Eric Last

Data Structures and Algorithms

Professor Xia

7 May 2017

1. **Introduction**

Project 3 brought back a familiar coffee shop, and since they were so impressed with our work, they assigned us to do an analysis on a corporate level. The job now was to plan out their shipment deliveries in the city of Coffeetown, a city with a 100x100 perfect grid layout. Company coffee shops and warehouses exist only at the intersection of roads, creating a perfect scenario for the use of a Graph. We were to optimize the routes taken by truck drivers so that all deliveries are processed in one day and the total distance traveled by the trucks was minimized. This project proved to be challenging in its own right, especially given the many possible algorithms we could use to tackle this problem.

1. **Approach**

This problem is NP-hard, meaning there is no concrete solution, and finding an optimal approach is very difficult. Chris Mayer and I teamed up to work together and find what we believe to be the most effective algorithm to tackle the problem set out for us. Obviously, we used a Graph as our choice of data structure, but we figured that a Graph with 10,000 vertices, one at each intersection, was way too difficult and excessive for the problem we were trying to solve. Instead, we used a Weighted Graph approach, with edges connecting every Warehouse to every Shop, and edges connecting every Shop to every other Shop. This made it so that any delivery could be carried out effectively. To calculate the weights of each edge, we simply added together the x and y distances between each vertex.

After trying several different algorithm structures, we finally came across one that was very effective. We first tried sending a Truck from each Warehouse to the closest respective Shop. The Truck would then proceed to the next closest Shop from the one it was currently located at until it ran out of cargo and returned to the original Warehouse. We tried a combination of ways to make this work, first by sending out one Truck at a time from each Warehouse, then sending out every Truck from one warehouse and repeating. We found that both of these approaches had too much overlap; the trucks would hit shops that were much closer to a different Warehouse, but since it was closest to a Shop already on the route, it got picked up by a farther away Warehouse. While this algorithm was effective and gave us good results, we realized that there are much better ways to optimize the path the trucks take. Our final algorithm used local proximity, rather than just finding the closest Shop at each step. More on how the algorithm works will be explained in the **Methods** section of this report.

A lot of classes were used in this project. Graph, Vertex, Edge, CoffeeShop, Warehouse, and Truck were all used as the basis of creating structure for testing. Vertex is a parent class of CoffeeShop and Warehouse to effectively use polymorphism to its best potential. Each of the two child classes store valuable methods for finding other vertices and instance variables that distinguish them from each other. The Graph class stores all of the Vertex and Edge data to fully construct a graph to be used as a data structure, and the Truck class was implemented as a fundamental piece to the Warehouse class to plan routes and carry out deliveries. We also utilized a Simulation class to carry out the full process of running the deliveries.

1. **Methods**

As mentioned earlier, it took us a while to finally comprehend the problem we were dealing with and the most optimal solution. Eventually, we figured out something that worked the best for us. We imagined the scanning range of a Warehouse to be like a spiral, hitting closer shops first before trying to reach the ones further away. But sending the trucks from the warehouses wasn’t giving us the best results, as there was still a bit of overlap. Instead, our final approach had a random Shop that still had orders remaining call the closest Warehouse to send a Truck. This randomized the order in which we sent out trucks so that the potential overlap was decreased. The Truck then delivers to the closest Shop (not necessarily the one that called it) that still has orders remaining. Once at the Shop, the algorithm searches for the next Shop where the distance between the current and next is minimized as well as the next to the original Warehouse. This method was repeated until the Truck ran out of cargo and had to return. Then another random Shop calls for a Truck and the process repeats itself until all shops have their orders satisfied.

To conduct experiments on this algorithm, we used both random cases and fringe cases to see all the possibilities. We came to a quick conclusion that fringe cases are brutal on our algorithm, however the layout of these cases is extremely inefficient for the coffee company, and they would be uneducated to franchise in such a manner. For example, we tested a care where there are shops on every corner of the two diagonals going across the grid (198 shops) and warehouses in central locations, equidistant from the diagonals. This layout is very hard on our program because the warehouses are so far from the shops and even chaining four or five shops in one trip still takes well over 200 units of distance. We also tried a similar case where there was a dense population of shops in an area where there weren’t many warehouses. This is also a very inefficient layout for a similar reason: entering a dense area with a limited amount of supply means turning around and sending another into the same area, essentially tracing over distance that has already been covered. It seemed that the random cases worked best, and as long as the spread was evenly distributed, it led to good results.

1. **Data and Analysis**

We were extremely pleased with how our data turned out. Using the test cases provided to us, we were able to come very close to the tested values. By printing out information on each Truck, we were able to see the routes they took and mapped them out to realize the efficiency. It turns out that each Warehouse utilizes all of its trucks before the main Warehouse (which has a very large supply of trucks) starts to. Since the main Warehouse is always at the bottom-left corner of the Graph, there are most likely warehouses that are much closer to shops in the general area. Using the test files, we produced numbers that were extremely close: warehouses1.txt produced 13630 distance units (expected 13838), and warehouses2.txt gave us 9744 distance units (expected 9770). While these numbers aren’t exactly the same, they are close enough to say our results were indeed expected. We also ran a few cases which were easy enough to compute by hand. First (can be run using shops2.txt and warehouses3.txt), we densely populated the top right corner with 10 shops and one warehouse, placed another warehouse a few blocks away, and had the main warehouse in the bottom left. We ran our algorithm by hand and produced a value of 102 which is the exact value the program generated as well. Another case (can be run using shops4.txt and warehouses5.txt) contains a graph with one central warehouse and a shop populating every possible corner around that warehouse in a 5x5 square (24 shops and 1 warehouse), plus four more shops extending from the middle row (see Figure 1).

Figure 1: Orange square represents warehouse, circles represent shops

The program produces a value of 86 distance units, which by hand calculated a total of 74. This seems minute, but with a larger vertex set (of maybe 200) this “small” margin of error becomes a huge factor in optimal pathing. The greedy approach we took was not enough to satisfy a graph like this, but as mentioned before, the coffee company would be ill-advised to lay out their shops in a layout that was less than optimal.

A final test case (using files shops3.txt and warehouses4.txt) was created by using a series to calculate distances between vertices. We guessed a layout that seemed pretty ineffective, and produced almost that. The graph is hard to represent using figures, but essentially consisted of a square around a vertex and vertices coming off the corners of the square. We ran the algorithm by hand first and found that the main warehouse at (2, 2) did not need to be used, so we got a distance of 90. However, the program showed us that it was impossible to fulfil every demand without sending an extra truck from the main warehouse and returned us a value of 390 distance units. The algorithm didn’t break down here, it just showed us how human error can play a large role in calculating this information.

1. **Conclusion**

In conclusion, it turns out that the greedy algorithm we used in this project was extremely effective in getting the job done for “normal” graphs. The biggest issue the algorithm runs into is dealing with straight or diagonal lines. Since our program checks for the closest shop from the one we are currently at in regards to its distance from the warehouse, it will travel down straight lines until it runs out of cargo, then will turn around and tackle the other segment of the line, which is extremely inefficient. Otherwise, however, the algorithm is great and does exactly what it needs to do. I am very happy with the way the program developed, and could not be more thankful and grateful to have worked with Chris on this project. While our minds work differently and we have different coding styles, we were very good at working together and finding solutions as a team.

1. **References**

"Java Platform SE 7." Java Platform SE 7. N.p., n.d. Web. 7 May. 2017.

Xia, Ge. CS150: Project III. N.p: n.p., Apr 2017. PDF.