COE 322 Final Project

Amazon Delivery Route Optimization

Hart, Preston
preston.hart@utexas.edu

Mason, John Matthew
jmmason@utexas.edu

December 7, 2020

1 Introduction

Route optimization is a challenging, yet important and applicable problem to tackle in today's world of online shopping and at-home delivery. Specifically, the Amazon delivery truck scheduling problem, outlined in chapter 55 of Professor Eijkhout's *Introduction to Scientific Programming in C++/FORTRAN2003*, is a twist on the traditional *Traveling Salesman Problem* (TSP). In addition to the guidelines of the TSP, a customer is guaranteed their package will be delivered within a certain range of days, unless it is an Amazon Prime order which guarantees next day delivery. Previous route optimization algorithms have to be modified to account for packages with these newly added weights tied to them. The method of optimization for the Traveling Salesman Problem pushes the problem into at least nondeterministic polynomial time because of the exponentially growing run time as the list of delivery addresses increases (Hoffman, Padberg, Rinaldi, 2001). The methodology behind the optimization is to remove the overlap, or crossover, in the original route of a delivery truck. The route can then be reordered to minimize the total distance traveled between all of the delivery locations for a day. The crossover optimization method used in this solution of the Amazon delivery truck scheduling problem is known as the Kernighan-Lin Method. However, the difficulty of optimization is amplified to even greater degree when a package is guaranteed next day delivery by Amazon Prime. Ultimately, different methods and procedures were researched, used, and tested through a day-to-day simulation to evaluate the performance of the Kernighan-Lin optimization algorithm on Amazon delivery routes.

2 Preliminary Considerations

Before we were able to begin exploring various methods for route optimization, we first created a virtual 'environment' which could easily facilitate the design and implementation of our simulations. We selected the object-oriented programming language C++ so we could employ abstraction and polymorphism when creating our environment. By default C++ does not have a function specific to route optimization and there are no types which resemble elements of our problem. We leveraged the power of object-oriented programming to create classes which represent key components of the route optimization problem and defined actions specific to each component. Through our class implementations, we taught C++ how to understand components of our problem allowing us to decrease implementation time and increase readability. In this section, we will discuss some of these classes and their importance to the problem.

2.1 The Address

The Address is the most basic component of the route optimization problem. An address object contains Cartesian coordinates and delivery priority information and represents a particular delivery which needs to be made.

2.2 The Address List

The Address List class is where the core of our framework was built. At a basic level, the Address List contains a set of addresses which represent a delivery route. However, there are many functions we have added to this class to allow for easy implementation of optimization methods. A few important examples are:

- Address addition, removal and update functions
- Smart retrieval functions
- Length computation function
- Reordering functions
- Other supporting geometric functions.

Because delivery trucks must start and end their day at the depot, defined as the origin (0,0), we added constraints to our constructors, addition, removal, and update functions to make sure that any given Address List always begins and ends at the depot. Since there is not a (reasonable) case where a truck would not begin and end the day at the depot, we chose to implement this constraint here as opposed to in the Route class.

2.3 The Route

After the Address List framework was built, we developed the Route class. The Route class inherits from the Address List, but also extends its functionality by including various optimization methods. We will discuss these methods in greater detail in Section 4: Optimization Methods.

3 Test Cases

We defined the following set of points as a test case to compare our method's performance:

```cpp
#include "route.hpp"

int main() {
    Route deliveries;
    deliveries.add( Address(10,7) );
    deliveries.add( Address(4,8) );
    deliveries.add( Address(3,13) );
    deliveries.add( Address(-10,5) );
    deliveries.add( Address(7,10) );
    deliveries.add( Address(0,10) );
    deliveries.add( Address(-3,6) );
    ...
    return(0);
}
```

4 Optimization Methods

4.1 The Greedy Method

The Greedy Method is the simplest optimization method we considered. The Greedy Method starts at the beginning of the list, the Depot, and moves to the next closest address. This process is repeated until every address has been considered. We observed that this method did offer a significant improvement to route efficiency versus unsorted routes, but it was far from perfect.

The Greedy Method is severely handicapped by its shortsightedness. Because the method is designed **only** to look for the next closest point, it lacks any consideration for what addresses it has already passed up, and what addresses the route will need to stop at in the future. This leaves room for a variety of inefficiencies if the given set of addresses leads the Greedy Method along an unusual path.

One of the easiest inefficiencies to identify occurs when the route crosses over itself. We addressed this problem in Section 4.2: The Kernighan-Lin Method.

4.1.1 Results

```
$ g++ -o myprogram.o address.cpp addresslist.cpp route.cpp 55-3.cpp
$ ./myprogram

The Greedy method found the following optimized route:
[0,0]
[-3,6]
[0,10]
[3,13]
[7,10]
[4,8]
[10,7]
[-10,5]
[0,0]
The original length of the route was: 75.0809.
The optimized length of the route is: 61.9192.
```

Using the test case defined in Section 3, we observed an **18% decrease** in total route length.

4.2 The Kernighan-Lin Method

The Kernighan-Lin algorithm is a heuristic solution to the *Traveling Salesman Problem* which addresses a key flaw in the Greedy Method: routes which cross over themselves. The algorithm can be summarized as: If any part of the route crosses over itself, reverse the enclosed part of the route and confirm the result is shorter.

When run by itself, the Kernighan-Lin algorithm did not always produce a better result than the Greedy method. However, when used after the Greedy Method we did observe occasional moderate improvements in total route length. Although the improvement beyond the Greedy Method was small compared to the improvement the Greedy Method made to the unsorted list, we did observe a consistent improvement using the Kernighan-Lin algorithm in almost every test case we tried.

4.2.1 Results

```
$ g++ -o myprogram.o address.cpp addresslist.cpp route.cpp 55-4.cpp
$ ./myprogram

The Kernighan Lin method found the following optimized route:
[0,0]
[10,7]
[4,8]
[0,10]
[7,10]
[3,13]
[-10,5]
[-3,6]
[0,0]
The original length of the route was: 75.0809.
The optimized length of the route is: 63.8051.
```

Using the test case defined in Section 3, we observed a **15% decrease** in total route length.

```
$ g++ -o myprogram.o address.cpp addresslist.cpp route.cpp 55-6.cpp
$ ./myprogram

```

```
The unsorted route length is 75.0809.
The Greedy optimization method decreased the length to 61.9192.
The Kernighan Lin optimization method decreased the length to 54.3889.
```

When we ran the Greedy Method first, followed by the Kernighan-Lin method, we see that the Kernighan-Lin method yielded a **13% improvement** beyond the Greedy Method alone.

5 Multiple Trucks

We also considered a case where the delivery company uses two trucks instead of one for daily deliveries. Implementing this solution was simple: first we used the Kernighan-Lin method to optimize the route for one truck, then we divided the route in two and assigned half the deliveries to one truck and half to the second truck. Through a variety of test cases, we were able to make a couple of observations about the practicality of the two-truck method.

Most importantly, we observed that in almost every case the total distance driven was **greater** when two trucks were used versus one truck. Although the packages would have been delivered in less time, the cost of running two trucks would be significantly higher than a single truck. Therefore, we conclude that running multiple trucks on the same day is not beneficial until the point where a single truck may not be able to complete all deliveries on time or in one day is reached.

Nonetheless, the Multiple Trucks exercise served as a good framework for implementing the Priority Delivery model discussed in Section 6.

5.0.1 Results

```
$ g++ -o myprogram.o address.cpp addresslist.cpp route.cpp 55-7.cpp
$ ./myprogram

The original length was: 75.0809.
The route for Truck 1 is:
[0,0]
[10,7]
[7,10]
[4,8]
[0,0]
The length of Truck 1s route is: 28.999.
The route for Truck 2 is:
[0,0]
[-10,5]
[3,13]
[0,10]
[-3,6]
[0,0]
The length of Truck 2s route is: 42.3955.
```

In this test case, we notice that the sum of length(Truck 1) and length(Truck2) is less than the original length. However, for many other iterations this was not the case, especially when the addresses were further from the depot.

6 Priority Deliveries

The priority, or Prime Delivery, method adds a new parameter to the Multiple Trucks algorithm: each delivery would either be a standard or a priority delivery, indicated by a Boolean value. Priority deliveries were assigned to a particular truck and could not be switched between trucks. This constraint somewhat limited the amount of optimization we could do on the two truck method.

To implement the priority delivery model, we created a function which identified locations of crossover between the trucks' routes and could check to see whether the deliveries were ”priority” before switching them. Although in many cases, this restriction caused (sometimes significant) increases in the total route length, priority deliveries are an important logistical distinction. Appropriately categorizing these deliveries not only makes our model more accurate, but also helped us best optimize the route based on the customers' needs.

6.0.1 Results

```
$ g++ -o myprogram.o address.cpp addresslist.cpp route.cpp 55-8.cpp
$ ./myprogram

Original Length: 63.8051

Route assigned to Truck 1:
[0,0]
[4,8]
[0,10]
[10,7]
[0,0]
The length of Truck 1s route is 36.0633.

Route assigned to Truck 2:
[0,0]
[-3,6]
[-10,5]
[3,13]
[7,10]
[0,0]
The length of Truck 2s route is 46.2502.
```

With the added limitation of priority deliveries, there are significantly fewer opportunities for route optimization. Shown above, the sum of the two routes is 80, almost **27% greater than without priority deliveries.**

7 Simulated Testing and Randomness

In order to make the delivery scheduling more realistic, a randomness factor and a day-to-day delivery simulation were added to our model. The randomness factor came from an implemented function *randomAdd* that generates a Route through the creation of random Address objects that contain X and Y values ranging from -100 to 100, and either a prime or no prime distinction. For simple testing of these methods, addresses had a 30% chance of being Prime. Then, the two truck optimization method mentioned above is performed on this random route. The two trucks' routes then contain a mixture of both Prime and not Prime deliveries.

In order to define what a ”day” was in the day-to-day delivery simulation, a maximum distance variable was created to limit the number of deliveries a single truck could make in one day. However, one key aspect differentiating this problem from the traditional *Traveling Salesman Problem* is the next day delivery guarantee of Prime packages. In order to satisfy this condition, Prime deliveries must be included in the next day's route in place of non Prime orders that may have been placed before the Prime orders. The filtering out of non prime orders in a truck's route was done through the implementation of the function *ReorderRoute* which is called after the original route is split and optimized between the two delivery trucks. The *ReorderRoute* function searches through the passed in route, selects all of the prime locations, adds them to a newly defined route, and removes them from the original route. This newly defined route is then optimized through the Kernighan-Lin method mentioned in the above sections and the distance of the route is calculated. If the distance of the route is less than the maximum distance for the day, then non Prime orders are added on to the new route until the optimized distance of the route exceeds the max distance. At the point the max distance is reached, the last added address is removed from the new route and all addresses still in the original route are salvaged in order to combine them with the leftover addresses from the second truck.

All packages in the newly created route are then all assumed to be successfully delivered at the different locations. To end the simulated day in the main method, new deliveries are added to the leftover addresses from the trucks to create the route for the next day. This workflow is outlined through the output of two simulated days shown below.

7.0.1 Output

```
$ g++ -o myprogram main.cpp Address.cpp route.cpp route.hpp address.hpp addresslist.cpp
addresslist.hpp
$ ./myprogram

Day 1
[0,0]
[-28,-30]
[6,3]
[-34,10]
[-4,11]
[27,5]*
[-8,-1]*
[-25,5]
[-9,14]*
[7,1]*
[-16,-31]
[-3,0]
[6,-27]*
[14,31]*
[32,-15]
[5,14]*
[0,10]
[0,0]

Route assigned to Truck 1:
[0,0]
[-3,0]
[-8,-1]*
[-25,5]
[-34,10]
[-9,14]*
[-4,11]
[27,5]*
[0,0]
Route Length: 126.606

Route assigned to Truck 2:
[0,0]
[6,3]
[7,1]*
[5,14]*
[14,31]*
[6,-27]*
[0,0]
Route Length: 127.54

Day 2
[0,10]
[32,-15]
[-16,-31]
[10,6]
[10,-10]
[-23,20]
[8,-31]
[-40,-38]
[-6,-44]*
```

```
[-36,12]*
[-32,16]
[16,-14]
[1,17]
[40,-7]
[-42,19]
[5,26]
[-35,-22]*
[-28,22]
[43,21]
[-28,-30]

Route assigned to Truck 1:
[0,0]
[-36,12]*
[-6,-44]*
[0,0]
Route Length: 145.884

Route assigned to Truck 2:
[0,0]
[10,-10]
[8,-31]
[-16,-31]
[-28,-30]
[-35,-22]*
[0,0]
Route Length: 123.249
```

Shown above is the simulated delivery routes and lengths of the routes specific to each truck for two simulated days. The maximum distance a truck could travel in one day was set to 130 units for this simulation. The first output for each day is the original route a single truck would take. Underneath that route are the deliveries each truck will make according to the optimization process described above and the length of that specific route. A "*" printed next to an addresses means the package at that location is a prime package. We know our simulated day-to-day process is working correctly, because prime packages from the first day do not appear in day two's list and packages that were not delivered in day one are being carried over into the second day.

Clearly seen in the output above is the ability for the number of leftover deliveries to skyrocket as the days of the simulation increase. This is a flaw in the day-to-day approach, because new addresses are only added to the pending delivery list at the end of the day, rather than a continuous addition of new orders throughout the day.

It is also worthy to note that this method was designed for prime deliveries to take precedence over the max distance. Seen in the output of day 2, truck 1's route exceeded the max distance of 130 in order to make sure all the prime deliveries were delivered within the guaranteed time frame.

8 Conclusion

Throughout the course of this project, we successfully implemented an object-oriented framework which allowed us to quickly prototype and test various route optimization methods. We also successfully implemented the Greedy Route method, the Kernighan-Lin method, and extended these methods to consider factors like multiple trucks and priority deliveries. At each stage of our progress, we recorded the strengths and weaknesses of each model we tested so that we could move towards the most realistic simulation possible. We discovered that route optimization is not a problem easily solved with a set of heuristics, but instead involves weighing various trade-offs, such as using two trucks to make deliveries faster but increasing total distance traveled. Moving forward, there are numerous other variables which would help our program make better informed decisions based on a set of defined goals.

For example, if the goal was to **minimize total cost**, information such as gas price, driver pay, and maximum acceptable delivery times would allow our model to not only predict the shortest route but also optimize find a balance between cost and delivery speed.

In another case, if the goal was to **minimize delivery time**, we could design a model to favor putting fewer deliveries on each truck and adding more trucks to the simulation. However, having this additional information would allow the model to also show the effects this focus would have on total cost and miles driven.

To better optimize the simulation to fit a more realistic scenario of delivery scheduling we could design a workflow of route optimization with a more life-like continuous addition of orders. This approach would incorporate the same process of optimization for each truck's route based on distance and prime deliveries. However, a truck would likely need to return to the depot before the end of the day to pick up new packages. Another solution strategy is to offset the start time of the trucks so that when a truck did return to the depot new orders could be incorporated into a new optimized route for the rest of the day. Ultimately, more reasoning and logic would need to go into fully panning out this approach, but in the end it would prove to be more aligned with the real flow of Amazon deliveries.

Below is a compiled list of a variety of factors which would be potential candidates for consideration and optimization:

- Total Cost
- Miles Driven
- Number of Drivers
- Maximum Number of Days to Delivery
- Vehicle Maintenance Costs
- Continuous Delivery Additions
- Likelihood of Unexpected Route Interruptions, for example: traffic, accidents, and lost packages

Overall, this project was a success. Not only did we successfully implement a variety of optimization methods and demonstrate that they preformed as intended, we also learned a lot about route optimization and were exposed to the strengths and weaknesses of in order to make improvements going forward.

9 Work Cited

Hoffman, K. L., Padberg, M. L., & Rinaldi, G. L. (2013). Traveling Salesman Problem.

Encyclopedia of Operations Research and Management Science, 1573-1578.

doi:10.1007/978-1-4419-1153-7_1068