**WGUPS**

**Stated Problem:**

The Western Governors University Parcel Service (WGUPS) needs to determine the best route and delivery distribution for their Daily Local Deliveries (DLD) because packages are not currently being consistently delivered by their promised deadline. The Salt Lake City DLD route has three trucks, two drivers, and an average of 40 packages to deliver each day; each package has specific criteria and delivery requirements.

Your task is to determine the best algorithm, write code, and present a solution where all 40 packages, listed in the attached “WGUPS Package File,” will be delivered on time with the least number of miles added to the combined mileage total of all trucks. The specific delivery locations are shown on the attached “Salt Lake City Downtown Map” and distances to each location are given in the attached “WGUPS Distance Table.”

While you work on this assessment, take into consideration the specific delivery time expected for each package and the possibility that the delivery requirements—including the expected delivery time—can be changed by management at any time and at any point along the chosen route. In addition, you should keep in mind that the supervisor should be able to see, at assigned points, the progress of each truck and its packages by any of the variables listed in the “WGUPS Package File,” including what has been delivered and what time the delivery occurred.

The intent is to use this solution (program) for this specific location and to use the same program in many cities in each state where WGU has a presence. As such, you will need to include detailed comments, following the industry-standard Python style guide, to make your code easy to read and to justify the decisions you made while writing your program.

***Assumptions:***

•  Each truck can carry a maximum of 16 packages.

•  Trucks travel at an average speed of 18 miles per hour.

•  Trucks have a “infinite amount of gas” with no need to stop.

•  Each driver stays with the same truck as long as that truck is in service.

•  Drivers leave the hub at 8:00 a.m., with the truck loaded, and can return to the hub for packages if needed. The day ends when all 40 packages have been delivered.

•  Delivery time is instantaneous, i.e., no time passes while at a delivery (that time is factored into the average speed of the trucks).

•  There is up to one special note for each package.

•  The wrong delivery address for package #9, Third District Juvenile Court, will be corrected at 10:20 a.m. The correct address is 410 S State St., Salt Lake City, UT 84111.

•  The package ID is unique; there are no collisions.

•  No further assumptions exist or are allowed.

This program is designed using an object-oriented design. All data transfer and processing is done via object reference. Python is inherently modular (Python 2020) and the problem is, at its core, about the interaction between physical objects which made modular OOP a great fit for the problem.

**Algorithm Overview:**

The problem is to be solved as following:

1. Packages are loaded onto the trucks according to the specifications defined in the package file.
2. Packages sharing addresses with these existing packages are loaded onto the corresponding trucks.
3. Using Christofides algorithm, a eulerian circuit is found in the remaining packages’ nodes.
4. For all trucks, a eulerian circuit is found for the nodes to be traveled by each.

for truck in trucks: # O(n^m) where n is the number of trucks and m is the number of packages

truck.packages = hard-coded

truck.packages += packages sharing address information with existing, hard-coded packages

for package in remaining packages: # O(n)

remainingNodes.append(package.address)

# O(n^2 \* log(n))

bestPath = remainingNodes.tsp # get a eulerian circuit from all remaining nodes

# O(n^2 \* n(log(n))) due to the Christofides algorithm being called

for truck in trucks:

while there is space in the truck:

add as many stops from the remaining nodes as possible

calculate eulerian circuit for all priority nodes and the rest as separate circuits

determine which node in the non-priority nodes is closest to the last node of the priority ones.

append the non-priority nodes to the priority nodes, starting with the closest determined in the next step

As there are multiple possible eulerian circuits possible in a complete graph, several iterations of package assignment may be necessary to find an optimal path which accommodates all delivery times. The best path is tracked and returned.

The current program requires hard-coding of packages with unusual requirements but could be modified to accept data either from an input file (.csv, json, etc). In a proper business-facing environment a GUI would be implemented alongside a database structure and possibly networking/communication features to adapt to drivers’ field conditions.

Maintenance would be fairly simple as the algorithm is mostly self-adjusting. Different heuristics could be chosen and applied without major renovation being done to the codebase. The objects which do the majority of the data handling are straightforward, easy to use and provide room for expansion.

If the program were to be scaled up, the issue of using hard-coded numbers for problematic packages would be solved via the use of a user interface. The traveling salesman problem , which is the crux of the program, would be solvable in the same manner. Locations can be added with the addVertix method of the graph class provided and packages could be added similarly. Scaling the program would be possible due to the efficiency of Christofides algorithm running at the program’s core.

This program is self-adjusting. Packages are assigned and delivered according to their distance and delivery time requirements. Packages without priorities are automatically shifted to the back of the queue. The use of list manipulation in Python allows for further adjustment. As the shortest path calculations returned by Christofides algorithm are complete circuits, they can be entered from any point and yield the same length. This allows the program to find the closest node to the current location of the truck and enter the circuit at that point.

Running time for Christofides algorithm, which is the main driver of this program is O(n^2 \* log(n)).

Christofides algorithm uses several steps to find a path within 3/2 of the optimal length of a traveling salesman problem (Neos-guide.org). These steps can be summarized as following (Ming-Yang Kao 517):

1. A minimum spanning tree is found by finding subtrees which are not cyclical until all nodes have been visited
2. A subgraph of all the nodes with an odd number of edges is found
3. A perfect matching with the minimum weight is found in this subgraph
4. This perfect matching is union joined with the subgraph from step 2.
5. From this joined graph, a eulerian tour is derived.
6. By removing any repeated vertices, a Hamiltonian path has been derived.

My program takes advantage of the cyclical nature of this algorithm’s output. The given path’s return value can be rotated so that any node in the path can be the starting point if necessary. This advantage, combined with the efficient and nearly ideal yield of the algorithm make it a great choice. Other options include shortest-path algorithms such as Dijsktra’s and A\* as well as other graph-oriented solutions or branch-and-bound. A\* and Dijsktra’s yield a shortest path from point-to-point while Christofides’ algorithm’s goal is to return the shortest path which visits all nodes.

The main data structure in my program is the graph, consisting of vertices and weighted edges. Also available for use would have been a tree or a distance matrix. Inserting a node into the graph is an O(n + 1) operation, where n is the amount of edges connected to the vertex. My program also uses adjacency matrices to represent the distances between nodes in some cases. Creating the matrix takes O(n^2) time to complete. The program uses both the graph’s interface and the adjacency matrix. The graph object includes, for each node, a list of all adjacent nodes and the weight associated with each. Either is readily accessible for processing, although the distance matrix is typically seen being used for quick lookups or to find the closest node to another using Python’s min() function (Python 2020).

It would have been possible to use a binary tree to represent the package addresses. However, the tree would become excessively large, having a space complexity of 2^n. Also possible would have been a plain adjacency matrix. Inserting and removing points would have been time-expensive as each time an address is removed or added, the entire matrix needs to be recalculated (Roman 6.4).

The highest order of complexity in this program is O(n^2). The use of graph operations and the delivery time/distance heuristic allows for relatively fast performance. As the requirements for the program are scaled up, it is possible that a new heuristic or process would need to be added. However, without consulting with the client or owner it is impossible to know the exact nature of any problems that may arise.

If I were to do this project again, I would have focused more on creating easier-to-work-with data structures and possibly spent more time on the whiteboard working out issues that I would have run into so that I didn’t have to struggle at the keyboard. I also did not delve into recursive solutions for this problem which it is perfectly set up for.

Works Cited

Works Cited

“3.7.3 Documentation.” *Python.Org*, 2019, docs.python.org/3/.

“Christofides Algorithm.” *Wikipedia*, 28 Feb. 2020, en.wikipedia.org/wiki/Christofides\_algorithm. Accessed 7 Apr. 2020.

Lysecky, Roman, and Frank Vahid. *Data Structures and Algorithms II*. 2018, learn.zybooks.com/zybook/WGUC950AY20182019.

Ming-Yang Kao. *Encyclopedia of Algorithms*. Berlin, Springer, 2008, pp. 517–519.

“Multiple Traveling Salesman Problem (MTSP) | NEOS.” *Neos-Guide.Org*, neos-guide.org/content/multiple-traveling-salesman-problem-mtsp. Accessed 7 Apr. 2020.

Zhuravchak, Andrew. “Retsediv/ChristofidesAlgorithm.” *GitHub*, 29 Mar. 2020, github.com/Retsediv/ChristofidesAlgorithm. Accessed 7 Apr. 2020.

‌

‌