C950 WGUPS Algorithm Overview

Joshua Hillary

ID #005250603

WGU Email : Jhilla9@wgu.edu

December 12th, 2022

C950 Data Structures and Algorithms II

**A.  Nearest Neighbor Algorithm**

This project uses the Nearest Neighbor Algorithm to attempt to find the shortest path possible to

deliver packages from location to location. The function ‘Delivery’ requires a ‘truck’ object as a

parameter. A list of packages to be delivered is created and filled with the packages of the

truck object provided to the function. Then each next location node is looped through to compare

distances between the starting node until the optimal path through all locations is complete. This is

repeated for every truck and its list of packages.

In my own variation, I have also included a ‘weight’, that is produced by looking at all the next

nodes possible next locations, then dividing the number of next locations at the next node by the

total distances, which outputs a percentage that is multiplied by the distance from current

node to next node.

**B.  Write an overview of your program, in which you do the following:**

**1.  Explain the algorithm’s logic using pseudocode.**

Supply a truck instantiation to the function

Initialize a list of ‘packages to deliver’

Iterate through the truck’s packages

Add them to the ‘packages to deliver’

Clear the packages in the truck instantiation

While there are packages to be delivered:

Iterate through the packages to deliver:

Get the weights for all possible locations attached to the next node

If the distance is less than or equal to smallest distance

Add to var for smallest distance

Add to var for smallest weighted distance

Set current package to package

Load the package on the truck

Remove the package from the ‘packages to deliver’ list

Add mileage to deliver this package to the truck

Update the current address of the truck to delivery address

Update the time of departure for truck and package

Update the time of delivery for the package

The weighted portion can easily be removed for a better time complexity but less optimal total mileage. Remove the portions from the code and only check for distance between all connected nodes to the current node and choose the least far one.

**2.  Programming Environment.**

The project is written using the Python 3.7 language.

It’s implemented using the PyCharm IDE.

Operating system is Windows 10.

**3.  Evaluate the space-time complexity of each major segment of the program, and the entire** **program, using big-O notation.**

HashMap.py



Delivery.py



PrettyText.py



Trucks.py



Packages.py



Main.py



**4.  Explain the capability of your solution to scale and adapt to a growing number of packages.**

The nearest neighbor is less than optimal compared to other solutions as far as growth. It has a

time complexity of O(N^2), which means the bigger the N the slower it exponentially will get.

Although, it will work and adapt to any data set albeit slower depending on how big the N.

The data structure that contains all our package information is the chaining hash table. It is

adaptable because it can hold a growing size of data. It is more efficient because it creates a key

that can easily compute the placement of the item in memory, which greatly reduces the time

complexity of adding or retrieving items from memory. The only exception is during data collisions,

where an item must be sorted under the same key.

**5.  Discuss why the software is efficient and easy to maintain.**

The efficiency of the software is O(N^2) or O(N^3) depending on the algorithm you choose to use:

weighted Nearest Neighbor or the vanilla Nearest Neighbor.

The object-oriented class structure of the program makes it easy to retool to your needs. Trucks

and Packages are controlled via objects of their respective classes. So, adding a larger data set of

packages or locations can easily be handled by the current code. It can also easily be upgraded to

be even more self-contained and efficient, such as a heuristic to sort the initial packages that are

entered given to a given truck object to deliver.

The code comments I’ve left behind also help to increase its usability for anyone who may use it in

the future.

**6.  Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).**

Chaining Hash Table: it is an efficient algorithm for data, but its weakness is that the more

collisions it has the less efficient it can be. The bigger the data set the more likely a collision will

occur. Also, if you need to iterate over a large range of data in a hash table it is not the most

efficient solution.

List: Elements within a list have a strict order which can be beneficial depending on what you’re

required to do with your data, but its weakness lies in its requirement to iterate through the list to

sort or retrieve items.

**D.  Chaining Hash Table**

The hash table’s structure is defined by mapping keys to values in a list. The way the data is inserted

using a hash is by taking its key mod (the size of the list) and storing it in the associated space. This

makes retrieval much more efficient because it allows for quick calculation of its location in memory

without any rearrangement or iteration. Although, with more data collisions are possible, which could

significantly impact the efficiency of search and insertion. Chaining is used to accommodate this

issue.

1. **Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.**

Text

Description automatically generated

1. **Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.**

Text

Description automatically generated

1. **Provide screenshots to show the status of all packages at a time between 12:03 p.m. and 1:12 p.m.**

Text

Description automatically generated

**H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.**

A screenshot of a computer

Description automatically generated with medium confidence

**I.  Justify the core algorithm you identified in part A and used in the solution by doing the following:**

**1.  Describe at least two strengths of the algorithm used in the solution.**

One strength is in its ease of implementation. It doesn’t require much code to create powerful

results. Another strength is that it locates the shortest path based on the distances and creates a

more optimal path. There are algorithms that will provide even shorter paths, but the cost of them

is much greater than O(N^2).

**2.  Verify that the algorithm used in the solution meets all requirements in the scenario.**

The algorithm meets all requirements and completes the deliveries in 87.3 total miles compared to

the allotted 140.

1. **Identify two other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario and how they differ from the chosen algorithm.**

Dijkstra’s algorithm would meet the requirement of the scenario. Compared to Nearest Neighbor,

Dijkstra’s algorithm is a graph search algorithm that uses vertexes and edge’s, target nodes and

start nodes, to return the shortest path over the map in its entirety. Although, it does not work

well with changes to the map, requiring it to run multiple times to update appropriately.

There is also the Greedy algorithm, which always chooses the optimal solution in a local sense.

Seeking out the higher valued nodes (hence the term ‘greedy’), but what it chooses in the moment

might not be beneficial for future ‘steps’ where it could have chosen an overall higher valued path.

It’s similar in a sense that it’s local, but nearest neighbor only chooses the shortest distance,

whereas the Greedy algorithm is heuristic and does not allow the formation of closed loops.

**J.  Describe what you would do differently, other than the two algorithms identified in I3 if you did this project again.**

If I was to continue working on this project, I would experiment with different algorithms for improved efficiency while balancing the accuracy of its chosen path. For one, I could try and implement it with the greatest accuracy, the other the greatest speed with a relatively comparable result. I would also find an automated solution for loading packages into the trucks because as it stands, I have manually chosen which packages go into which truck. I could also improve my weight function so that it does not tax the Nearest Neighbor algorithm, by pre-calculating the weights one time and then storing them to be grabbed based on a given address, instead of looping through the CSV file every time.

**K.  Justify the data structure you identified in part D by doing the following:**

1. **Verify that the data structure used in the solution meets all requirements in the scenario.**

**a.  Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.**

As the number of data items grow, the chance for collisions increases. To handle collisions, the

lookup is not as simple and affects the time complexity of the process. The more collisions the

slower the look-up function will take to retrieve the data. Otherwise, it is a consistent and time

efficient method without collision.

1. **Explain how the data structure space usage is affected by changes in the number of packages to be delivered.**

The hash map should be equal to or greater than the size of the data set for the least chance of collision. So, as the data set grows, so too should the size of the hash table. The more data, the more collisions that are possible, which also increases the size of the lists within a certain key.

1. **Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.**

Both lookup time and space would expand based on a larger set of trucks and cities. Although in my own code, trucks are held separately from the hash table, but in other implementations, it could prove to be a burden on lookup time and space. The number of cities would certainly increase the space complexity because the data set would grow and the number of collisions would also increase along with it, which would also slow it down.

**2.  Identify two other data structures that could meet the same requirements in the scenario.**

1. **Describe how each data structure identified in part K2 is different from the data structure used in the solution.**
2. A binary search tree could be used. It differs because it progressively splits the data in two, based on whether a half is less than or greater than the key, and it progressively splits until it finds the solution.
3. Another data structure that could be used is a stack. Given the limited interaction available to a stack, it is less than optimal compared to a hash table to quickly look up items. An items position in the stack is also of the utmost importance, which differs from a hash table, where an item can be found through a single key, given that there are no collisions in that key.