NHP3 — NHP3 TASK 2: WGUPS ROUTING PROGRAM IMPLEMENTATION

**SALAHADEAN ARJEH**

**July 22nd, 2024**

### 

## **A: Algorithm Selection**

This program uses the 2-opt search algorithm to find the shortest path between addresses. The 2-opt algorithm optimizes the route to avoid self-crossing paths, potentially finding an optimal route (Croes, 1958). The 2-opt algorithm operates in O(n2)O(n^2)O(n2) time, similar to the nearest neighbor (NN) algorithm, but it appears to be a more elegant solution to this problem. The NN algorithm typically does not produce the most optimal tour (Gutin et al, 2002). My objective was to maintain a runtime around O(n2)O(n^2)O(n2); otherwise, I could have opted for the 3-opt algorithm, which operates in O(n3)O(n^3)O(n3).

## **B1: Logic Comments**

The following explanation describes the process of how my program solves this problem:

**Load/store package data given to us in an excel/csv format**

Read the csv file to parse each data row.

Take those parsed rows and create a package object.

Store those package objects into a hash table.

**Load/store distance data given to us in an excel/csv format**

Read the csv file to parse each data row.

Clean the parsed data so that it is usable.

Store the data so that it is easily iterable/retrievable

## **C.  Write an original program that will deliver all packages and meet all requirements using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and “WGUPS Package File.”**

To run the main program that delivers the packages, the deliver\_packages() function is invoked. This function creates three truck objects. Using these truck objects and the package data, the program sorts the packages into trucks 1, 2, and 3. During the sorting process, each truck gets a list of addresses that need to be visited. Once the trucks are loaded with their respective packages and addresses, the address lists are optimized to determine the best route. The optimization is achieved by passing each address list into the optimize() function. This function performs a 2-opt swap on the addresses, iteratively checking if the new version of the route is shorter than the previous one. If it is, the new version becomes the optimal route. The 2-opt swap process involves swapping every address with every other address until the optimal route is found. Each time a route undergoes a 2-opt swap, it is evaluated using the cost() function, which calculates the total distance required to visit each address in the given list, starting from index 0.

**1.  Create an identifying comment within the first line of a file named “main.py” that includes your student ID.**

**2.  Include comments in your code to explain both the process and the flow of the program.**

**Command Line Interface (CLI)**

The rest of the program uses a CLI that allows end users to perform the following actions:

* Insert a package into the hash table
* Look up a package by ID
* Look up a package by ID and time
* See details of all packages

### **2-Opt Algorithm Overview**

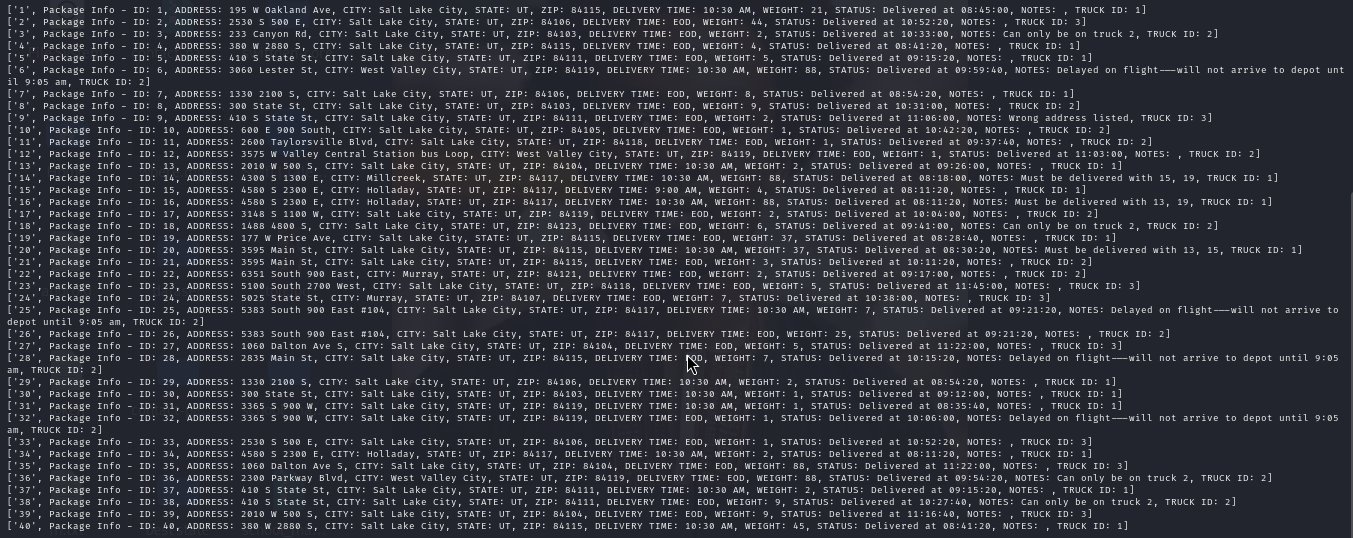
The 2-opt algorithm works as follows:

1. Initiate a random list of nodes and assign it as the optimal route.
2. From the second node to the second-last node (i.e., starting node + 1 to length of nodes - 1):
   * From the third node to the last node (i.e., starting node + 2 to length of nodes):
     + Create a new route by making a copy of the current optimal route, then swap nodes from index i to k.
     + For each new route, calculate the cost to travel from node to node starting at the first node.
     + If the cost of the new route is less than the current optimal route, assign the new route as the optimal route.
     + Repeat the process until the routes can no longer be optimized.

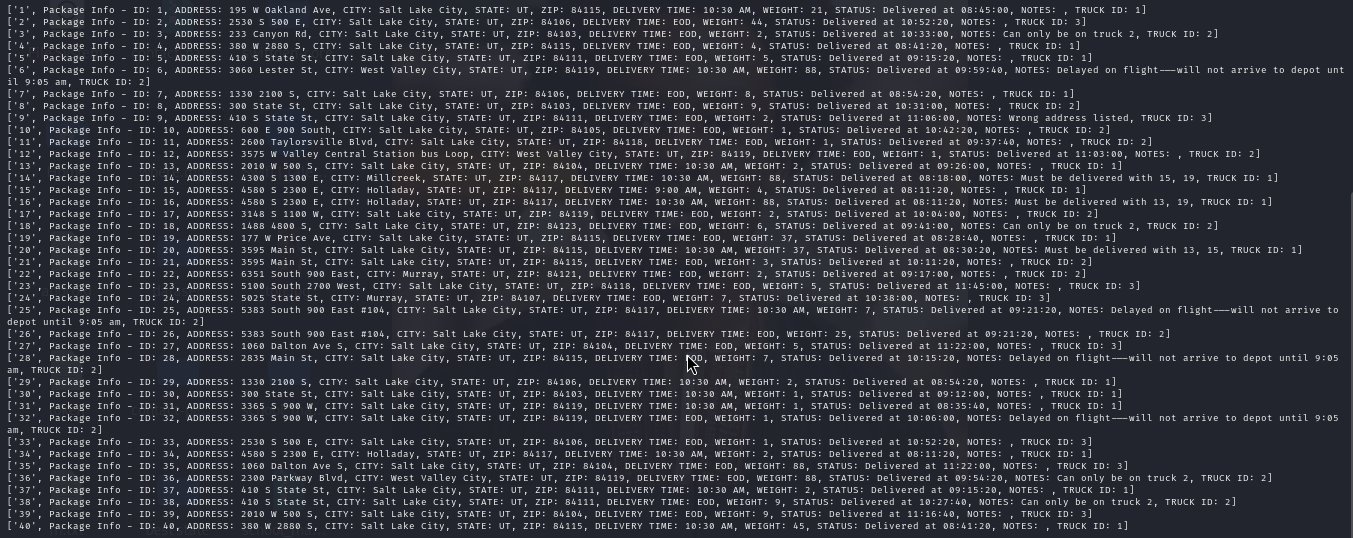
| Pseudocode for the 2-Opt Algorithm |
| --- |
| function two\_opt(route):  best = route  improved = True  while improved:  improved = False  for i = 1 to length(route) - 2:  for k = i + 1 to length(route) - 1:  new\_route = swap(route, i, k)  if cost(new\_route) < cost(best):  best = new\_route  improved = True  return best  function swap(route, i, k):  new\_route = route[0:i] + route[i:k+1][::-1] + route[k+1:]  return new\_route  function cost(route):  total\_cost = 0  for i = 0 to length(route) - 1:  total\_cost += distance(route[i], route[i+1])  return total\_cost |

## **D.  Provide an intuitive interface for the user to view the delivery status (including the delivery time) of any package at any time and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)**

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

****

#### **2.  Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 9:35 a.m. and 10:25 a.m.**

****

#### **3.  Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 12:03 p.m. and 1:12 p.m.**

None was delivered at this time

## **E.  Provide screenshots showing successful completion of the code that includes the total mileage traveled by *all* trucks.**

### 

## **F.  Justify the package delivery algorithm used in the solution as written in the original program by doing the following:**

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

* Efficient Route Optimization: The 2-opt algorithm is effective in optimizing routes by iteratively swapping nodes to reduce the total travel distance. This heuristic approach helps in significantly reducing the route length, making it more efficient for delivering packages within a given distance constraint (Croes et al…1958).
* Simple and Practical Implementation: The 2-opt algorithm is relatively simple to implement and understand. Its simplicity allows for quick development and deployment, which is beneficial for practical applications like package delivery. Despite its simplicity, it can provide substantial improvements in route optimization compared to unoptimized routes.

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

* Package Sorting into Trucks: The deliver\_packages() function successfully sorts packages into three trucks based on given data.
* Address List Creation: Each truck generates a list of addresses to be visited, ensuring that all required destinations are covered.
* Route Optimization: The 2-opt algorithm optimizes each truck's address list, ensuring the shortest possible route within the provided constraints.
* Distance Calculation: The cost() function accurately calculates the total travel distance for each route, ensuring that the optimal route is selected.

Thus, the algorithm meets the requirements of efficient package delivery, address list creation, and route optimization.

#### **Identify two other named algorithms that are different from the algorithm implemented in the solution and would meet *all* requirements in the scenario.**

a. Genetic Algorithm:

b. Simulated Annealing:

##### **a.  Describe how *both* algorithms identified in part F3 are different from the algorithm used in the solution.**

a. Genetic Algorithm:

* Description: A genetic algorithm (GA) uses concepts from natural selection to find an optimal solution. It starts with a population of possible routes, selects the fittest individuals, and combines them to create new routes. Mutations are also introduced to maintain diversity.(Scott Thede, 2004)
* Difference: Unlike the 2-opt algorithm, which iteratively swaps nodes to improve a single route, a genetic algorithm explores multiple routes simultaneously. This allows it to avoid local optima by maintaining a diverse population of solutions.

b. Simulated Annealing:

* Description: Simulated annealing (SA) is inspired by the annealing process in metallurgy. It starts with a random solution and explores neighboring solutions with a probability that decreases over time. The probability of accepting worse solutions decreases, allowing the algorithm to escape local optima (Darrall Henderson, 2006).
* Difference: Simulated annealing explores solutions by probabilistically accepting worse routes initially, allowing it to escape local optima more effectively than the 2-opt algorithm. It does not rely on deterministic swaps but rather on probabilistic transitions to explore the solution space.

## **G.  Describe what you would do differently, other than the two algorithms identified in part F3, if you did this project again, including details of the modifications that would be made.**

#### If I were to do this project again, I would consider the following modifications:

#### 1. Use of Heuristic-Based Initial Route Generation

* Modification: Instead of starting with a random initial route, I would use a heuristic such as the nearest neighbor algorithm to generate a more efficient initial route.
* Details: The nearest neighbor algorithm would construct an initial route by starting from a random node and repeatedly visiting the nearest unvisited node. This provides a better starting point for the 2-opt algorithm, potentially reducing the number of iterations required for optimization.

#### 2. Dynamic Time Windows for Delivery

* Modification: Implement dynamic time windows for each package delivery to ensure packages are delivered within specified time frames.
* Details: This would involve incorporating time constraints into the route optimization process. Each package would have a delivery deadline, and the optimization algorithm would need to account for these constraints when determining the best route. This could be achieved by extending the cost function to include penalties for late deliveries.

#### 3. Parallel Processing for Route Optimization

* Modification: Utilize parallel processing to optimize the routes for multiple trucks simultaneously.
* Details: By leveraging multithreading or multiprocessing, the optimization process could be sped up significantly. Each truck’s route optimization could be run in parallel, reducing the overall time required to determine the optimal routes for all trucks.

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

#### **Verification**

**Efficiency:** The data we used in the structure is a series of data points that make up a package object. Each package object is created and then stored in the hash table using the package ID as a unique key. Using this unique key the data can then be retrieved from the hash table in constant time. The unique key translates directly into the index that the package is located in the structure. This allows package data points to be accessed and updated as the program runs(M. M. Flood). **Overhead:** For handling the data we are able to achieve a constant time complexity since we have built a hash table. This means retrieval and storage of data happens in constant time. Since this program is run from a local machine the only memory or bandwidth concerns remain within

1. **Identify two other data structures that could meet the same requirements in the scenario.**
2. Balanced Binary Search Tree (BST): Description: A balanced BST such as an AVL tree or a Red-Black tree can store packages in a way that allows for efficient insertion, deletion, and lookup operations. Each node in the tree would represent a package, with package IDs as keys (Croes et al…1958).
3. Trie (Prefix Tree): Description: A trie is a tree-like data structure that stores keys as a series of characters. It can be used to store packages by ID, allowing for efficient lookups and insertions.

#### **a.  Describe how *each* data structure identified in H1 is different from the data structure used in the solution**.

1. Balanced Binary Search Tree (BST)

* **Difference**: A BST maintains a sorted order of keys, allowing for range queries and ordered traversals. Unlike hash tables, which offer average-case O(1)O(1)O(1) lookup, insert, and delete operations, a BST provides O(log⁡n)O(\log n)O(log n) operations, but ensures worst-case performance guarantees.
* **Advantages:** Supports in-order traversal, range queries, and maintains data in a sorted order, which can be beneficial for certain types of queries and operations.

1. Trie (Prefix Tree)

* **Difference:** A trie stores keys as paths from the root to the leaves, which is different from the flat structure of a hash table. Lookups and insertions in a trie are O(k)O(k)O(k), where kkk is the length of the key (Gutin et al…2002).
* **Advantages:** Efficiently handles prefix-based queries and can be more space-efficient when storing a large number of keys with shared prefixes. Ideal for scenarios where package IDs share common prefixes.

## **References**

* Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, *6*(6), 791-812.
* Gutin, G., Yeo, A., & Zverovich, A. (2002). Traveling salesmen should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics*, *117*(1-3), 81-86.
* M. M. Flood, The traveling-salesman problem. Operations Res. 4 (1956), pp., 61-75. <https://en.wikipedia.org/wiki/2-opt#cite_ref-2>
* Scott Thede, 2004. An introduction to genetic algorithms <https://www.researchgate.net/publication/228609251_An_introduction_to_genetic_algorithms>
* Darrall Henderson, Sheldon H Jacobson, Alan W. Johnson (2006) The Theory and Practice of Simulated Annealing <https://www.researchgate.net/publication/225260290_The_Theory_and_Practice_of_Simulated_Annealing>