**Package Routing Optimization Utilizing Nearest-Neighbor Algorithm**

Duncan Gwin

ID #008698673

C950 – Data Structures & Algorithms II

January 23, 2023

**PROBLEM STATEMENT**

The Western Governors University Parcel Service (WGUPS) needs to determine an efficient 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 an algorithm, write code, and present a solution where all 40 packages (listed in the attached “WGUPS Package File”) will be delivered on time while meeting each package’s requirements and keeping the combined total distance traveled under 140 miles for both 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.” The intent is to use the program for this specific location and also for many other cities in each state where WGU has a presence. As such, you will need to include detailed comments to make your code easy to follow and to justify the decisions you made while writing your scripts.

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 at what time the delivery occurred.

1. **ALGORITHM IDENTIFICATION**

The algorithm that was chosen for this project is the Nearest-Neighbor algorithm. The reasoning behind selecting this algorithm was due to the simplicity and effectiveness for a complete graph, that is a graph in which all vertices have an edge shared with all other vertices.

**B. PROGRAM OVERVIEW**

1. The way that the nearest-neighbor algorithm functions is by taking in a list unvisited destinations and an arbitrary starting point and then visiting the closest destination (“neighbor”) of the remaining destinations. Once a destination is visited, it is removed from the list and then placed into a separate array representing the best route. This process is repeated until the unvisited destinations list is empty, representing that all destinations have been visited.

The pseudocode for this algorithm, is as follows:

// the function takes the list of packages, a graph with edge weights, and

// a hash table with package data as arguments.

def create\_route(list\_of\_packages, graph, package\_hash\_table)

// Initialize empty array of unvisited destinations

unvisited\_destinations = []

// Initialize an empty array for the route

route = []

// This portion will add packages to unvisited\_destinations array

// get id from list\_of\_packages array

for id in list\_of\_packages:

// use id as the key to get the package object from hash table using // look\_up function

package = package\_hash\_table.look\_up(id)

// add package to array

unvisited\_destinations.append(package)

// This portion will determine the best-route using Nearest-Neighbor

// Loop continues until unvisited destinations is empty

while len(unvisited\_destinations) > 0:

min\_distance = 500 // arbitrary high value for comparison

next\_package = null

for package in unvisited\_destinations:

// compare\_distance returns the edge weight of two vertices

// if value is less than min\_distance, assign to min\_distance

if graph.compare\_distance(current\_location,

package\_location) <= min\_distance:

min\_distance = graph.compare\_distance(current\_location, package\_location)

// assign the package that had lowest value to next\_package

next\_package = package

// Once the package with the minimum distance is identified the package is // added to the route array

route.append(package)

// The package is then removed from the unvisited\_destinations array

unvisited\_destinations.remove(package)

// The while loop continues until there are no elements in the // unvisited\_destinations array

2. The programming environment for writing this code was the PyCharm IDE with Python 3.11.

**C. ORIGINAL CODE**

**C2. PROCESS AND FLOW COMMENTS**

**D. DATA STRUCTURE**

**D1. EXPLANATION OF DATA STRUCTURE**

**E. HASH TABLE**

**F. LOOK-UP FUNCTION**

**G. INTERFACE**

**G1. FIRST STATUS CHECK**

**G2. SECOND STATUS CHECK**

**G3. THIRD STATUS CHECK**

**H. SCREENSHOTS OF CODE EXECUTION**

**I1. STRENGTHS OF THE CHOSEN ALGORITHM**

**I2. VERIFICATION OF ALGORITHM**

**I3. OTHER POSSIBLE ALGORITHMS**

**I3A. ALGORITHM DIFFERENCES**

**J. DIFFERENT APPROACH**

**K1. VERIFICATION OF DATA STRUCTURE**

**K2. EFFECIENCY**

**K1B. OVERHEAD**

**K2. OTHER DATA STRUCTURES**

**K2A. DATA STRUCTURE DIFFERENCES**

**L. SOURCES**