C950 Task-2 WGUPS Algorithm Implementation

(Task-2: The implementation phase of the WGUPS Routing Program)

Forrest VanDuren

ID #010620769

WGU Email: fvandu1@wgu.edu

Date: 8/31/2023

C950 Data Structures and Algorithms II

# D1. Status of all packages between 8:35 a.m. and 9:25 a.m.

The program showing all the statuses couldn’t fit in one screen shot. These two show all package statuses at 9:10 a.m.

A black screen with white dots

Description automatically generated

A black screen with a black background

Description automatically generated

# D2. Status of all packages between 9:35 a.m. and 10:25 a.m.

The program showing all the statuses couldn’t fit in one screen shot. These three show all package statuses at 10:00 a.m.

A black screen with many small colored text

Description automatically generated with medium confidence

A black screen with white and blue lines

Description automatically generated

A black screen with white text

Description automatically generated

# D3. Status of all packages between 12:03 p.m. and 1:12 p.m.

The program showing all the statuses couldn’t fit in one screen shot. These three show all package statuses at 12:05 p.m.

A black screen with white dots

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white and red dots

Description automatically generated

# E. Completion of program with total milage of all trucks.

This is the completion of option 2 of my python program. It shows the delivery time of all packages and then the total round trip milage of all trucks. From departure to the last return to the hub. And so should fulfil prompt E.

A pixelated image of a house

Description automatically generated

# F1. Describe two or more strengths of the algorithm used in the solution

The algorithm I used in this solution is called the insert method. Or the best insert or nearest insert method. The description of this algorithm was found in a paper called “Heuristics for Vehicle Routing Problem: A Survey and Recent Advances” (Fei Liu et al., 1). Strengths are described below.

1: It’s more adaptable than simpler algorithms like the nearest neighbor method. This is because instead of appending each next delivery location to the end of a list it can insert it anywhere. This allows it to adapt as new inputs are given and find a position for a given element that might be better than the end of the list.

2: Since the algorithm can pick more than one spot to insert at it typically offers more optimal routing.

3: The insert algorithm creates a more “rounded” route than a lot of other routing algorithms. For example, the nearest neighbor method might find a decent route from start to finish but since it’s always appending to the end and not taking previous locations into considerations you sometimes end up with a long trip back to your starting location. Since the insert method considers previous addresses, it generally leads to a route that rounds back to the start location more naturally.

# F2. Verify that the algorithm used in the solution meets all requirements in the scenario

The algorithm delivers all 40 packages on time using two drivers and without going over 140 miles for all trucks involved. It takes into consideration the notes given in the packages and delays loading for any package that is delayed. It also doesn’t route back to the hub for these packages until after they’re set to arrive and re-addresses package 9 with the correct address after 10:20. All truck2 packages are delivered with truck 2 and all packages required to go together are placed on the same truck. The program has a user interface that allows the supervisor or whoever else to enter any time of day and see the package status of all packages for both truck1 and truck2 including delivery times.

# F3. Identify two other named algorithms that would meet all requirements in the scenario

The nearest neighbor algorithm should work as well as the sweep algorithm. Both of which are defined in the paper cited in prompt F1.

A: The nearest neighbor algorithm differs by adding packages only to the end of the list. It calculates nearest neighbor by the package with the shortest distance from the last address in the route. Where the insert method checks all possible insert locations for all packages that haven’t been routed. It determines best insert by finding the lowest value of Δj. Δj is the minimum value of all Δij for a given unrouted address where Δij = Cij + Cj(i+1) – Ci(i+1). C is the distance between the two addresses in its subscript. Subscript i is the already routed package address it’s checking against, and j is the unrouted address. The sweep method takes a starting position and adds locations radially in a clockwise or counterclockwise fashion. All three of these methods-in fact any method-will likely need to be customized to some degree to fit the deadline constraints for those packages that have them.

# G. Describe what you would do differently if you did the project again

When I make the 2d distance list that holds all the distances I leave it one sided like it is in the excel sheet. This means that if I try to access a distance by putting the row number lower than the column number, I just get back a blank string. I figured this would be fine and I’d just check in an if statement which package ID is higher before getting the distance. It turns out this led to a lot more additional if statements than I thought. I had initially created a function that creates a new list that is the distance list rotated 180 degrees. I then merged the two lists together making it so you could use distanceArray[addressID1][addressID2] or distanceArray[addressID2][addressID1] and get the same distance. I thought the extra time taken merging these two was unnecessary so decided not to use it. After completing the program, I think the reduction of if statements may have been worth the extra time.

The main algorithm has a secondary algorithm that will adjust the route if the original solution makes any of the packages go outside of their delivery deadline. It’s essentially the same insert algorithm in a slightly different context. This is done recursively and while it works for the scenario given in some cases it can lead to infinite recursion. I didn’t feel it necessary to address this since it works well for the given scenario. However, doing it again I would create a second algorithm for reworking the packages if any go outside their deadlines. This algorithm would be designed to find a solution that worked even if it was less optimal and would never lead to infinite recursion. I would then keep track of the recursion depth. If the original algorithm hadn’t found an optimal solution that keeps everything within their deadlines by X number of recursions. The program would switch to the less optimal algorithm.

# H. Verify the data structure used meets all requirements

I used a hash table to store the package data as described in part A. I also created a package class and stored all the information in the package hash using package objects for each package. Any package can be found or inserted by ID. The data structure provides package address, deadline, city, zip code, weight, and delivery status. The package hash functionality isn’t dependent on the package class or any other class or library. So, it should fulfil that requirement stated in part A.

# H1. Identity two other data structures that meet the same requirements

A simple array could have been used. As well as a Packages class in the form of a linked list with pointers to each package object.

A: The array would have a longer search time than the hash table if it wasn’t sorted. You’d have to search each index by ID making the search time O(N) instead of O(1). Insertion could be done in O(1) time like the hash table in this case cause you could just put each new package at the beginning or end of the array. If you sorted it by package ID, you could search in O(1) like with a hash table. But the insertion time would be O(N) cause you’d have to find the right place to insert it. A linked list would have O(1) insertion times at the ends of the list. Matching the time complexity of the hash table insert function. However, whether you sorted the list or not you’d always be searching for a package in O(N) time.

# I. Sources

[1] Fei Liu et al. (March 03, 2023) Heuristics for Vehicle Routing Problem: A Survey and

Recent Advances