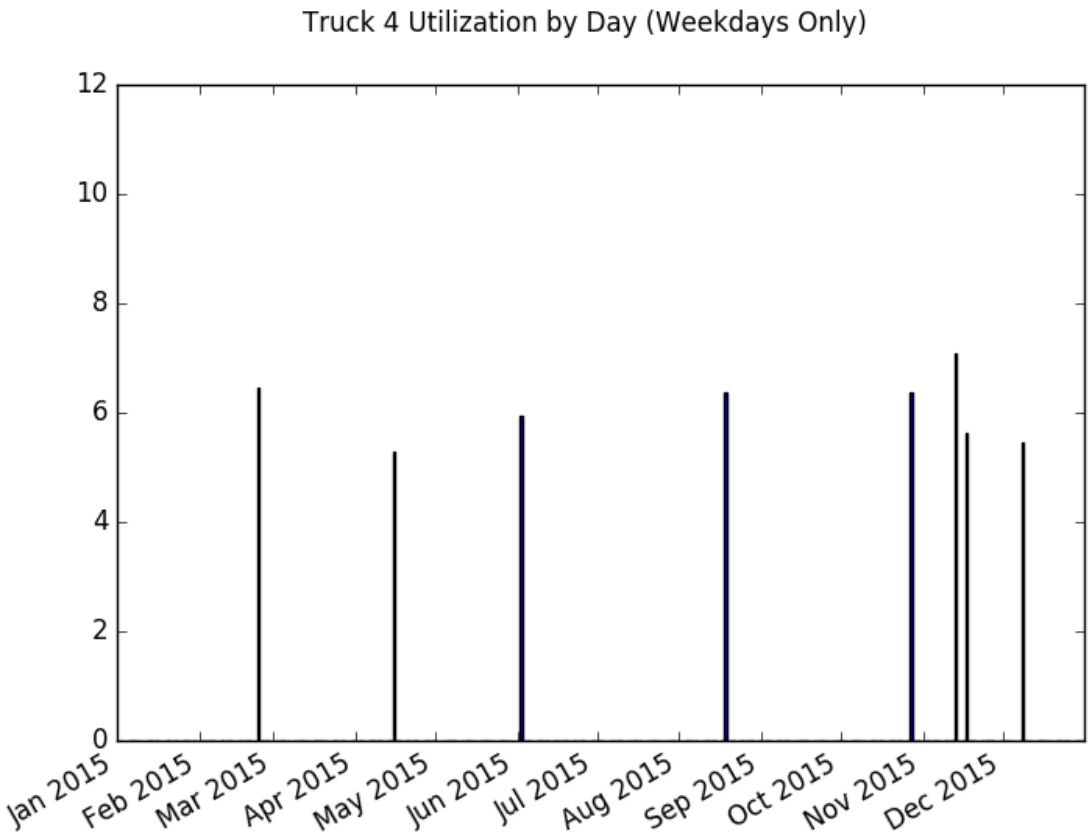
5.1 Equipment Hauler Cost

The first step in the reporting process is to summarize how often each hauler is utilized. Each day when we recorded the hours each hauler worked, we assigned hours to haulers from greatest to least, so the first hauler got the most hours, the second got the second most, etc. By doing this, when we average how many hours each hauler works each day and how many days in a time period each hauler worked, we ensure the fewest amount of haulers do the greatest amount of work. Based on how often each hauler is utilized, a decision can be made as to whether or not that hauler needs hired and his semi needs purchased, or if overtime by other haulers can be worked to cover the extra drop-offs and pick-ups. Later in the reporting process, this decision can be reinforced by graphs mapping out how much each hauler works on each day, telling us whether or not such an overtime strategy would be effective in replacing a given equipment hauler. With this information we can calculate the cost to staff the required number of equipment haulers, the first required piece of information in estimating cost.

For example, the image below illustrates what the results of this summary might look like. It's pretty easy to tell that the last three haulers probably do not need hired. The second image typifies the graph that maps how much each hauler works each day, and supports the last three not being hired as 4's hours are pretty spread out, could be covered by another hauler working extra, and are greater than 5's and 6's.

Total Miles Driven by All Trucks: 94795.0

	Hours Worked in One Year	Days Utilized	Percentage of Working Days Utilized	Average Hours Worked per Utilized Day
1	1576.0	220	84.3	7.2
2	875.0	120	46.0	7.3
3	360.0	47	18.0	7.7
4	49.0	8	3.1	6.1
5	6.0	1	0.4	6.3
6	6.0	1	0.4	6.3



5.2 Semi-Truck Cost

Next we can induce how much should be spent on equipping our haulers with semi- trucks. We already know how many trucks will be needed (as we would like one for each hauler), but another part of the cost comes from how often we have to replace the semi-trucks based on how many miles they run per period of time. This part of the calculation begins by summing the total number of miles the fleet runs each day for our time period. An example of this number appears at the top of the first image. Then taking the ratio of fleet expected lifetime mileage and time period mileage yields how many time periods the fleet can run before replacement. Knowing the size of our fleet and how often it needs replaced, we can infer costs for our semi-trucks, our second crucial piece of information.

5.3 Equipment Sets Cost

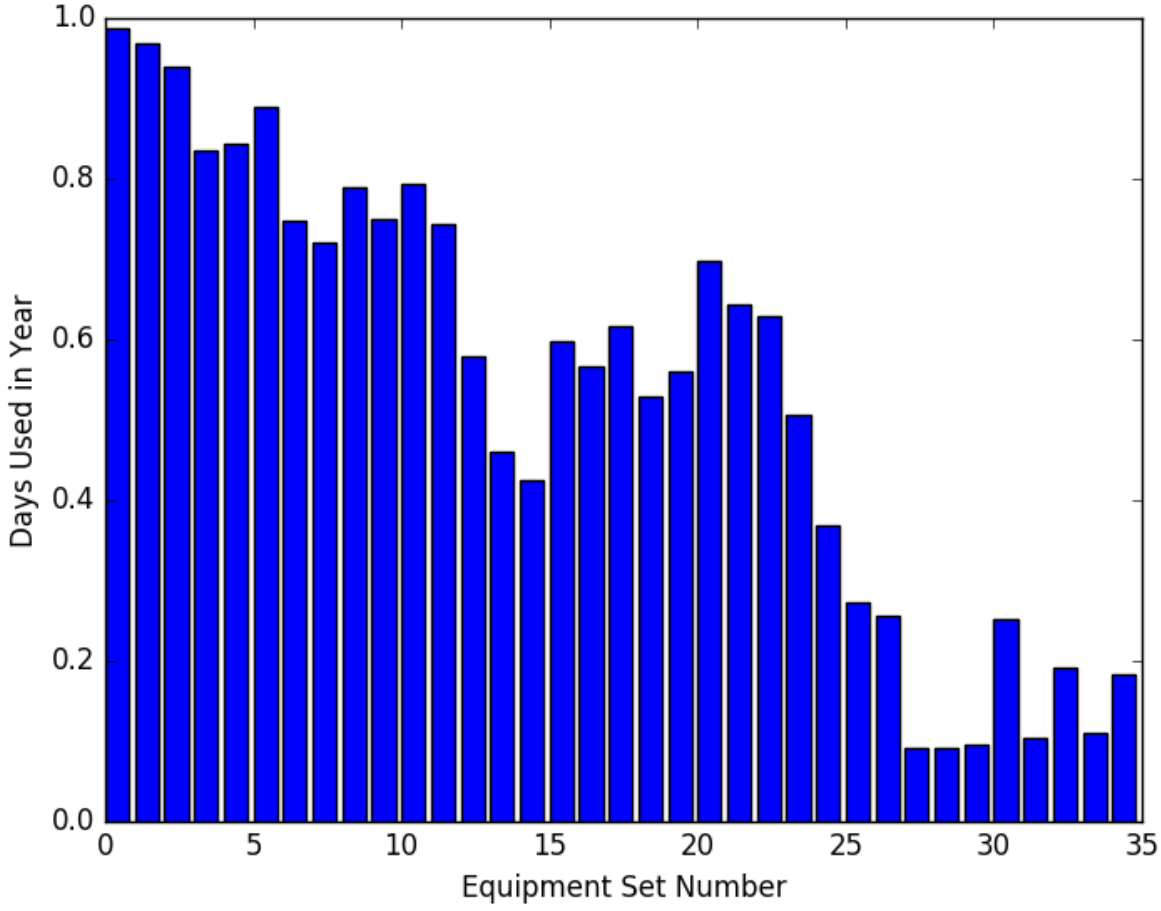
One final crucial piece of information has yet to be determined for making our cost calculation, and that is how many sets of equipment will be needed for this system to work. For each day in our time range, we repeated the following process:

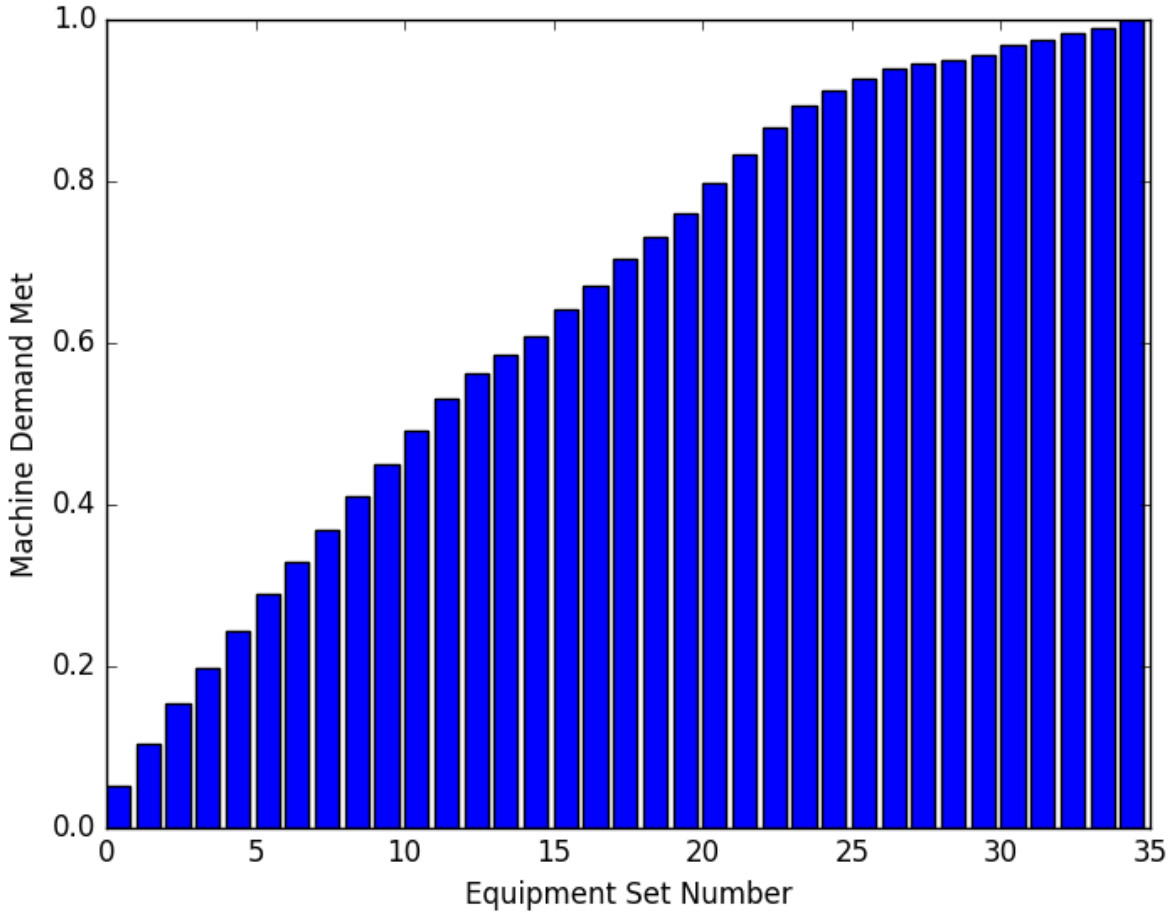
1. If a set of equipment is at a site, mark it as in use. If it gets picked up from that site later in the day, it is still being used as it either goes to another site or spends the day returning to the hub.
2. If a site has a demand for equipment to be picked-up, assign that number of that site's sets of equipment to be free for usage elsewhere.

3. If a site has a demand for equipment to be dropped-off, assign that number of sets of equipment to that site. This can include the sets of equipment that were just released, as haulers can meet one site's demand for drop-offs by picking-up from a site that released equipment on that day.
4. If a set of equipment is at a site, mark it as in use. We repeat this step to ensure that additional sets of equipment that were brought out to job sites, if necessary, are counted as in use.

Also worth noting in the process, the sets of equipment are indexed in increasing order, and assignment is made to the set of available equipment with the lowest index, ensuring we use the smallest number of equipment sets for the majority of the work.

From here we make two graphs to illustrate the machine set usage. The first graph depicts how often each set of equipment gets used, helping us decide whether or not a given set of equipment should be bought or rented. The second graph shows us how much additional utility each set of equipment adds, further detailing how many equipment sets are needed to meet the demand. With this information, we can now infer how many sets of construction equipment are needed as well as how much each set will cost (renting/buying is determined by a utilization threshold, anything below is rented and above is bought). An example of the first graphic is shown below, followed by an example of the second. With an idea in hand of what threshold of utilization is needed to buy a set of equipment vs. rent it, we can calculate a total cost for the sets of equipment required by the delivery system.





5.4 Decision

At this point, all of the above information is compiled into a single report corresponding to the given variation of a region, and can be used to calculate the cost of this variation's delivery system subject to the time window used. For an example of what a complete report looks like, I invite you to download `topeka_base_report.pdf` (available for download at https://github.com/spkelle2/Equipment_Routing if you're reading from a pdf). Finding the minimum cost of all variations and time windows tested for a given geographical region reveal to us the estimated cost of investment for the internal delivery system. In the case of the construction company, comparing these minimum costs to the current cost of operation in each of their geographical regions was used to determine if the internal delivery system was desirable.

In summary, the results of all the code and models are accurate estimates of the investments an organization would need to support a single, large commodity delivery network. These investments can be clearly interpreted by means of the graphs compiled in the output report. With a full means now of understanding how a decision can be made, I invite you to look at my closing remarks on the project.

Continue to *Conclusion*

5.5 Documentation

Report generator:

reporting.**make_report**(*data*, *variation*, *fixed_parameters*)
Creates a PDF 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
- **variation** (*str*) – The variation of the geographical region for which we solve
- **fixed_parameters** (*dict*) – Parameters that are constant for any variation and region (as defined in the main function)

Summarization of hauler 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

Bar graph for daily hours of a hauler:

reporting.**hauler_graph_maker**(*hours_df*, *index*, *plotlist*, *directory_name*, *variation*)
Plots a bar graph detailing amount of hours an equipment hauler worked each day

Parameters

- **hours_df** (*DataFrame*) – How many minutes each hauler works each day
- **index** (*int*) – Which hauler we are currently plotting
- **plotlist** (*list*) – List of strings containing file names for all previously created hauler plots
- **directory_name** (*str*) – File path of our working directory
- **variation** (*str*) – The variation of the geographical region for which we solve

Returns **plotlist** – List of filepaths for previously made graphs in this variation

Return type list

Equipment set utilization:

reporting.**equipment_usage_analysis**(*demand_df*, *directory_name*, *variation*)
Determines for each day whether or not a given set of equipment was used

Parameters

- **demand_df** (*DataFrame*) – How many sets of equipment each site needs dropped-off or picked up each day
- **directory_name** (*str*) – The path for the directory where we'll save our graphs
- **variation** (*str*) – The variation of the geographical region for which we solve

Returns **usage** – A matrix of binaries representing whether or not a given set of equipment was utilized on a given day

Return type numpy.ndarray

Equipment set utilization graphs:

reporting.**equipment_graph_maker**(*demand_df*, *variation*, *directory_name*, *plotlist*)

Makes bar graphs for proportion of time range equipment is utilized and proportion of demand that can be met with a given number of equipment sets

Parameters

- **demand_df** (*DataFrame*) – How many sets of equipment each site needs dropped-off or picked up each day
- **variation** (*str*) – The variation of the geographical region for which we solve
- **directory_name** (*str*) – The path for the directory where we'll save our graphs
- **plotlist** (*list*) – List of filepaths for previously made graphs in this variation

Returns **plotlist** – List of filepaths for previously made graphs in this variation

Return type list

CHAPTER

SIX

CONCLUSION

6.1 Possible Improvements

Looking back on the work I did, there are a couple avenues I wish I would have had the time to explore that I think would have improved our result.

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 quadratic function, I would have liked to have had the time to research approaches to smoothing demand with an Integer 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 (never more than 10 locations), dealing with exponential constraint growth was not an issue. However, finding a way to model the problem in the same manner but without the exponential constraints would be necessary for scaling the number of locations for which we could solve at one time.

Lastly, I think the reusability of the project could have benefitted from having a more automated way of generating data for the model to use. Due to time constraints, the best solution we could come up with was pulling data from a database and moving it by hand into separate spreadsheets, but I think that could have been accomplished with much less difficulty on the end user if an end-point for a database and a query were input instead. That way, the data on locations and demand for drop-offs and pick-ups for each variation could have just been mined automatically, saving time and stress on the user.

6.2 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. I could not be more thankful for the people who have helped me get to this point, especially my father who instilled in me the work ethic required to gain the experience I needed to complete this project.

Moving forward, I'd like to continue developing products, like this one, that will allow any person, regardless of background in math or computing, to leverage optimization methods for providing decision support. For any questions or inquiries, please feel free to reach me at the contact information below.

Thank you for taking the time to read my work.

Sean Kelley

217-722-2228

seanpkelley94@gmail.com

INDEX