WGUPS Algorithmic Overview

Stated Problem:

The purpose of this project is to create a solution that provides the user with an optimized system for handling postage delivery. To do so, the use of a self-adjusting heuristic greedy algorithm[1] programmed in Python is deployed to organize the truck delivery schedule for all packages based firstly on destination proximity, and secondly on truck distribution. To store the subsequent information established by the core algorithm, a linear hash table that adjusts its capacity to the number of packages provided. This table will hold ordered "package" objects that hold the necessary package data of weight, delivery address, deadlines, and special instructions.

By being able to store the delivery data of each individual package, the subsequent grouping and shipping of items can be readily scheduled for the same truck route. Through the use of the hash table, information can be readily accessed and operated on to curate efficient delivery scheduling results.

Algorithm Overview:

The program will read the distance data from the *WGUPS Distance Table* CSV file and create arrays of data for each column from left to right sequentially. The *WGUPS Package File* CSV is used to retrieve package data that is then paired via location matching to routes. The core algorithm parses through the data to create a delivery schedule in the following manner:

1. Searches for the smallest distance in the column.
2. Uses row data to add location to route.
3. Finds packages that match that destination address.
4. Checks for any special delivery conditions. (if found, see 5.a.)
    a. Reconfigures routing based on condition codes to favor certain distributions
5. Appends package(s) to the truck route.
6. Marks location as visited.
7. Cycles to the next column array (and route array if at package capacity).

To develop this software, the environment of PyCharm will be utilized with the code being written in Python 3.10. And, while being developed locally on a standard Windows OS running laptop with an Intel core i5 9th gen processor, this project will be transferable to other systems via a Docker application that can be made readily ported if the need arises.

This WGUPS solution has 3 distinct segments that each contribute to the overall operation time of the software. First, the extraction of data from CSV files takes O(n) to complete, as each item needs to be visited to be implemented into the program. Location distance data is kept in distinct arrays that preserve the column and row information, and package data is sorted then into chained hash tables based on their delivery location (expressed by row numbers established by the WGUPS Distance Table). These are sorted directly from the base CSV documents, but with the ordering of packages based on the distance table location order, the time complexity rises to O(n+k). This is a necessary increase that the software must have, as distances and packages need an intermediary value to be paired, which is delivery location. Next, the core algorithm is deployed to create a delivery schedule for the packages. Routes are propagated by choosing the smallest value in a given column, thus the time complexity of this is O(n), though the program is still O(n+k). Lastly, there are the special delivery conditions that may affect package-to-route sorting. This segment uses condition codes that organize routing for a given item or a set of items. These codes are assessed at the beginning of the solution to mitigate interrupts of routing. As each package is checked once for this, the complexity of this segment is O(n) while the program overall has O(n+k) complexity.

The space complexity of this program will be O(n+k), with n and k being the data from the WGUPS Distance Table and the WGUPS Package File respectively. This is the case because to perform operations on the n data, the proposed program must sift through it. And for k, each package must be accounted for and saved. Thus the space complexity is proportional to the combined data in the 2 CSV files.

Pseudocode:

Data Extraction and Sorting

```
Open  "Distance Table CSV"
Open "Package File CSV"
For Addresses in column 1:
    add address to addressArray      //allows row to address pairing
For Columns:
    For Row:
        if distance < smallestDist:
            smallestDist = distance
            location = addressArray[row]
    packageToRouteSort(smallestDistance, location)     // adds package(s) to route
    locationVisitedArray[row] = true
```

[The distance and package files are opened and sifted through to find the shortest distances between 2 package locations. This information is then continued on to the routing decider for packages referenced by the above function “packageToRouteSort()”.]

Package Routing

```
packageToRouteSort(smallestDistance, location) :
    For Packages in *Package File CSV:*
        if deliveryAddress = location:
            if route >= 16 packages:
                route = next route
            if package has instructions:
                do specialInstructions()
            else:
                add (package, distance) to route
```

[This segment locates the packages that match the location address to the applicable packages. There is then a check to see if there are any special instructions that need to be adhered to, or otherwise the package and distance information is added to the current route.]

Special Instructions

```
specialInstructions() :
    check package for instruction codes      // a.e. “pairing”, “priority”, “delay”, “truck bound”
        for paired packages:
            if paired package has instructions:
                do specialInstructions()
            while route packages < 16:
                add paired packages
            if route packages >= 16:
                route = new route
        for priority packages:
            swap package with a route 1 or route 2 package
```

```
for delayed packages:
    add package to next route
for truck bound packages:
    if route number % bound truck ID:
        add package to route
    else:
        altRoute = route where route number % = 0
        add package to altRoute
```

[This segment reads package instructions and performs appropriate action tailored to each type. This assists the overall program by aligning any nuances that need to be handled outside of the scope of the initial package routing method.]

Takeaways:

With the simple and efficient solution outlined above, the user of the new WGUPS routing system can manage package deliveries and mitigate the current problem of delayed drop-offs.

This provided solution is scalable and adaptable for ongoing business needs. The current architecture readily supports increases or changes of company resources including delivery addresses, trucks, and cargo limits. The data structures that hold such information within the program are also scalable and adaptable by means that they are all based on self-adjusting models.

This hash table utilized for scalable data storage impacts the overall performance by providing a central, accessible point for information to be stored and shared. But, with such a structure there comes a weakness of resource usage if there is an increasing amount of data trying to be stored by the program in such a way. The resources of the operating machine might not be equipped to handle the types of RAM memory that an adjusting hash table can grow to need. This outcome, however, is unlikely for the current usage of this program, as the data being processed is not intensive to modern computer designs. If, however, WGUPS would like to expand to cover also other cities or states, then resource management techniques would have to be put in place to have this solution work as intended at that scale.

The code is both efficient and easily maintainable by merit of its simplistic design. The process reads all information aptly from the source documents, sorts them directly from there according to program needs, and is then able to calculate possible viable routes from the data. With this linear complexity, the time and resource costs are left at a minimal while also keeping distinct transformation segments intact in case an error needs to be addressed. Also, if adding new dimensions like weight limits or more integrated special instruction codes this solution can easily be amended with new modules for the respective changes and implement them in adjacent styles to the already included factors.

Lastly, in regards to the abilities of the route generating algorithm aforementioned, the configuration specified provides the user with a much more efficient and reliable system for planning on-time package delivery throughout the service area. A key nuance that assists in this effort is the choice of using locations to connect packages with routes while minimizing distances traveled. By using these addresses for the hash table key as well, this provides the solution with a singular translation between the many facets of the program that would otherwise have to be cross examined to meet the solutions required goals. Thus, using location mitigates both time and complexity constraints due to its staple as an intermediary data set.

Sources

1

Lysecky, R., & Vahid, F. (2018, June). C950: *Data Structures and Algorithms II*. zyBooks. Retrieved May 30, 2025, from https://learn.zybooks.com/zybook/WGUC950AY20182019