**Core Algorithm Overview**

**Stated Problem:**

The purpose of this project is to create an algorithm using the Python programming language to develop an effective and efficient solution for the traveling salesman problem (TSP) which Wikipedia defines as the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" [[1]](#footnote-1) .

This assignment adds additional requirements of:

* Package delivery deadlines
* Packages which must be on the same truck
* Packages which must be on a specific truck
* Truck load limitation
* Truck speed constraint
* 2 Trucks can be operational at one time.
* One package has the wrong address assigned and must be fixed at 10:20am

**Program Blueprint and Architecture:**

Classes:

* csv\_import: imports all the data from the csv’s provided
* Config: Contains all the global variables for the application
* Truck: represents a single truck which follows a route.
* Route: Represents an ordered collection of deliveries and locations
* Delivery: Represents a collection of Packages which are delivered to the same location
* Location: a single point in a Route which has a name, address, city, state, zip.
* Package: The smallest unit to deliver which is imported from the packages.csv file.
* CLI: The Command line Interface which presents the user with options and reports
* Debug: functions which print out reports for Trucks, Deliveries, Routes, Packages
* Distance: A class which calculates the distances of routes and between two deliveries/locations.
* HashTable: A custom hash map class which is used instead of the python dictionary.

**Algorithm Overview:**

The proposed solution for this assignment uses a **brute force** method which tried to insert a delivery into each index within the route. Determining the miles which would be added to the route at a particular index. The method will return the route and the index which would cause the route to increase in the least number of miles possible which adhering to the deadlines of the packages within a delivery.

**Relationship between data points stored**

Route

Location

Package

Package

Package

Package

Delivery

Delivery

Delivery

Delivery

Location

Package

Package

Package

Package

Package

Package

Package

Package

Location

Package

Package

Package

Package

Location

Truck

Deliveries are composed of Packages grouped by locations.  
Routes is a list of Deliveries with a sorted list of Locations to follow through the route.  
Each Truck is assigned a Route.

**Logic of Algorithm Applied**

The **distribute\_deliveries\_to\_trucks** and the **added\_distance** function within the routeclass performs this procedure with the following pseudocode.

**distribute\_deliveries\_to\_trucks** (Big O(n^2))**:**

**first assigns any deliveries which must be assigned to truck 2.   
Retrieve the remaining deliveries with unassigned trucks.**  
**Define the time during when a particular truck will depart the depot (starting point)**  
**Loop through each Delivery and call the added\_distance method** **for each truck**   
 **Calculate the best truck and best route index for a given delivery.** **Check the delivery ETA is before the deadline and return the Route index or None.**  
**Assign the delivery to the truck with the least added distance**.

**Repeat this loop until all deliveries are assigned to a truck.**

**Handling Constraints:**

1. **Package delivery deadlines**The **added\_distance** function will check to make sure all packages arrive to destination before deadline when returning a route and index  
   If no options were found to meet these criteria None will be returned.
2. **Packages which must be on a specific truck and Packages which must be on the same truck**The **distribute\_deliveries\_to\_trucks** methodfirst assigns any deliveries which have packages with assignment constraints
3. **Truck speed constraint**The Truck class has a property of **max\_speed** which is used to calculate how many miles were traveling during a specified time frame.
4. **Truck load limitation**The Truck class has a **will\_fit** method which calculates to see if a truck has enough space for a delivery before doing any additional assertions and calculations.
5. **Two Trucks at one time.**The main program loop will check to see if a truck has completed its’ route and it is past 10:20am.  
   Then it will dispatch Truck 3 and start its route.
6. **Handling the wrong address constraint:**Keep Package 9 unassigned since the address is incorrect, assign it to Truck 3 when the correct address is available (10:20 am)

**Adaptability to a changing market**This application has some flexibility in adjusting to more packages up to the point that another truck is needed.

**Scalability**If the user were to need additional trucks another truck can be easily added and dispatched.   
CSV Files can be easily loaded with more data.

**Efficiency and maintainability of the software.**The application is split into logical files which individually handle each aspect of the structure of the application. Each method is commented to be able to easily follow the lifecycle of the application.  
In order to make changes and make sure the results are as expected the future developer can rely on the Debug Class to output a printed table of all the data structures within the application at any given time using the CLI

**Self-adjusting data structures chosen**

The Hash Table Class implements a simple hash key taking into account the sum of the Unicode code point value for each character in the key’s string modulo into the hash Table’s size[[2]](#footnote-2).  
The default value for the size of the HashTable Class is 128 but it can be adjusted.  
Magic Methods have been implemented to make interaction with the HashTable class simpler to iterate with and check if it contains a value.

**Strengths of the chosen algorithm**

1. Having a Brute Force algorithm makes sure all options are taken into account and evaluated to select the most efficient route.
2. Having a simple algorithm such as Brute force for a small sample of packages and possible routes makes the program simpler to test.

**Different Approach**

This program is not scalable to many packages as is 2-opt or Dijkstra’s[[3]](#footnote-3) algorithms.  
The Trucks are explicitly defined so any changes to the trucks would need to be defined explicitly.  
A different approach would make the program loop the array and not call each truck explicitly

**Screen Shots**

**Screenshots of Code Execution:** CLI Options presented to user.

**Text

Description automatically generated**

**First Status Check 9:30am**

**Text

Description automatically generated**

**Second Status Check 10:20am**

**Text

Description automatically generated**

**Third Status Check 1:00pm**

**Text

Description automatically generated**

Delivery Chart of which truck each delivery was assigned

**Text

Description automatically generated**

Route table listing the locations each specified truck will go through throughout the day.

**Text

Description automatically generated**

Truck table notifying the current location of the truck, packages on board and miles it will travel during the assigned route.

**Text

Description automatically generated**

1. Traveling Salesman Problem. (2020) wikipedia.org

   https://en.wikipedia.org/wiki/Travelling\_salesman\_problem [↑](#footnote-ref-1)
2. Lysecky, R., & Vahid, F. (2018). C950: Data Structures and Algorithms Ii. zyBook. [↑](#footnote-ref-2)
3. Nilsson, C. (2003). Heuristics for the traveling salesman problem. Linkoping University. [↑](#footnote-ref-3)