Cameron Zurmuhl

CS 150

Professor Xia

7 May 2017

Project 3 Report

I. Introduction

This project presented the travelling salesman problem. The problem asks from a single vertex in a connected graph, what is the shortest path to every other vertex and back. The context involves coffee shops and warehouses on a 100x100 connected grid (a weighted, directed graph). The shops and warehouses serve as vertices, and edges connect every shop and warehouse together with weight equal to their distance apart. The shops have cargo orders that delivery trucks need to satisfy in one day; however, the trucks cannot load past 500 units of cargo. If the truck does fill up to this capacity, it must return to its originating warehouse. Each truck must deliver a full cargo-order for a specific cargo; there are five different types of cargo. The number of trucks per warehouse varies, and there exists one "master warehouse" at the edge of the grid with unlimited trucks. The objective of the project is to minimize the travelling distance of all trucks while satisfying all the shops. This problem is NP-hard, meaning it cannot be solved in polynomial time. Thus, I took a greedy approach in solving the problem. In my analysis, I will analyze the efficiency of my optimization. My hypothesis is that my solution is efficient in most cases, but not the best.

II. Approach

In my solution, I mapped the shops and warehouses on a weighted, directed graph. Every shop is connected to every other shop and warehouse with edge weights equal to their total horizontal and vertical separation distance. Each warehouse as a stack[2] of trucks of varying size. The general idea is that once a truck leaves a warehouse, the truck will go to the closest possible shop and fulfill all its needs possible. There are three checks that happen before a truck can go to a shop. First, a truck can only go to a shop if it can fulfill any cargo needs. Second, the truck can't go to the same shop twice. Last, a truck can't go to a shop where there is a warehouse closer by. This last condition only applies to

1. "Package java.util (Java Platform SE 8)" Oracle, 2016. Web. 8 March. 2017. https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
2. "Package java.util (Java Platform SE 8)" Oracle, 2016. Web. 8 March. 2017. https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

trucks that have visited at least one shop, so every warehouse (besides the master one, of course) utilizes all their trucks. If any of these checks fail, the truck will go to the next closest shop to fulfill those needs. The truck will continue this algorithm until it is either full or it has seen every prior shop. Then, it will return to its originating warehouse. To find the closest shop, I store all neighboring facilities to a given shop in an ArrayList [1] and sort them by edge-weight (distance). I dispatch one truck at a time, one warehouse at a time, starting with the first warehouse, the second warehouse, and so on to the tenth warehouses. When the first nine warehouses run out of trucks, I only use the unlimited warehouse for trucks. Last, I sorted each shop's cargo order by weight largest to smallest, so each truck will pick up the shops' largest orders first.

III. Methods

I came up with five scenarios to test my solution. Scenario one involves one warehouse with one truck surrounded by shops. I have one truck going to each shop. The cargo orders are light enough so the truck can pick up every cargo and return. The optimal distance in this scenario is 8 units.

Figure 1: Set-up for Scenario One. Shop ID in white, cargo order in yellow.

Scenario two involves two warehouses with one truck each. The warehouses are distinctly apart from each other, and five shops are clustered around each warehouse for a total of ten shops. The cargo orders for each shop is 100 units each. The optimal distance in this scenario is 22 units. Scenario three replicates scenario two, except cargo orders are mixed and under 100. The optimal distance is still 22 units. Scenario four replicates scenario three, except cargo orders are mixed—some under 100, some over 100, and there are two trucks per warehouse. The optimal distance is 28 units.

Figure 2: Set-up for scenario two, three, and four. Second Scenario: all orders 100 units. Third Scenario cargo load in yellow, fourth in pink.

Scenario five is like scenario four, except there are only five shops huddled around one warehouse. The optimal distance is 12 units.

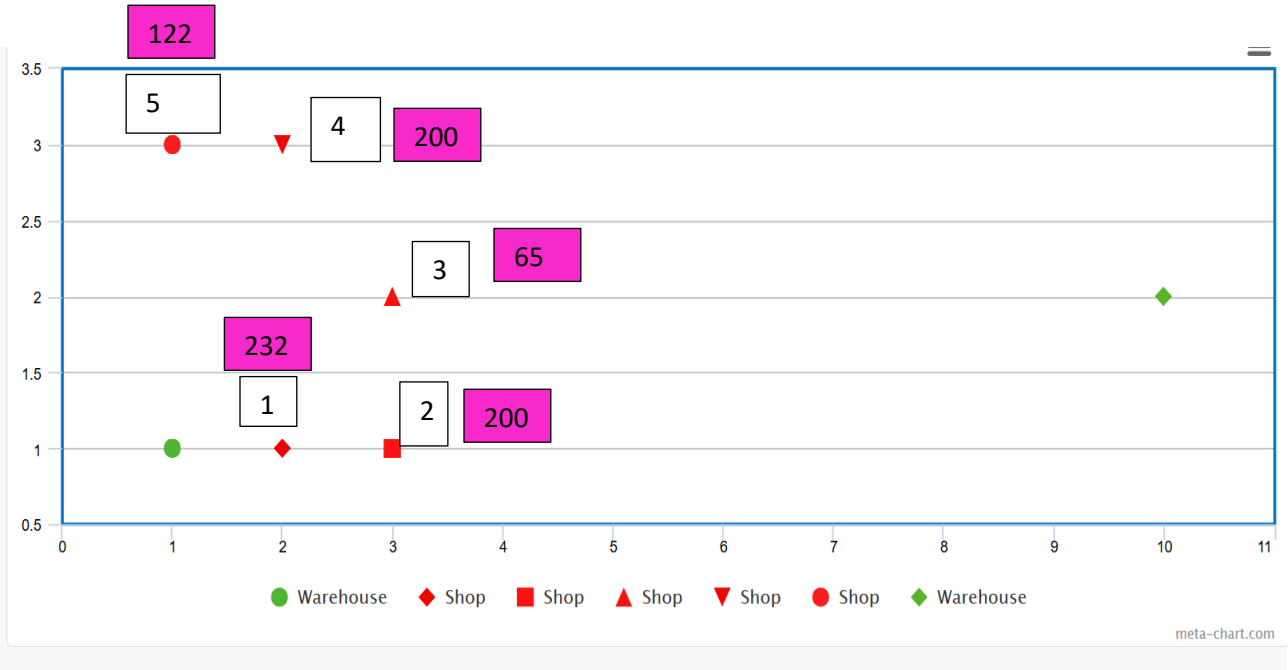

Figure 3: Set-Up for scenario 5. IDs of shops are in white, cargo loads are in pink

IV. Data and Analysis

In this section, I will go through each test case and explain what the best route is and compare that route with my output. I will denote shops with the capital letter "S' and warehouse with capital letter "W".

Case1: Best route is S1, S2, S3, S4, S5 from W1 for a route length of 8.
Output: S1, S2, S3, S4, S5 from W1.

Case2: Best route is S1, S2, S3, S4, S5 for W1, and then S6, S7, S8, S9, S10 for W2.
Route length: 22
Output: Matches analysis

Case3: Best route is the same route as last case. Distance is still 22.
Output: Matches analysis

Case4: Best route is S1, S2, S3, S4 from W1 then return. Next, S6, S7 from W2 then return. Then from S5 from W1 then return. Finally, S9, S8, S10 from W2 then return. Total Distance: 28
Output: Matches analysis

Case5: Best route is S1, S2 S3 from W1 and then return. Next, S4, S5 from W1 and return. Route Length: 12
Output: S1, S2, S3 from W1 then return. Next, S4, S5 from W2 and then return. Total Distance: 26.

Thus case 5 represents a flaw in the solution. Since I have every warehouse in the case using all their trucks, a warehouse may dispatch a truck across the grid when there are warehouses closer by to those shops. However, if shops are distributed across the grid uniformly, this should not be too much of an issue, as seen when the program is run with the test file[1]

Since case 5 represents a flaw, I will attempt to make the algorithm greedier by setting a distance limit trucks can travel. If they go over this limit, they must return. If improvements are noticeable, then my solution is not optimal. If distances get worse, then my solution is a valid greedy approach.

Any restrictions below 10 for my test cases result in exceptions (shops were not satisfied). Thus, I will use the provided test files "shops.txt" and "warehouses1.txt".

[1]Results were 12190 for 'warehouses1.txt' and 7968 for 'warehouses2.txt', compared to given reference values of 13838 and 9770, respectively

Distance Restriction	Distance Travelled
None	12190
100	14900
150	12500
175	12190
200	12190

Figure 4: Table representing changes in total distance travelled when there are restrictions. Input file used: "warehouses1.txt"

As the table shows, restrictions on distance trucks can travel do not better performance, rather make the total distance travelled the same or worse.

Next, I will show that sorting the cargo by heaviest first is part of the greedy solution. Using "warehouses1.txt", The distance travelled when the orders are picked up heaviest first is 12190. The distance travelled when the orders are not sorted is 12540. Using "warehouses2.txt" the distance travelled when the orders are picked up heaviest first is 7968. The distance travelled when the orders are not sorted is 8240. Thus, improvements are noticeable when cargo is picked up strategically.

Overall, the algorithm demonstrates efficiency by using greedy approaches. Comparing to the reference values, the results were reasonable. When trying to further optimize the program, either shops were not fulfilled or distances were increased. To optimize it further, you would have to take in account all combinations of shops that could be visited along the way, and which combinations of cargos are best to pick up and with shortest distance. This would require artificial intelligence. Perhaps an easier optimization technique is to not require every warehouse to use each truck. When applying that change to the program, all my tests perform at the optimal distance. This is obviously because W2 in case 5 will not be used. However, when applying the change to the test files provided, I get worst results. "Warehouses2.txt" results in a distance of 8684 and "warehouses1.txt" results in

12310. Further tests should be conducted. Perhaps this change may help if warehouses were more analytically selected to dispatch trucks, rather than dispatching one at a time from W1-W10.

V. Conclusion

From the collected data and the test results, I conclude that our solution is efficient. Our distances were 1000 distance units under the reference values, and the most optimal path was used in most test cases. Of course, there were inefficiencies as the algorithm takes a greedy approach and not the optimal approach. Since the problem is NP hard, for our purposes, the task is to reach efficiency. If the shops are spread out uniformly, my theory is that test case 5 would not have that much of an impact as it does in the test case. Comparing the collected data to the reference values supports this idea. Plus, when I set distance restriction on how far trucks can travel, results worsened.

There are other solutions to the problem. Perhaps to make the algorithm more efficient, one can program the computer to strategically select which warehouses dispatch trucks when and where instead of requiring warehouses 1-9 to use all the trucks before using the main warehouse. Similar ideas leave the program open to more unit testing. Overall, our algorithm's key components of visiting closest shops first to satisfy all possible needs, going to shops that don't have warehouses closer by after at least one shop visit, and picking up largest cargo first demonstrates an efficient solution, but perhaps not the best in some cases.

VI. References

1. "Package java.util (Java Platform SE 8)" Oracle, 2016. Web. 8 March. 2017.

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

2. "Package java.util (Java Platform SE 8)" Oracle, 2016. Web. 8 March. 2017.

https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html