Luke Wilson
Scientific Computation
Dr. Eijkhout

Amazon Delivery Trucks

0 - INTRODUCTION

A classic problem in computer science and mathematics is the *Traveling Salesman Problem* (TSP). This problem was formulated in 1930 by an Australian mathematician named Karl Menger and goes like this: a traveling salesperson must visit n number of cities in a day. He knows the distance between any two cities. How does the salesman go about minimizing his distanced traveled throughout the day?

Over 90 years later, the importance of this problem has grown significantly with the advent of online retailers delivering their products to customers homes. The most successful of these being Amazon. In this paper, I will be investigating the Traveling Salesman Problem with a business like Amazon in mind.

There will be a number of things to consider when tackling this problem. Not only will we have to consider the TSP, but we must consider it in light of Amazon's business. The immediate importance of TSP is self-evident to Amazon. Longer delivery routes require more labor, and like any good business, Amazon wants to be as efficient as possible. A good solution to TSP allows amazon to cut costs and deliver products at a faster rate. Amazon receives thousands of orders every day, and each of their orders has a unique guaranteed delivery date attached. Given the high volume of deliveries Amazon has promised and their varying delivery dates, Amazon must find an Algorithm to deliver all these products on time in an efficient manner. Our algorithm will thus consider the delivery dates of each order. Amazon also has many vehicles operating at once. Thus, finding the optimal route for each vehicle is key. In this paper, I will seek simulate a scenario for an Amazon Delivery Trucks operating in the real world.

0.1 - Approach

We will tackle this problem as follows:

1. Coding the basics (address class and methods, route class and methods, depots)
2. Optimizing the route (greedy method, opt2 method)
3. Working with multiple routes (opt2 method extended)
4. Daily Deliveries (prime now, driving speed, work hours)

Source code files will be contained in project1.cxx, which contains class definitions and methods, and project1_functions.cxx, which will store all additional functions needed to run simulations and optimize routes. There will also be a project1_main.cxx file, which was used to run simulations and obtain insights and results.

1 - CODING THE BASICS

1.1 - Address Class

The first thing we do is create an address class that represents a single house or delivery location. This address will exist in Euclidian space (on a 2d plane). The class will have three members:

i. *i*, representing the x coordinate of the address
ii. *j*, representing the y coordinate of the address
iii. *delivery_date*

The members *i* and *j* will be float datatypes in my code. We will also include a *delivery_date* member which stores the day by which the package must be delivered to the address. I used an integer datatype to simplify calculations. A more complex timestamp can be used if we decide to do more than simulations.

1.1.1 - distance method

The only method I will include with this class in the *distance* method which will calculate the Euclidian distance between its host address and another address. The method will receive an address and return a float.

1.2 – Route Class

The next class I implemented is the *route* class. This class is a route for a delivery person to follow. The textbook mentioned an *address list* class, but given the similarity between the two, it was easy for me to use solely the *route* class. The key member to this class is its *route list* which contains a vector of *addresses.* This class holds a number of methods which will be outlined below. We assume a depot address at (0,0) where the truck will begin and end each day. Methods are designed to keep these addresses at the beginning and end of the list.

1.2.1 – Length method

The length method will return the length of the route. This is done by creating a *len* variable of float datatype. The method loops through each address, calculating the distance between such address and the next address. These distances are cumulatively added to len to obtain a total length of the route. Note: looping ends one before the last address since the last address has no following address.

1.2.2 – index_closest_to method

The *index_closest_to* method return the index of the address closest to the inputted address. The method loops through each address, calculating distance and, if distance is smaller than any previously calculated, storing that addresses index in a *closest_index* variable. After looping though each address the method return the closest_index variable.

1.2.3 – Add_address method

The *add_address* method receives a new address, loops through existing addresses to see if address exists, then appends the address to the route list if it is unique. Note: since there is no defined method of equality for addresses, we say two addresses are the same if the distance between the two is zero.

1.2.4 – display_route method

The *display_route* method is used to output the coordinates of the addresses its host route. This is done through looping.

These are the methods and members we will begin with to create our simulation.

2 – OPTOMIZING THE ROUTE

2.0 – gen_rand_route function

I am about to go through a couple route optimizing methods. As we do this, we want to have a way to test how much better they are then one another. For this I have made a random route generator. The function gen_rand_route receives a route size and a width (how far horizontally and vertically addresses can be from the origin). The function returns a route containing random addresses in a width wide square about the origin. The function takes these steps:

i. receives route size and width size
ii. initializes route size and loops though route
iii. defines a random address in route using rand()
iv. return random route

This will allow us to easily test our route optimizing methods.

2.0.1 – randomize_delivery_days function

This is another function which will be used later in simulations to assign delivery days. The function,

i. receives a *route* by reference, a *days* variable, a *today* variable
ii. loops through all addresses in the route
iii. assigns each address a random delivery date between *today* and *today + days*

2.1 – greedy route

The first method of optimization will be through a greedy route. The greedy route begins at the depot and always travels to the nearest address until all addresses are exhausted and the truck returns to the depot. This is an easy and fast way of calculating a generally more efficient route. Note: this is done in my code by creating a parallel address list and appending and erasing as the method loops thought the route, but it can also be done using a Boolean vector whose indexes match the route. This vector would tell whether an address has been added to the new greedy route or not.

2.1.1 – Greedy Route Simulation

We will first test our greedy route against a random route. This simulation is carried out in the *greedy_sim* function in *project1_main.cxx*. For this we will generate 100,000 random routes with varying amounts of addresses each on a certain unit wide square about the origin. We will test on multiple sizes of routes and squares observing the percentage change in length of route. The results are as follows.

Route Size:	**10**	**20**	**50**	**100**
Width = 20	-4.3%	-4.5%	-3.3%	-1.4%
Width = 50	-4.9%	-5.9%	-6.0%	-5.6%

Width = 100	-5.1%	-6.0%	-6.6%	-6.6%
Width = 200	-5.2%	-6.2%	-6.8%	-6.9%

We can see that the greedy route typically improves the random route by around 4-7%. We see a couple outliers at low sizes, but this is expected since the average distance between two random points is much smaller.

2.2 – opt2 route

The next method of optimization will be through a "opt2" method. This method relies on the idea that if you have a path that crosses itself, you can make it shorter by reversing it. Using this information, we

i. Loop though every possible segment in the route
ii. Test if reversed segment is shorter than original segment
iii. Reverse segment if it is short, do nothing if not

This process works very well and generally returns a much more efficient route. Note: since this method goes though every possible segment of the route, it takes much longer than the greedy route.

2.2.1 – opt2 route simulation

With a solid understanding of how the greedy route improves a random route, we will now compare it with the opt2 optimization method. We will preform the same process as in 2.1.1 except now comparing the percent change in efficiency of opt2 versus greedy. Also, we will only generate n = 10,000 random routes because 100,000 takes too long using opt2() method. The results are as follows.

Route size: Width:	**10**	**20**	**50**	**100**
20	-31.62%	-48.68%	-65.92%	-74.80%
50	-31.61%	-48.80%	-66.24%	-75.63%
100	-31.64%	-48.78%	-66.26%	-75.70%
200	-31.64%	-48.73%	-66.27%	-75.63%

Here we see some big numbers compared to last table. The opt2 method proves to be much more effective than the greedy route. Note that the efficient of opt2 versus greedy does not very much with width size. Do not confuse this with opt2 not improving individually with size. It likely does, but only at the same rate as greedy, which is why we don't see much change on this table.

3 WORKING WITH MULTIPLE ROUTES

Amazon does not have one truck, so neither will we. In this section we will seek to understand and optimize delivery routes for more than one truck. So far, we have only used class methods to optimize routes, but now since we are using multiple routes, I opted to use functions

defined in *project1_functions.cxx*. Here parameters are passed by reference, so we do not have to worry about returning vectors of variables.

3.1 – Extending the opt2 method

The "opt2" method of optimization can be extended to two routes. The intuition here is that if segments of two routes cross one another, they can be uncrossed and shortened by swapping their addresses. The process for this is slightly more complicated and computationally rigorous. To simplify things, we will only consider segments of length 4. Considering segments of varying length makes memory allocations much more difficult. To do such a thing, we would probably have to move away from vector datatypes. Nevertheless, in the algorithm, we first

i. Loop though every possible combination of route segments
ii. we swap segments and test to see if we became more efficient. Note: segments may be flipped when swapping between routes. Therefore, we must flip each route, testing each time to see if efficiency has improved. This means we have 5 possible options:
    a. No change
    b. Swapped, no flipping
    c. Swap, flip first route
    d. Swap, flip second route
    e. Swap, flip both routes

As you can see this easily becomes computationally rigorous and increasing computation speed very quickly. Testing and swapping is very present in this loop, so I made another function *swap_if_shorter_routes*. This function receives a pair of routes, passed by reference, and a potential pair of potentially shorter routes. If you later routes are more efficient, the former routes are redefined as their more efficient counterparts. While the computation is inefficient, the routes this method produces are not.

3.1.1 opt2 extension simulation

As you can expect, we will perform a similar simulation for our optimization of two routes. This comes from the *opt2_two_routes_sim* function in *project1_main.cxx*. We will be comparing the combined lengths of two random routes with the two optimized routes generated by out algorithm. Since this method is more complex, we will only use n = 1000 samples of random routes for each unique pairing below. Here are the results.

Route size: Width:	**10**	**20**	**50**	**100**
20	-20.31%	-16.15%	-9.94%	-6.27%
50	-19.93%	-16.30%	-9.56%	-6.09%
100	-20.11%	-16.05%	-9.58%	-6.07%
200	-20.11%	-16.70%	-9.72%	-5.99%

We can see here that the opt2 methods improves the two routes efficiently. However, as the number of addresses increases our method becomes less effective. This is in contract to the single route greedy and opt2 methods which become more efficient as route sizes increase. Why this is? I am not sure. Perhaps there is a computational error on my part. This was by far the most

difficult optimization method to code. The reasons for opt2's poor performance two routes of increasing size is something further research could be done on.

4 – DAILY DELIVERIES

We now want to model an Amazon trucks delivery day more accurately. Here we will begin to utilize the delivery day member in our address class. We will also consider the time or speed at which deliveries can be made within a workday. According to a forum on Amazon's website, the typical amazon driver delivers to around 200-250 addresses a day while working for 10 hours. "We can't afford to spend more than 30 sec to a min per stop."[1] According to another forum, Amazon typically has the driving within 5-20 mile radius. [2] We will use this information in out simulations.

4.1 – Driving Speed and Work Hours

Amazon trucks have limitations. Truckdrivers cannot drive 24/7 and they can only drive so fast. They are limited in how far the truck can travel in one workday. For our simulations we will have an average *speed* (mph) and a *workday_hours* variable. An Austin American-Statesman article claims, "the average speeds of commuters. Austin drivers drove an average of 24 mph during peak traffic times and just over 46 mph during less-congested hours."[3] We can assume the daily average is somewhere around 35 mph. We must also consider that in out simulations, drivers are traveling in straight lines between addresses. This obviously does not happen in real life. We have streets and stop lights, and some streets are faster than others. All of this could be considered for future simulations. In our simulations, a truck route cannot surpass *speed * workday_hours* miles. Note: in the real world, truck speed cannot be adjusted very easily, but workday hours can. This is where we run into ethical issues.

4.2 – New Deliveries Simulation

We will now look to conduct a simulation for a given Amazon driver in the area. We will begin with about 800 packages, each with a delivery date, to be delivered over a 5-day period. We will assume a 15-mile square city where the driver travels at 30 mph; (this number is slightly lowered since he is driving in a straight line in our simulation). Each day between 200 and 300 new orders will be placed with varying delivery dates (over the next 5 days). Each day, the driver will decide an optimal route to deliver all the packages that need to be delivered that day. If that driver has to spend more than 10 hours delivering packages, we say he had to work overtime. This simulation is described in *new_deliveries_simulation* in *project1_main.cxx* and the *simulate_days* function defined in *project1_functions.cxx*. In this simulation we will recreate the week described above for n = 1000 iterations, each time keeping track of when the driver is required to work overtime. After 1000 iterations, I found that the driver works overtime 60% of days. This number is large, but in honestly, we shouldn't think much of it. These simulations used a lot of parameters that we had to use reasonable guesses on. How accurately this describes real life? I am not sure. Finding better parameters on amazon driver speed, delivery drop off time, average distance from depot and new order placement frequency can help us generate a more accurate simulation. One thing I would like to have explored is modeling two delivery trucks in the situation.

5 - CONCLUSIONS

5.1 - Results

We learned a lot about different optimization methods and how might amazon go about selecting delivery routes. The greedy route proved to be a quick and effective method of reducing route length. The opt2 method was much more effective at returning efficient routes, but took a significantly larger time to compute. The opt2 method extended to two routes is effective yet time consuming. The method seemed to be less effective as route size increased, a property neither previous method shared. We finished by modeling a week in an Amazon delivery person's life. We found that, given the parameters we chose, one is expected to work overtime 60% of days. As we discussed, we are not incredibly confident in this number.

5.2 - Ethics

Amazon is a company that has been heavily criticized for its high employee standards, especially for its treatment of delivery workers. From this project, we have discovered how difficult it can be to efficiently deliver packages. Amazon makes big promises to its prime customers, and the drivers are often the ones who must work overtime to fulfil those promises. Finding efficient solutions to the problems described in this project are especially important to those drivers.

5.3 - Further Research

Throughout the paper, I have mentioned opportunities for further investigation. I will go over those here.

5.3.1 – k-means clustering

While not mentioned in the project description, I am aware that k-means clustering is a popular clustering algorithm and one that could easily be applied to the traveling sales-person problem. K-means clustering methods can be used in scenarios involving more than two trucks. We could also use k-means techniques to help determine how many trucks should be out driving at once. K-means also provides a nice geometric intuition for how the algorithm works. In the future k-means would be an interesting thing to consider.

5.3.2 – Using vector alternatives for opt2 method on multiple routes

The opt2 method used in this project for two routes considered only segments of fixed sizes. This decision was made due to the memory allocations difficulties faced when swapping variable sized segments between vectors. To avoid this problem, one could consider using lists or arrays. Using pointers is also a potential solution as lists of pointers have no fixed length. This would allow for more efficient swapping of route segments.

5.3.3 – opt2 method's decreasing efficiency with size

When optimizing one route, the greedy and opt2 methods became more efficient as route size increased. This was not the case however for opt2 with two routes. There is no obvious intuition as to why that is.

5.3.4 – Amazon Delivery Driver Parameters

Finding accurate parameters to model a typical amazon delivery drivers' day would allow up to better study and analyze Amazon's labor policy. A better model of the typical delivery day would also allow us to hopefully find more efficient solutions to Amazon's problems.

5.3.5 – Route Visualization

The TSP is unique in that it is inherently geometric. Finding a good way to visualize the algorithms would allow us to better understand what they are doing (and where they are going wrong).

5.4 – Reflection

Amazon faces a difficult yet fascinating problem. Every day they must figure out how they are going to deliver thousands of packages across the United States. It is not easy, especially considering the rate at which orders are placed. The problem is computationally fascinating and ethically important considering the hours which Amazon's employees are asked to work. This paper only touched the basics of the problem and I am excited to play around with TSP and similar problems in the future.

5.5 - Citations and references

[1] https://www.amazon.com/ask/questions/Tx3DJUCU5ERKKLU/#:~:text=We%20deliver%20250%2D300%20packages,hour%20depending%20on%20the%20weather.

[2] https://www.indeed.com/cmp/Amazon-Flex/faq/how-far-from-the-pick-up-location-do-you-travel-does-amazon-send-you-a-block-that-is-in-the-vecinity-of-your-address?quid=1brfehmifas2id9m

[3] https://www.statesman.com/story/news/local/flash-briefing/2019/02/13/how-many-hours-did-average-austin-driver-spend-in-traffic-last-year-over-100/5998795007/