WGUPS Routing System

WGU - Data Structures and Algorithms C950

Kaitlynn Abshire

The problem proposed is loading three trucks with a total of 40 packages in a total distance of under 140 miles. Each truck can only hold 16 packages and moves at 18 miles per hour. Some packages are delayed, marked with the wrong address, or can only be placed on unique trucks.

Algorithm Identification

The algorithm used in this solution is the nearest neighbor algorithm. The nearest neighbor algorithm searches for a point that is the next closest to the given point and sets it as the new minimum. There are two main forms of nearest neighbor which are Hart's and Wilson's nearest neighbor (Yu, Ji, & Zhang, 2002). Wilson's version uses a "leave-one-out" format where it removes any accidentally referenced points while the Hart formula keeps all points. The formula used in this solution compares the amount of miles between each package delivery location to find the smallest one. It sets this as the new minimum and takes the truck to that location. After the delivery it repeats the loop for each location until the best route is found. This formula also checks for a distance of 0 miles to account for the need to deliver multiple packages to the same location if they are labeled as such.

Other algorithms that could be potential solutions to this problem are Dijkstra's algorithm which uses a point from a graph and locates the shortest path based on edge weights (Broumi et al., 2016). This would require mapping out the next point using all prior points to determine the shortest route. This algorithm may provide a route in fewer miles than the nearest neighbor path; however, the nearest neighbor path was easier to comprehend.

Logic Comments

The nearest neighbor greedy algorithm takes the length of the normal path

```
while len(normal) != 0:
    min = [0, path]
    for place in normal:
        miles = edge[best[-1], place]
```

and runs it through a series of if statements to set the new minimum for the truck to travel to.

```
if miles < min[0]:
    min = [miles, place]

if miles != 0 and miles < min[0]:
    min = [miles, place]
```

It also checks to see if the package delivery location is at the current location for all mail in the truck to assure that it delivers all mail in the same truck at the same time to the same place efficiently. This also helps reset the truck back at the hub.

```
if min[0] == 0:
    min = [miles, place]
```

In order to remove the places the truck has already delivered from the greedy route, a remove function is used to remove the most recent minimum at the end of the given function. Otherwise, the truck would try to return to the same locations it has already delivered to.

```
if min[1] not in best:
    best.append(min[1])
normal.remove(min[1])
```

Development Environment

Visual Studio Code was used to create the program as it is a commonly used tool in Python jobs. The software is very friendly for Python applications and running csv files for excel data sheets needed in the code. One must download the program from https://code.visualstudio.com/ and then select a new file. It will ask what language is preferred and the user would enter python. Excel sheets are saved as CSV files and inserted into the same file as the project. The project was completed on a Windows computer using Google Docs to convert the excel sheets to csv.

Space-Time and Big-O

Space-Time complexity for major segments in the program are organized below.

Main.py

Function	Line #	Space	Time
load_trucks_efficiently	29	O(N^2)	O(N^2)
delivery_stat	44	O(N^2)	O(N^2)
delivery_stat	83	O(N^2)	O(N^2)
delivery_stat	99	O(N^2)	O(N^2)
send_mail	115	O(N^2)	O(N^2)
all_total_miles	116	O(N)	O(N)
Total	=	N+5(N^2)=O(N^2)	N+5(N^2)=O(N^2)

GreedAlgorithm.py

Function	Line #	Space	Time
greedy	10	O(N^2)	O(N^2)
Total	=	O(N^2)	O(N^2)

Trucks.py

Function	Line #	Space	Time
insert	22	O(1)	O(1)

delete	27	O(1)	O(1)
begin_delivering	32	O(1)	O(1)
update_time	36	O(1)	O(1)
end_delivery	141	O(1)	O(1)
load_trucks_efficiently	61	O(N^2)	O(N^2)
total_miles	131	O(N)	O(N)
all_total_miles	142	O(N)	O(N)
time_format	155	O(1)	O(1)
send_mail	163	O(N^2)	O(N^2)
en_route	223	O(N)	O(N)
delivery_stat	232	O(N^2)	O(N^2)
Total	=	6+3(N)+3(N^2)=O(N^2)	6+3(N)+3(N^2)=O(N^2)

HashTable.py

Function	Line #	Space	Time
insert	15	O(1)	O(1)
lookup	26	O(N)	O(N)
delete	36	O(N)	O(N)
get_mailboxes	46	O(N)	O(N)
print_result	59	O(1)	O(1)
Total	=	3N+2=O(N)	4N+2=O(N)

EdgeWeights.py

Function	Line #	Space	Time
add_point	13	O(1)	O(1)

add_weight	17	O(1)	O(1)
box_dict	22	O(N^2)	O(N^2)
get_distance	29	O(N)	O(N)
create_graph	40	O(N^2)	O(N^2)
Total	=	N+2+2(N^2)=O(N^2)	N+2+2(N^2)=O(N^2)

Scalability and Adaptability

Using the nearest neighbor greedy algorithm enables the project to account for scalability. More trucks could be added as well as more packages and the algorithm would still work the same. The time and miles taken would increase as more addresses for packages are added to the route; however, adding more trucks and drivers could reduce both. The nearest neighbor algorithm itself is adaptable as it's code can be altered into a Wilson's style nearest neighbor or a kernel type approach. It takes the data points and maps them out which is more sufficient in more complicated nonlinear problems (Yu, Ji, & Zhang, 2002). The code could also be altered to go to the furthest distance first and then make its way slowly back towards the hub stopping at each nearest location to the prior delivery.

Software Efficiency and Maintainability

The main algorithm along with many of the functions are O(N^2) making it efficient. The program runs smoothly and does not get stuck in loops. It is easy to maintain and add more code to. It is also easy to maintain the amount of trucks as well as mail. More packages can be added to the CSV while more trucks or drivers can be added to the trucks.py file. Errors are identifiable by adding a simple print function to the end of loops to see which loops are successful and which get stuck or do not occur.

Self-Adjusting Data Structures

The main data structure used in this program is a hashtable. The hashtable organizes the package data from the csv into 10 buckets. It is able to insert new data of packages to the table, lookup data within the table, and remove packages from the hashtable by identifying the package id (Lysecky& Vahid, 2018). A disadvantage with a hashtable is that if the data is not unique enough the data can run into collisions causing misrepresentation of data. For example, a lot of packages with the same address, but on different trucks can cause issues in the deliveries. The other self-adjusting data structure used is the edge weight graph of the distance data. The graph is made using the mileage in between locations as data points and established edges between the points. This adjusts by adding a new point and edge with new entries in the distance csv file. However, edge weight graphs are not good to use with a surplus of data points. Another self-adjusting data structure used is arrays of lists. The truck lists hold the mail data that is inserted inside of them up to 16 packages. The address array holds all addresses before they are inserted into trucks. The normal route arrays hold the original paths of trucks while the greedy arrays hold the best paths to take after the nearest neighbor is applied. The mile arrays hold the distance traveled by the trucks and the time is recorded by adding it to the time arrays. Old points and old times are adjusted in the arrays or removed. The issue with lists and arrays is that injecting specific data in a certain order can become difficult.

Strengths and Weaknesses of Chosen Algorithm

The chosen algorithm delivers all packages within 134.6 miles which is within the instructions for the program. It is an efficient method to find close locations to perform the intended result. The algorithm is adaptable and customizable having many variations. It can accommodate the addition of packages and trucks. It can also be altered to go to the furthest location and make its way back to the hub stopping at each location. The truck algorithm used was direct loading of trucks by presorting the mail into a truck based on special notes. This ensured all packages were accounted for and not double placed on trucks.

The nearest neighbor algorithm is not as efficient as some of the other algorithms. In fact, this same problem has been solved in shorter distances such as 76 miles. Other specific greedy algorithms will be discussed below. The truck algorithm could have been vastly improved by starting with priority sorting. Packages with deadlines could have been sorted into a first priority load. Delayed packages and packages that are required to be on truck two could have been sorted into second priority. The final no special notes packages could have been randomly assigned while the distance was recorded. The lowest distance random assignment could be saved and used out of all potential random assignments. This would have vastly improved loading and the distances traveled.

Other Possible Algorithms

Djikstra's shortest path algorithm is a common algorithm used in greedy path problems. It creates a graph and uses the edges connecting the points to determine the shortest route. It then creates a "shortest path tree" in which every point is checked against other points (Deng et al., 2012). This algorithm often produces faster routes than the nearest neighbor route as it goes more in depth. A second algorithm commonly used in traveling scenarios is dynamic programming. Dynamic programming also uses a set of vertices like nearest neighbor, but uses sub-problems to find the minimum route (Righini & Salani, 2008). However, this program usually has a run time total of O(N^2*2^n) making it less efficient as more packages and locations are added.

Implications

The addition of more trucks would speed up the deliveries only if there were more drivers with the same amount of packages or less packages. Increasing the amount of packages at different locations would increase the time and distance traveled in the program, unless more trucks and drivers were added to accommodate this. The algorithm would still function at O(N^2) and would not impact the software run time, just the output.

Other Data Structures

Almost every form of data structure in the zybook was used in this program except for binary search trees and balanced trees. Instead of searching the hashtable a binary tree could have been constructed with the two closest points to form the route (Lysecky & Vahid, 2018). Binary search trees work by ordering nodes by predecessor and successors, with the top being the hub and the below successors being the package locations.

Balanced trees alter themselves when a new point is added or deleted (Lysecky & Vahid, 2018). This tree could have also been used connecting the hub to all locations. It could have altered itself until it found the routes with least miles traveled by removing and adding nodes as needed.

References

Broumi, S., Bakal, A., Talea, M., Smarandache, F., & Vladareanu, L. (2016, November). Applying Dijkstra algorithm for solving neutrosophic shortest path problem. In *2016 International conference on advanced mechatronic systems (ICAMechS)* (pp. 412-416). IEEE.

Deng, Y., Chen, Y., Zhang, Y., & Mahadevan, S. (2012). Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, *12*(3), 1231-1237.

Lysecky, R., & Vahid, F. (2018). *C950: Data Structures and Algorithms II*. Retrieved from https://learn.zybooks.com/zybook/WGUC950AY20182019

Righini, G., & Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, *51*(3), 155-170.

Yu, K., Ji, L., & Zhang, X. (2002). Kernel nearest-neighbor algorithm. *Neural Processing Letters*, *15*(2), 147-156.