Core Algorithm Overview

Stated Problem:

The purpose of this project is to create an algorithm using Python for Western Governors University Parcel Service (WGUPS) to determine an efficient route and delivery distribution for their Daily Local Deliveries (DLD) in Salt Lake City. The route has three trucks, two drivers, and an average of 40 packages to deliver each day. Each package has specific criteria and delivery requirements.

A: ALGORITHM SELECTION

The WGUPS Routing Program utilizes a greedy algorithm to deliver packages. Two main purposes of this self-adjusting algorithm are to load packages into respective trucks and optimize each truck's route. Refer to Section B1 for pseudocode.

B1: LOGIC COMMENTS

The greedy algorithm described in Section A is implemented in `optimize.py` for package loading and route optimization. The following pseudocode represents the algorithm (along with package delivery simulation implemented in `simulation.py`):

PACKAGE LOADING [`optimize.py`]

```
    Load Truck 1 (Time-sensitive) with time-sensitive packages
    Load Truck 2 (Required) with required and delayed packages
    Load Truck 3 with remaining EOD packages
        If (package is on the route of truck 1 or 2) and (space is available):
            Load package into corresponding truck
        Else:
            If truck 3 has available space:
                Load package into truck 3
            Else:
                Load into truck 2
```

ROUTE OPTIMIZATION [`optimize.py`]

```
     For each truck:
        Initialize route using nearest neighbor algorithm
        Optimize route by 2-opt algorithm to avoid crossing
```

PACKAGE DELIVERY [`simulation.py`]

```
    Truck 1 delivery starts at 8:00 AM
    Truck 2 delivery starts at 9:05 AM
    Truck 3 delivery starts when truck 1 or 2 arrives at hub
    For package(s) with wrong address:
        Wait until 10:20am to update correct address
```

B2: DEVELOPMENT ENVIRONMENT

The WGUPS Routing Program is a Python program written in software and hardware specifications below.

Hardware	Software
Operating System: Windows 10 Processor: AMD Ryzen 5 2600 3.40 Ghz Memory: 16 GB 1200 MHz DDR4	Programming Language: Python v3.9.2 IDE: PyCharm Community 2020.3.3

B3: SPACE-TIME AND BIG-O

The core algorithm for package loading and route optimization described in Section B1 (implemented in `optimized.py`) has the space-time complexity in the table shown below:

Section	Space Complexity	Time Complexity
Package Loading	O(N)	O(N)
Route Optimization	O(N)	$O(N^2)$

Other major blocks of code in the program have space-time complexity as shown below.

`HashTable.py`

Method	Space Complexity	Time Complexity
__init__	O(1)	O(1)
insert	O(N)	O(N)
get	O(N)	O(N)
remove	O(N)	O(N)

`functions.py`

Method	Space Complexity	Time Complexity
get_package	O(N)	O(N)
assigned_truck	O(1)	O(1)
load	O(1)	O(1)
diff_distance	O(1)	O(1)
optimized_route	O(N)	$O(N^2)$
blockPrint	O(1)	O(1)
enablePrint	O(1)	O(1)
print_truck_header	O(1)	O(1)
print_package_header	O(1)	O(1)

`simulation.py`

Method	Space Complexity	Time Complexity
run	O(N)	$O(N^2)$
truck_mileage	O(1)	O(1)
package_status	O(N)	O(N)

∴ The entire program has **space complexity of O(N)** and **time complexity of $O(N^2)$.**

B4: ADAPTABILITY

The self-adjusting greedy algorithm and package/location hash tables makes scaling simple. The algorithm has O(N) space complexity and $O(N^2)$ time complexity. The hash tables allow insertion and lookup time complexity of O(N) at worst and O(1) on average. Thus, the application can adapt to an increasing number of packages, as well as number of locations.

However, with the increasing number of packages, the number of different restrictions/constraints imposed on the packages might also escalate. This scenario will possibly require an additional code to parse the special package requirements.

B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY

Efficiency:

This finding a delivery route problem is known as the "Traveling Salesperson Problem" (or TSP) which belongs to NP complexity class. The entire WGUPS program has $O(N^2)$ time complexity, which might not be optimal. However, the selection of the greedy algorithm makes the program efficient and easier for maintenance.

Maintainability:

With the possibility of business expansion, the program has been compartmentalized logically into manageable packages. The program is well-documented with docstrings and comments sparingly to assist troubleshooting and understanding the code, making maintenance and improvement effortless.

Package	Description
`main.py`	Main program running with user-friendly interface
`HashTable.py`	Hash Table class
`Classes.py`	Define classes (Package, Location, Format, Status)
`readData.py`	Read .csv input for locations, packages, and distances
`init.py`	Initialize assumptions
`functions.py`	Define commonly used functions
`optimize.py`	Implement greedy algorithm to load packages and optimize routes
`simulation.py`	Simulate package delivery

B6: SELF-ADJUSTING DATA STRUCTURES

The hash tables implemented for packages and locations are an example of self-adjusting data structure.

Strengths:

The main advantage of hash tables is fast lookup, which takes O(1) if items are uniformly distributed in buckets, avoiding collisions. The other advantage of using hash tables is flexible data types can be used for keys to lookup. This is demonstrated in a location lookup with a known address on a package.

Weaknesses:

Hash tables can experience collisions, which occur when two or more items map to the same bucket. To address this problem, a collision resolution technique called chaining is utilized, where each bucket has a list of items. Another weakness of hash table is single-directional lookup, i.e., fast lookup is only possible when look up the value of given key, while searching for the keys for a given value requires looping through the whole dataset with O(n) time.

C: ORIGINAL CODE

Refer to the code submission.

C1: IDENTIFICATION INFORMATION

Refer to the code submission (`main.py`, line 1)

C2: PROCESS AND FLOW COMMENTS

Refer to the code submission.

D: DATA STRUCTURE

The self-adjusting package hash table is used to store the package information and perform well with the greedy algorithm identified in Section A.

A package contains:

- package ID number
- delivery address
- delivery deadline
- delivery city
- delivery zip code
- package weight
- delivery status

The `insert()` function inputs all information of a package to the hash table.

The `get()` function returns a package with corresponding information given a package ID.

D1: EXPLANATION OF DATA STRUCTURE

A simple linear search is O(N) time complexity. With hash table, retrieving information could take O(1) with a good hash function. Hashing is mapping an item's key to an index with a hash function. In the package hash table, the key is package unique ID. A hashed key is an index for the bucket assigned for the item. This eliminates the need for searching the entire list and makes retrieving information accurate and more efficient than a simple linear search.

E: HASH TABLE

Refer to Section D.

F: LOOK-UP FUNCTION

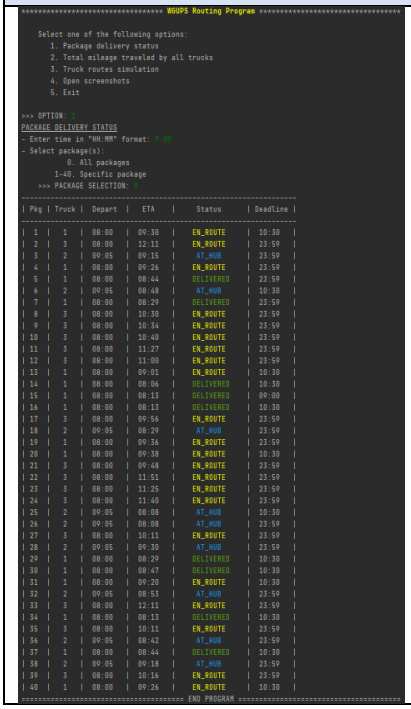
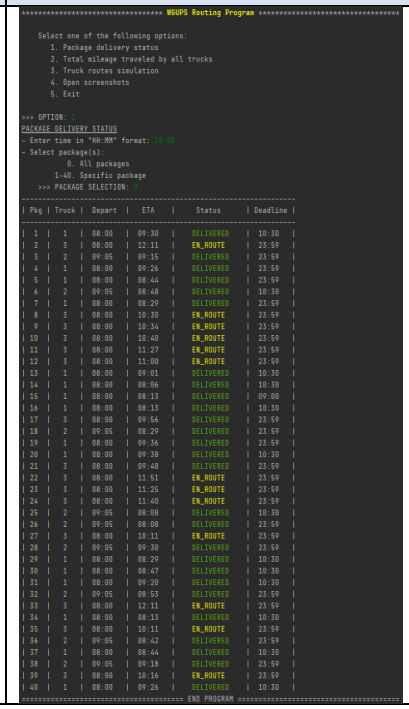
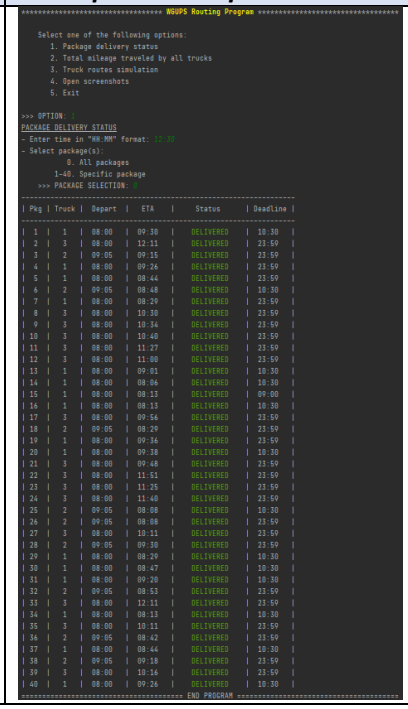
Refer to Section D.

G: INTERFACE

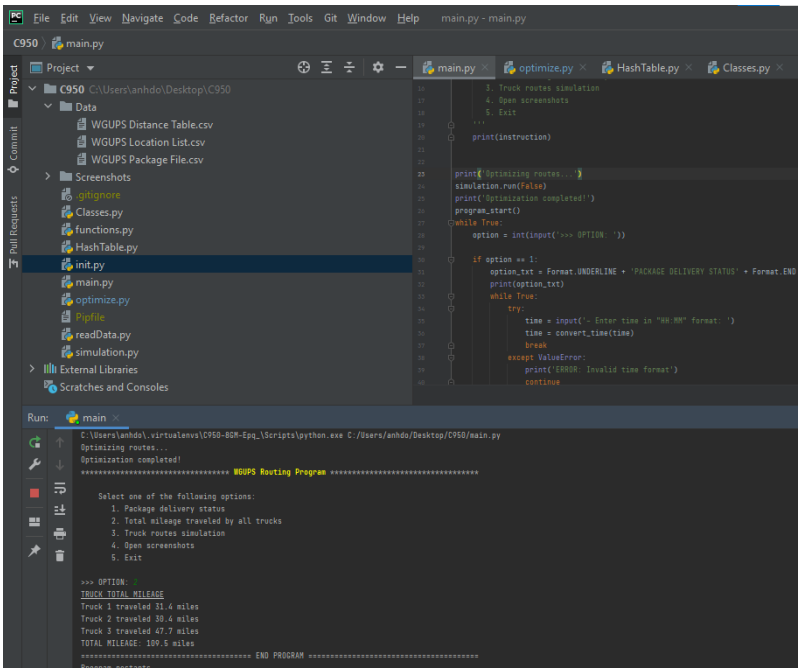
Refer to the code submission or result screenshots.

G1-G3: 1st, 2nd, and 3rd status checks.

Refer to the code submission or result screenshots (attached here for convenience).

8:35 a.m. - 9:25 a.m	***9:35 a.m. - 10:25 a.m***	***12:03 p.m. - 1:12 p.m***
		

H: SCREENSHOTS OF CODE EXECUTION



I1: STRENGTHS OF THE CHOSEN ALGORITHM

The greedy algorithm is selected due to its ease to implement for finding close to optimization results. Another advantage of greedy algorithm is that it is significantly more efficient compared to brute force algorithm for TSP problem.

I2: VERIFICATION OF ALGORITHM

The solution provided by the algorithm meets all the requirements.

- The total miles traveled by all trucks is less than 110 miles
- All packages were delivered on time and according to their delivery specifications

Refer to Section G for verification.

I3: OTHER POSSIBLE ALGORITHMS

Other possible algorithms can also be applied. Two algorithms I found interesting are Simulated Annealing and Ant Colony Optimization, both of which utilize randomization.

I3A: ALGORITHM DIFFERENCES

Simulated Annealing comes from annealing process in metallurgy, involving heating and controlled cooling of a material to alter its physical properties. Initially at high "temperature" during the optimization process, it has high energy (wide randomization range) to exit local optima. One of the disadvantages is that it requires carefully controlling the cooling rate, which largely affects the runtime. Another minor disadvantage is inconsistency in results given the same input due to its random nature.

Ant Colony Optimization is based on real ants' behavior to find short paths between food sources and their nest, using trail pheromones deposited by other ants. One advantage of this algorithm is that pheromones can be evaporated, which avoids being trapped in local optima. Even though convergence is guaranteed, time to convergence is uncertain.

J: DIFFERENT APPROACH

The current program handles package notes in a somewhat static way. This can be improved by implementing a more generalized system to handle variety of restrictions, which is beneficial when business expands, facing different needs.

K1: VERIFICATION OF DATA STRUCTURE

The solution provided by the data structure meets all the requirements.

- The total miles traveled by all trucks is less than 110 miles
- All packages were delivered on time and according to their delivery specifications
- Package hash table was utilized with a look-up function `get()`
- Package status and information can be verified through the user interface "report"

K1A: EFFICIENCY

The hash table provides an efficient solution to data retrieval. With O(1) time complexity on average, adding packages should not have significant effect on lookup function runtime.

K1B: OVERHEAD

The hash table provides an efficient solution to data storage. With O(N) space complexity, space usage increases linearly as the number of packages grow.

K1C: IMPLICATIONS

As the number of trucks can be assumed to be significantly less than the number of packages, additional trucks should not have significant impact on look-up time and space usage.

A hash table is also implemented on cities/locations. Thus, additional cities will have minimal affect on look-up time and space usage.

K2: OTHER DATA STRUCTURES

The alternative data structure can represent packages in destination locations' hash table, i.e., each location holds a list of packages addressed to that location.

Another alternative data structure is packages represented in a matrix, whereas a[i,j] determines the delivery distance from package i's destination to package j's destination.

K2A: DATA STRUCTURES DIFFERENCES

The use of destination locations' hash table might make the route more optimized since each location only need to be visited once to deliver all packages destined at that location. However, complication arises with changing address, requiring moving a package to another destination location. Or, if we need to visit a location again on different truck to satisfy a specific requirement.

The use of package matrix can be useful when applying nearest-neighbor algorithm. However, same as the other alternative, complication arises when changing address, requiring updating the matrix. Unlike the destination locations' hash table, package matrix is more flexible in terms of package loading.

L: SOURCES

Abreu, N., Ajmal, M., Kokkinogenis, Z., & Bozorg, B. (2011, 01 17). Retrieved from Universidade do Porto: https://paginas.fe.up.pt/~mac/ensino/docs/DS20102011/Presentations/PopulationalMetaheuristics/ACO_Nuno_Muhammad_Zafeiris_Behdad.pdf
Busetti, F. (n.d.). Retrieved from AI in Finance: http://www.aiinfinance.com/saweb.pdf

Lysecky, R., & Vahid, F. (2018). *C950: Data Structures and Algorithms II.* Zyante Inc. (zyBooks.com).

M: PROFESSIONAL COMMUNICATION

Refer to this document and code submission.