C950 Task-1 WGUPS Algorithm Overview

(Task-1: The planning phase of the WGUPS Routing Program)

Trayvonious Pendleton

ID 011205284

WGU Email: tpend32@wgu.edu

9/5/2024

C950 Data Structures and Algorithms II

# Introduction

For this WGUPS Routing Program, I recommend using a nearest neighbor greedy algorithm with a self-adjusting data structure as a hash table. The closest neighbor greedy algorithm can adjust its route by picking the following closet location, which ensures a traversal of delivery addresses. A hash table can store and manage package information, such as addresses, deadlines, and delivery status.

# A. Algorithm Identification

The nearest neighbor greedy algorithm is a self-adjusting algorithm that chooses the closest unvisited delivery point at each step, ensuring trucks follow the most efficient route without starting from scratch. This is important because it allows the trucks to adapt to changes, such as updated package addresses, without losing performance.

# B. Data Structure Identification

The hash table will store package details, like Package ID, address, and deadline. The average time complexity is O (1) for lookups, which makes it a good match for dynamic data access for determining the delivery point.

# B1. Explanation of Data Structure

The hash table stores each package's data using a package ID as the key; each entry can hold multiple information like address, constraints, and status. The hash table makes the data structure easy to update, retrieve, and change package information.

# C1. Algorithm’s Logic

The algorithm uses the nearest neighbor algorithm to find the most efficient route for each truck to deliver packages. This algorithm always selects the nearest package to the current location. It considers package delivery deadlines and constraints, such as the incorrect address for package #9, which is only corrected after 10:20 am.

1. Load all package data into a hash table.

2. Initialize three trucks.

3. Assign the first batch of 16 packages to each truck based on proximity to the hub.

4. For each truck:

a. Use the nearest neighbor algorithm to calculate the most efficient route.

b. Deliver each package, adjusting for deadlines and time constraints.

c. If the truck needs more packages, then the truck will return to the hub and repeat.

5. Ensure all packages are delivered before their deadline and total mileage remains below 140 miles.es here.

This approach ensures efficiency by minimizing the distance traveled at each step while considering time-sensitive deliveries. The has table allows quick lookups of package details, and the nearest neighbor algorithm ensures that the trucks cover minimal distance, reducing overall mileage.

1. Loading Package Data:
   * Insert package data into a hash table for O(1).

**for each package in package\_data:**

**hash\_table.insert(package\_id, package)**

1. Initialize Trucks:
   * Create 3 trucks, each with an initial location set to the hub, and assign the first 16 packages to each truck.

**trucks = initialize\_trucks(3)**

**for each truck in trucks:**

**assign\_first\_batch\_of\_packages(truck, 16, hash\_table)**

1. Nearest Neighbor Routing:
   * For each truck use the nearest neighbor algorithm to calculate the next package to deliver based on proximity to the current location.
   * Update the truck current location,total distance, and time

**for each truck in trucks:**

**while packages remain:**

**nearest\_package, distance = find\_nearest\_package(truck.current\_location, truck.packages)**

**if deadline or time constraints:**

**adjust routing**

**truck.deliver(nearest\_package)**

**remove package from truck.packages**

**if truck needs more packages:**

**return to hub and load more**

1. Ensure Delivery Before Deadline:
   * Monitor package delivery deadline and adjust routing accordingly

**if current\_time < 10:20 and nearest\_package == package\_9:**

**skip package\_9 until correct time**

1. Total Mileage Calculation:
   * Track the total distance traveled by each truck, ensuring that the combined mileage stays under 140 miles.

**for each truck:**

**total\_distance += truck.total\_distance**

**if total\_distance > 140:**

**return error**

The algorithm uses a greedy approach to find the most efficient delivery routes. By consistently delivering the closest package, the trucks minimize their travel distance. The hash table allows fast access to package data, improving efficiency during routing. The algorithm dynamically adjusts for time-sensitive packages, ensuring that constraints like deadlines and address corrections are respected.

The nearest neighbor algorithm works well for this scenario because it simplifies route planning, which is essential when delivering a relatively small number of packages

# C2. Development Environment

The program will be developed using Python in a Visual Studio Code Ide environment. I'll use macOS to work on a machine with 16GB of RAM and an m1 processor chip. The version of Python that would be used would be 3.11.3.

# C3. Space and Time complexity using Big-O notation

The time complexity of the nearest neighbor algorithm runs in O(n^2), where n is the number of delivery locations. The hash table lookup for package information runs in O (1), and for storing packages, the space complexity is O(n). The overall complexity is dominated by the nearest neighbor algorithm, O(n^2), assuming the number of trucks is constant. The overall space complexity is O(n) since no part of the program significantly increases the space complexity beyond storing the packages and routes.

# C4. Scalability and Adaptability

Scaling the application for more packages would involve adding more trucks or increasing delivery batches. The nearest neighbor algorithm will adjust its routes dynamically, making it adaptable to more numbers of delivery points.

# C5. Software Efficiency and Maintainability

The design is efficient due to using polynomial-time algorithms and hash tables for fast lookups. Its maintainability is enhanced by the clear separation of logic for routing, data storage, and package management, making it easy for other developers to improve and maintain.

# C6. Self-Adjusting Data Structures

The strengths of the hash table are fast lookups and updates, which makes it ideal for dynamically managing package information. The weakness of hash tables is the potential for collisions, which can lead to slower lookups if too many packages are hashed in the same bucket.

# C7. Data Key

The package ID is the best key for managing deliveries because it uniquely identifies each package. It allows quick lookups and updates to track package status and ensure each delivery is appropriately accounted for.