UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Physical Sciences
Department of Electronics and Computer Science

A project report submitted for continuation towards
MEng Software Engineering

Supervisor: Dr Thai Son Hoang
Examiner: Professor Michael Butler

Simulation of Vehicle Routing Problem

by Suyash Datt Dubey

May 24, 2020

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for continuation towards a MEng Software Engineering

by Suyash Datt Dubey

Vehicle Routing Problems (VRP) are combinatorial optimisation problems that are related to the transportation of people or goods between depots and customers. Finding an optimal solution to the vehicle routing problem is NP-hard. There is no single algorithm that can be used to find the best route for different variations of the VRP. Therefore, in general, heuristics and metaheuristics are used to solve VRPs in real-world applications due to the scope of the problem. Obtaining and implementing optimal routes for a set of customer locations is crucial since it results in reduced costs for organisations and also has a positive impact on the environment. This project aims to develop an application that can generate near-optimal routes for given customer locations, using algorithms for different variations of the VRP. The efficiency, scalability and effects of constraints on the different VRP variations are examined. These routes can then be pursued by a single or a fleet of autonomous vehicles to provide services in the least possible time to maximise customer satisfaction.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I have acknowledged all sources, and identified any content taken from elsewhere.
- I have used some resources produced by someone else. In this project I have used a number of libraries all of which have been documented in the dissertation.
- I did all the work myself, or with my allocated group, and have not helped anyone else .
- The material in the report is genuine, and I have included all my data/code/designs.
- I have not submitted any part of this work for another assessment.
- My work did not involve human participants, their cells or data, or animals.

Acknowledgements

I would like to extend my sincere thanks to my supervisor, Dr Thai Son Hoang, for his guidance and continuous support throughout this endeavour. His advice, feedback and prompt replies were invaluable during the course of the project. His motivation and sincerity during testing times inspired me to accomplish this project successfully. As the UK and rest of the world went into lockdown due to COVID-19, he continued to support me remotely. It was a great privilege and honour to work under his guidance.

I am extremely grateful to my second examiner Professor Michael Butler for his advice, which helped me enhance my project.

I am thankful to my friends for their moral support and reassurance at all times.

I am indebted to my parents for their encouragement and motivation. This project would not have been successful without their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Vehicle Routing Problems (VRP) are combinatorial optimisation problems that deal with the transportation of people or goods between depots and customers. The objective is to identify a set of near-optimal routes for multiple vehicles in the shortest possible time, while minimising operating costs and maximising productivity. Optimal routes are the routes that maximise the number of customer locations the vehicle visits while minimising the length of the route, the total duration of time spent by the vehicle on the route and fuel consumption of the vehicle. Determining effective solutions for route planning have numerous socio-economic and environmental benefits.

The VRP is different from the conventional Travelling Salesman Problem (TSP) as it deals with several vehicles and additional constraints such as vehicle capacities, customer availability, etc.

1.1 Existing Problems

1.1.1 NP-hard

Finding an optimal solution to the Vehicle Routing Problem is NP-hard, which means it is impossible to find an optimal solution in polynomial time (Bruniecki et al., 2016). Therefore, heuristics and metaheuristics are used to solve VRPs in practical applications due to the scope of the problem. In certain variations, finding a suboptimal or workable solution is NP-hard (Kilby and Shaw, 2006).

1.1.2 Limited Scalibility

Many solutions to the VRP do not scale well, and hence problems involving more than a hundred customers become extremely complicated to solve (Gounaris et al., 2011). For a large number of locations, the algorithms often fail to find solutions.

1.1.3 Additional Constraints

Applying additional constraints such as vehicle capacities, time windows, etc. further complicates the problem and affects the ability of the algorithms to find solutions.

1.2 Project Goals

The goal of this project is to create an application that simulates the Vehicle Routing Problem. The application must identify and analyse the set of routes for multiple vehicles and locations in the shortest possible time, while minimising operating costs and maximising the number of locations visited.

The application will be entirely software-based. The main application will be deployed on Microsoft Azure App Service and implemented using a combination of JavaScript and HTML/CSS frontend and Python backend. Storage and retrieval of vehicles, locations and algorithms will be achieved using Microsoft Azure Table Storage and Microsoft Azure Function Apps. For evaluation purposes, the algorithms will be implemented using Java.

A convoy of vehicles will be considered to simulate routes and delivery systems. It is assumed that the vehicles will operate autonomously without any direct human interaction. The routes can either be established by the user or generated by various algorithms programmed in the application. Programming the vehicles to perform the deliveries is out of the scope of this project.

The application will incorporate multiple constraints and variations of the Vehicle Routing Problem such as customer availability, vehicle mileage, the capacity of vehicles, etc. The objectives are different for each variation of the VRP. Finally, the user can select the algorithm and route they find most appropriate as per the requirement.

For evaluating the performance of multiple variations of the VRP, the problem will be simulated by varying the problem size. The size of the problem can be varied by selecting different number of customer locations to visit, route lengths, number of available vehicles, vehicle capacities, etc.

It will be a user-friendly and scalable application that will streamline the planning of transportation routes.

Chapter 2

Literature Review

2.1 The Vehicle Routing Problem

The Vehicle Routing Problem is one of the most studied optimisation problems of all time. It involves finding the best route for a fleet of vehicles to deliver goods to customers. The problem was first proposed by Dantzig and Ramser in 1959. They studied a real-world problem involving the delivery of gasoline to service stations and proposed a mathematical and algorithmic approach for solving it (Toth and Vigo, 2002).

Hosny (2011) explains how the VRP is a generalisation of the well-known Traveling Salesman Problem (TSP). Both the TSP and VRP are node routing problems since the service they provide is associated with nodes or locations. The VRP, in its simplest form, is a type of node routing problem, in which a single vehicle visits several locations, with no limitations on the vehicle's capacity. In most cases, it is also assumed that the routes start and end at the same depot. However, most variations of the Vehicle Routing Problem involve multiple vehicles and identification of a specific depot location. Vehicles serve several different customer locations, each of which, in certain variations, have distinct service requirements (pick up or delivery). It is also noteworthy that each route identified in the VRP is a Hamiltonian cycle. This is because on a particular route, each location is visited exactly once and the start and end of each route are at the same location, i.e., at the depot.

Hasle et al. (2007) highlights the fact that existing algorithms can only solve problems involving 50-100 locations. It even suggests that finding an algorithm that solves all variations of the VRP optimally in an acceptable time, even in the future, is improbable. However, the ability to find suitable solutions for variations of the VRP within adequate time has improved since the problem was first introduced. This is mainly due to an increase in computing power and improvements in algorithms and heuristics for the VRP.

There are several variations of the problem, some of which are explored in this project:

- **Capacitated Vehicle Routing Problem (CVRP):** This is a variation of the VRP in which each vehicle's capacity for carrying goods is limited, and this limit should not be exceeded for any of the vehicles. It is also assumed that all vehicles are stationed at a single depot and that each route will begin and end at the depot. Additional constraints enforced are that each customer's location will be visited only once and that the total demand of a route will not exceed the capacity of the vehicle (Borčinova, 2017). Often the service time window constraint is also imposed, and the problem is then known as Capacitated Vehicle Routing Problem with Time Window (CVRPTW).

- **Capacitated Vehicle Routing Problem with Penalties (CVRP with Penalties):** This is a slight variation of the Capacitated VRP in which new costs called penalties are introduced at all locations. A penalty is applied to the total distance travelled if a visit to a location is dropped. The algorithm finds a route that minimises the total distance plus the sum of the penalties for all locations dropped (Google Developers, 2020a).

- **Open Vehicle Routing Problem (OVRP):** This variation is similar to the CVRP. The difference is that in the OVRP, vehicles do not return to the depot after they have completed their journeys along their respective routes (Ge et al., 2010). Since the vehicles do not return to the starting depot, in the OVRP each route is a Hamiltonian path instead of a Hamiltonian cycle. This scenario is usually encountered by organisations that outsource transport services to other organisations (Brandão, 2004).

- **Pickups and Deliveries Vehicle Routing Problem:** The aim is to identify routes for vehicles to complete both pickups and deliveries of items between specified locations. The pickup and delivery locations for a particular item are distinct. It is assumed that the vehicles start and end their routes at the same depot (Golden et al., 2008). Theoretically, it is assumed that an unlimited number of vehicles will be available, and every customer will be delivered their items. However, this is not practical and therefore identifying optimal routes is important to maximise customer deliveries with a limited number of vehicles (Andriansyah et al., 2018). In the case where people are transported, the problem is also called the dial-a-ride problem.

- **Vehicle Routing Problem with Initial Routes:** In this problem, a set of initial routes is specified for the vehicles in the VRP. The algorithm is run to obtain the initial solution routes, after which the basic VRP algorithm is also executed. This is useful for cases where a good solution to the problem is already known, and where comparing performance of different possible routes is required.

- **Vehicle Routing Problem with Starts and Ends:** In this variation of the VRP, different start and end locations can be set for each vehicle. In many cases, the total distance of routes is shorter since the vehicles are not required to start or end at the depot.

- **Vehicle Routing Problem with Time Windows (VRPTW):** A service time window (TW) constraint is imposed on the VRP, in which the visiting time of a vehicle at a location is restricted according to the specified time period. This enforces that each customer must be served during the time period specified. If the vehicle reaches the customer destination before the start of its time window, the vehicle should wait until the specified service time begins. Similarly, the constraint is violated if the vehicle arrives after the time window has ended. It is assumed that all customers require the same type of service (either pickup or delivery but not both) and vehicles start and end their routes at the same depot. Another constraint that is strictly imposed is that the vehicle capacity should not be exceeded at any time (Hosny, 2011).
- **Vehicle Routing Problem with Time Windows with Resource Constraints (VRPTW with Resource Constraints):** A variation of VRPTW in which certain constraints are applied at the depot. Vehicles need to be loaded before departing the depot and unloaded upon return. There are only two available loading docks so at most two vehicles can be loaded or unloaded simultaneously. As a result, some vehicles must wait for others to be loaded, which delays their departure from the depot. The problem involves finding routes for the VRPTW that satisfy the loading and unloading constraints at the depot as well (Google Developers, 2020b).

2.2 Importance of Optimal Route Planning

European Environment Agency (2019) illustrates that in 2016, the transport sector was responsible for 27% of total greenhouse gas emissions in the European Union, out of which, road transportation contributed to 72% of the total emissions. Transport is highly dependent on fossil fuels and was one of the contributing factors to the 0.6% increase in emissions in 2017. However, it still seems likely that the EU will be able to meet their 2020 target of reducing emissions by 20% compared to 1990, albeit narrowly. In spite of this, the EU's goal of reducing emissions 40% by 2030 seems to be in peril as reductions are expected to slow down due to current national policies. Drastic measures will be required to ensure this 40% reduction. Since the transportation sector is a significant contributor to air pollution, optimal planning techniques could help reduce emissions and reduce transportation costs as well.

E-commerce and online shopping businesses are rapidly expanding in today's world, thereby increasing the demand for more efficient planning strategies. In the EU, the transport sector accounts for more than 10% of GDP and employs 10 million workers. From a more global perspective, transportation requirements are geographically imbalanced in most countries, which often leads to lower consumption. This imbalance, coupled with the lack of coordination, is the primary reason for low efficiency. Even today, transport planning is usually carried out manually in most cases. However, robust applications for identifying near-optimal routes are

being developed at a very fast rate (Hasle et al., 2007). These applications aim to help e-commerce businesses reduce costs and use their resources more efficiently.

2.3 Autonomous Vehicles

An autonomous vehicle operates without any human interaction by detecting its environment. It can travel anywhere that a regular vehicle can and perform all the tasks of a human driver, without the presence of a human at any time (Synopsys, 2019).

- 94% of serious accidents occur due to human error. Autonomous vehicles lead to increased safety on roads by eliminating the need for human drivers and the risk of reckless driving (NHTSA, 2019).
- These vehicles provide economic and social benefits. They lead to more efficient vehicle traffic and reduce congestion, which results in reduced travel time and transportation costs (Arbogast, 2017).
- Autonomous vehicles can be programmed to reduce fuel consumption and pollution to a significant degree (Ohio University, 2019). Researchers have demonstrated that such vehicles can reduce total fuel consumption by up to 40% (Arbogast, 2017).

Autonomous vehicles are being tested rigorously and will soon become the primary mode of transportation used by both people and services. For instance, currently, in some cities, Amazon provides the Prime Air service, in which drones are used to deliver packages to customers. The next step is most definitely larger autonomous vehicles that can transport packages to and from customers without any human interaction at all.

Chapter 3

Analysis and Specification

This section outlines the functionality and features that the application must incorporate and draft a set of requirements based on this.

3.1 Stakeholder Analysis

Any entity that is directly or indirectly involved or affected by the outcome of a project is known as a stakeholder. Stakeholders can be primary, secondary, tertiary or facilitating depending on how they are involved/affected by the project. The major stakeholders of the project are examined below in Table 3.1.

Stakeholder	**Type**	**Description**
Route Planners / Transportation Managers	Primary	Manage vehicles and locations and select appropriate algorithms to generate routes.
Field Staff / Autonomous Vehicle Managers	Primary	Meeting their delivery targets and keeping the staff motivated.
Organisations and Businesses	Secondary	Usage of the application will lead to optimisation of labour and resources, thereby increasing profitability. Subsequent reductions in emissions support corporate social responsibility.
Vehicle Maintenance Department	Tertiary	Maintenance costs of vehicles will be reduced significantly
		Continued

Developer	Facilitating	Facilitates the design, development, maintenance and extension of the application.

Table 3.1: Stakeholder Analysis

3.2 Use Case Diagram

A use case diagram provides a visual description of a user's interaction with the application that is currently under development. It describes the expected behaviour of the system in various scenarios and illustrates the relationships between use cases, actors, and systems. Use case diagrams are useful because they help create an application from the end user's perspective (Bruegge and Dutoit, 2013).

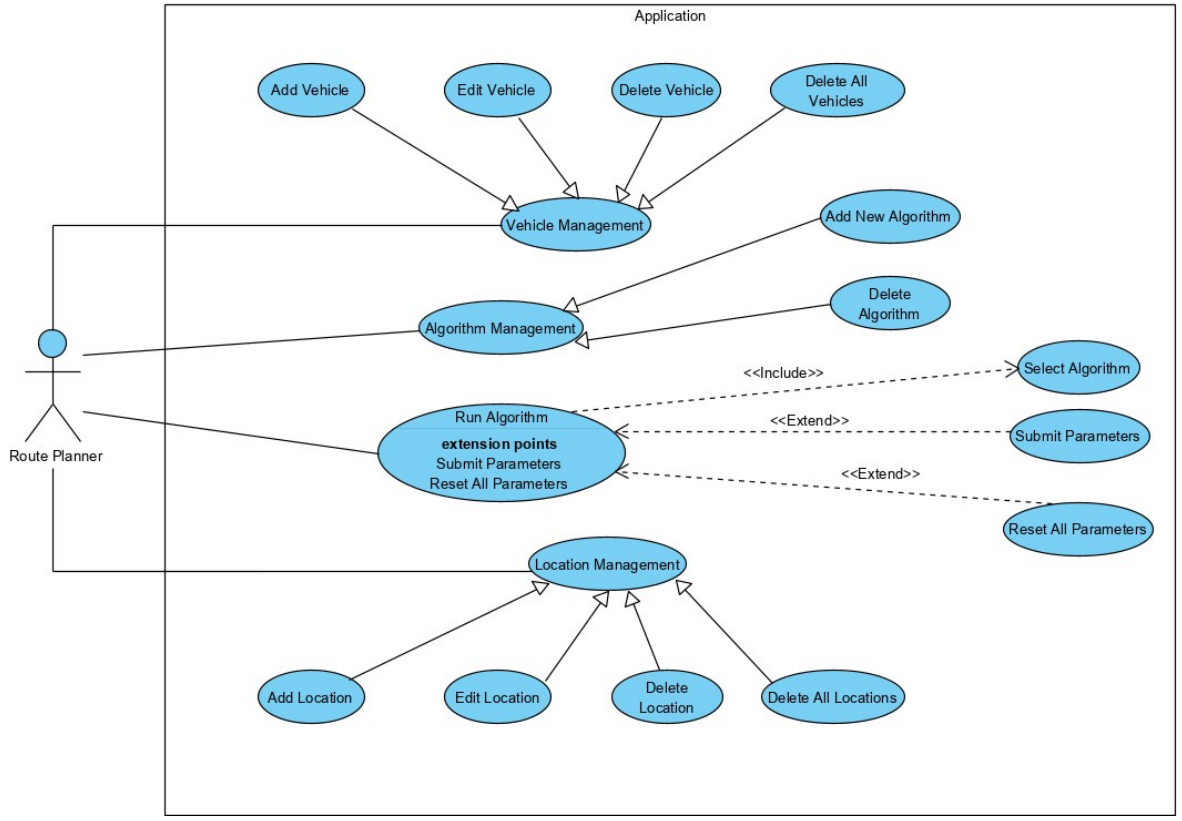


FIGURE 3.1: Use Case Diagram

3.3 Requirements Analysis

Requirements analysis involves identifying and describing the features and behaviour an application must possess. Requirements can be of two types: functional and non-functional (Bassil, 2012).

3.3.1 Functional Requirements

Functional requirements outline the users' interactions with the software (Bassil, 2012). The application must possess the following functionalities in order to be useful:

- The user can add, edit and delete vehicles and their capacities.
- The user can add, edit and delete location coordinates.
- The user can add and delete new algorithms.
- The user can select an algorithm to run. This will generate the route in textual form and display a graph of the routes taken by each vehicle.
- The user can specify the parameters for algorithms that require input parameters.
- The user can access a help page describing the various default algorithms, their input parameters and the format of these parameters.

3.3.2 Non-functional Requirements

Non-Functional requirements refer to the constraints and requirements imposed on the design and operation of the application rather than its behaviour (Bassil, 2012). The application should incorporate the following qualities:

- The user interface of the application should be intuitive and straightforward to use.
- Certain user interface elements, that are not expected to be used while the algorithms are running, should be disabled to avoid errors in the application.
- The application should be fast and responsive to prevent long waiting times.
- The application should be designed such that it can be extended easily to include more algorithms and additional features.

Chapter 4

Design

The design phase of a project involves creating a plan and devising solutions for the development of various aspects of the application (Bassil, 2012). This chapter describes the various design decisions that were made with regards to the user interface and the database, and includes UML sequence diagrams representing step-by-step user interactions with the application.

4.1 Database Design

An Entity Relationship Diagram (ERD) was created to determine the attributes to be stored in the tables. It provided a clear and concise notion of the data and data types in each table. Three tables were created: Vehicles, Locations and Algorithms as seen in Figure 4.1.

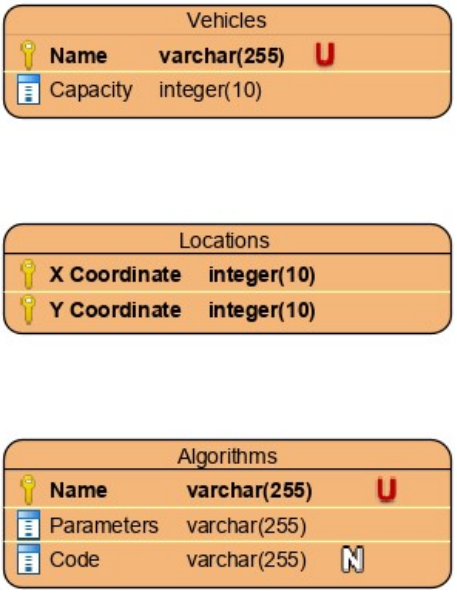


FIGURE 4.1: Tables in the Database

The tables are independent of each other and have no foreign keys or relationships between them.

4.2 Model-View-Controller

During the development of applications, modularity of components is extremely beneficial. One method of achieving this separation of components is by employing the Model-View-Controller (MVC) framework (Burbeck, 2012).

4.2.1 Model

The Model manages the shape, behaviour and data of the application domain. It is domain-specific and should work independently of the View or Controller. The Model interacts with the View by responding to requests for information about its state and with the Controller by responding to instructions regarding change of state (Burbeck, 2012).

4.2.2 View

The View handles the textual and graphical output rendered from the domain data to the display (Burbeck, 2012).

4.2.3 Controller

The Controller is the interface that mediates between the Model and the View. It interprets inputs from the user and directs the Model and View to change accordingly (Burbeck, 2012).

4.3 Class Diagram

For developing the Model, a class diagram was created, showcasing the data that the application contains. A class diagram defines the system structure by denoting the system's classes, properties, methods and the relationships between objects (Bruegge and Dutoit, 2013).

The class diagram in Figure 4.2 describes the three main classes that comprise the application domain. The various attributes, their types and methods in the classes are also depicted in the diagram. The association relationships are justified, because at least one vehicle and two locations are required to run an algorithm.

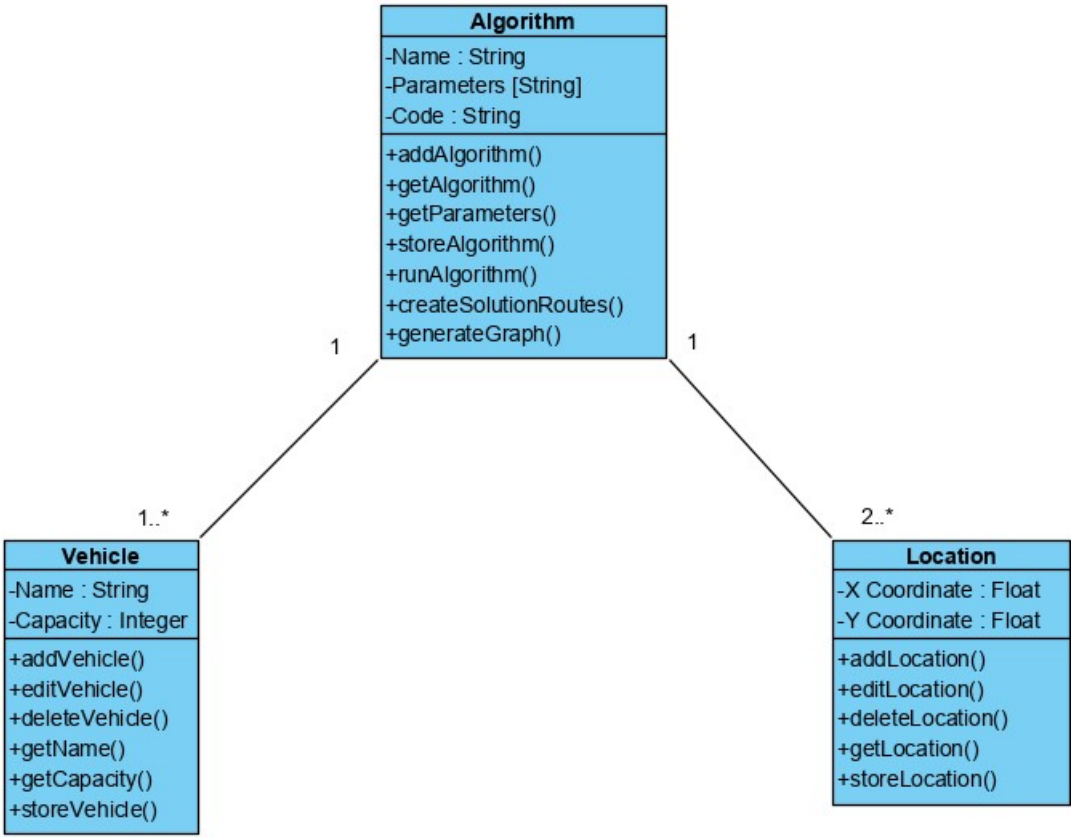


FIGURE 4.2: Class Diagram

4.4 User Interface Design

For the View, wireframe diagrams were drawn to acquire a distinct perception of the appearance of the user interface.

Wireframes are digital tools that are used to layout the fundamental interactions and prominent design elements of an application. They provide a template for the functionality and features of the application, and the user's interactions with it. Lucidchart was used to create wireframes showcasing the initial design of the application.

The plan was to divide the user interface into three main sections: Vehicle Management, Location Management and Vehicle Routing. The initial wireframe design for the user interface is given in Figure 4.3.

In the initial design, the Reset All Parameters, Delete All Vehicles and Delete All Locations buttons were not included. These features were considered essential and hence added during the development process.

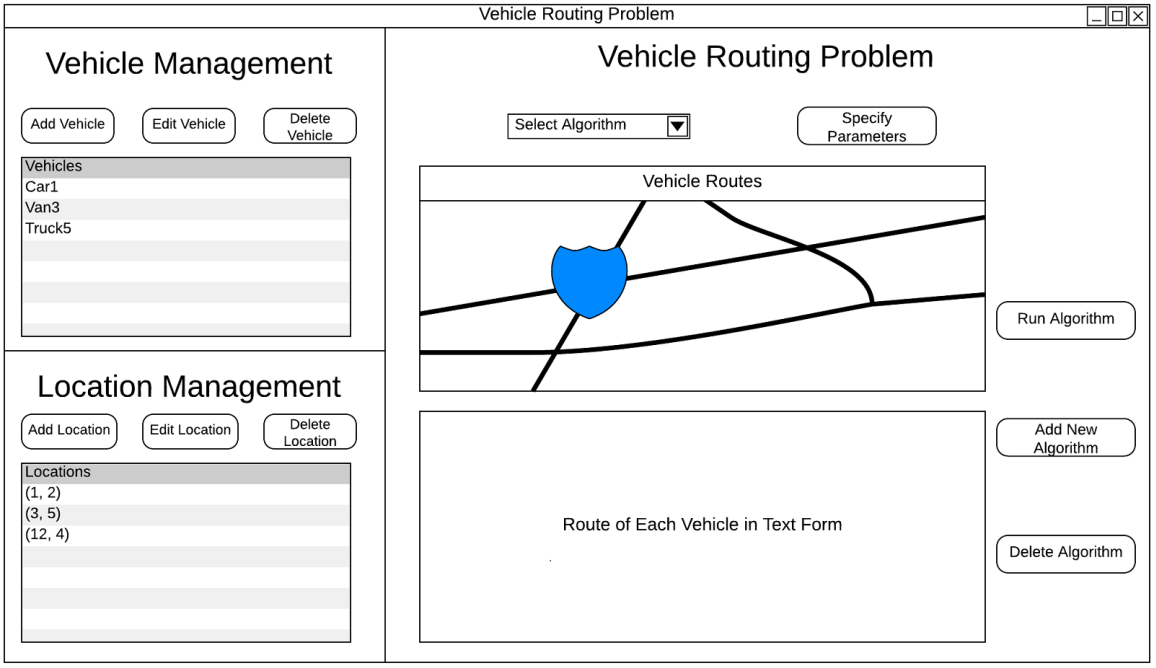


FIGURE 4.3: User Interface Wireframe

The plan included creating dialog boxes for all the Add and Edit buttons in Vehicle Management and Location Management and also for the Add New Algorithm and Specify Parameters buttons.

Figure 4.4 shows the design of boxes for the Add and Edit buttons. A similar layout was used for the dialog box of Specify Parameters button.

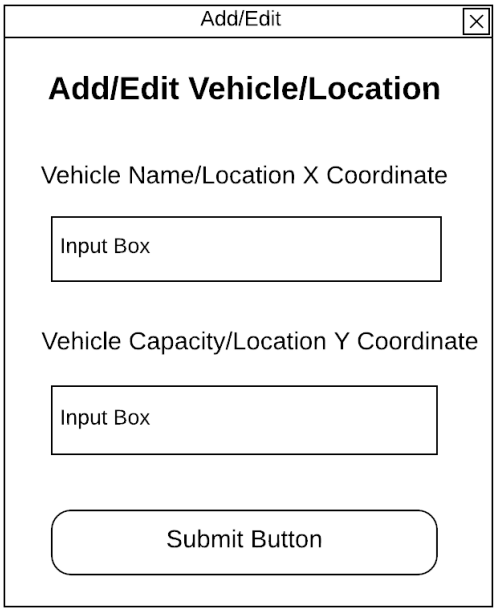


FIGURE 4.4: Add/Edit Dialog Box Wireframe

The box for the Add New Algorithm button was slightly different since the code for the algorithm is also entered. This dialog box takes up almost the entire page.

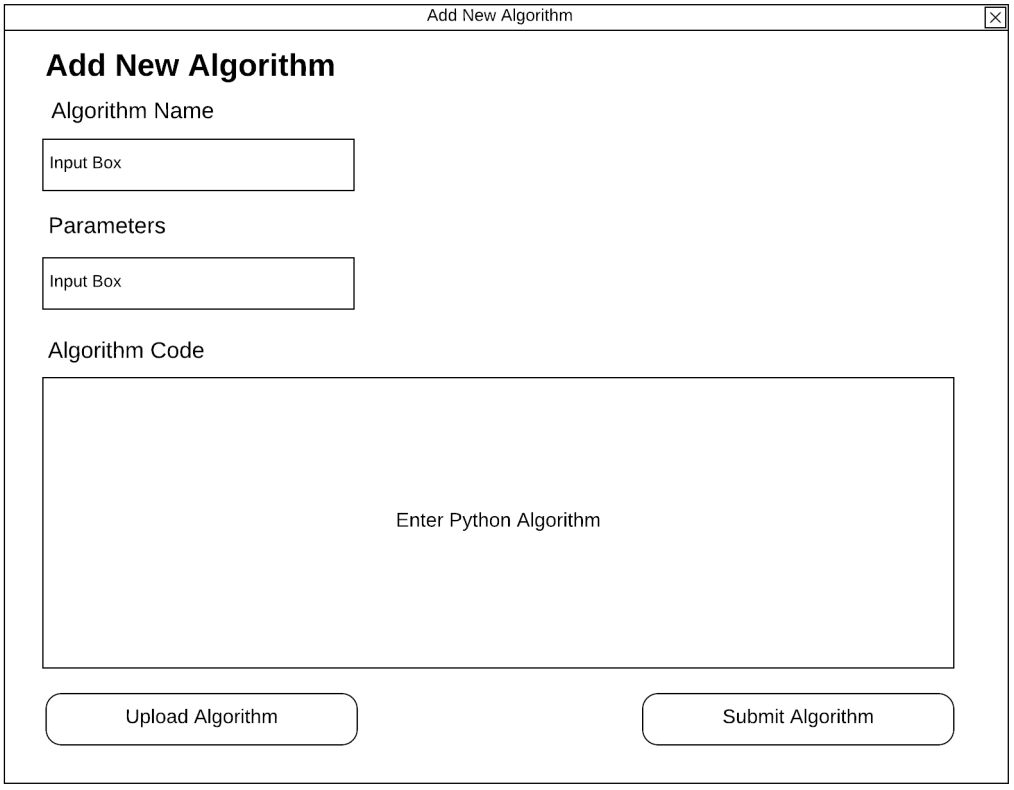


FIGURE 4.5: Add Algorithm Dialog Box Wireframe

4.5 UML Sequence Diagrams

Sequence diagrams describing various scenarios of user interaction were created for the Controller phase. Sequence diagrams illustrate the occurrence of certain operations in the system (Bruegge and Dutoit, 2013).

4.5.1 Adding and Editing a Vehicle or Location

The sequence of events for adding a new vehicle is depicted in Figure 4.6.

The user enters the vehicle name and capacity on the web application and clicks the submit button in the dialog box. If a vehicle with the same name already exists, the user is notified to enter a different vehicle name. If the vehicle name is unique and does not exist in the table, the vehicle name and its capacity are stored in the Vehicles table, the user is notified of the addition, and the list on the user interface is updated.

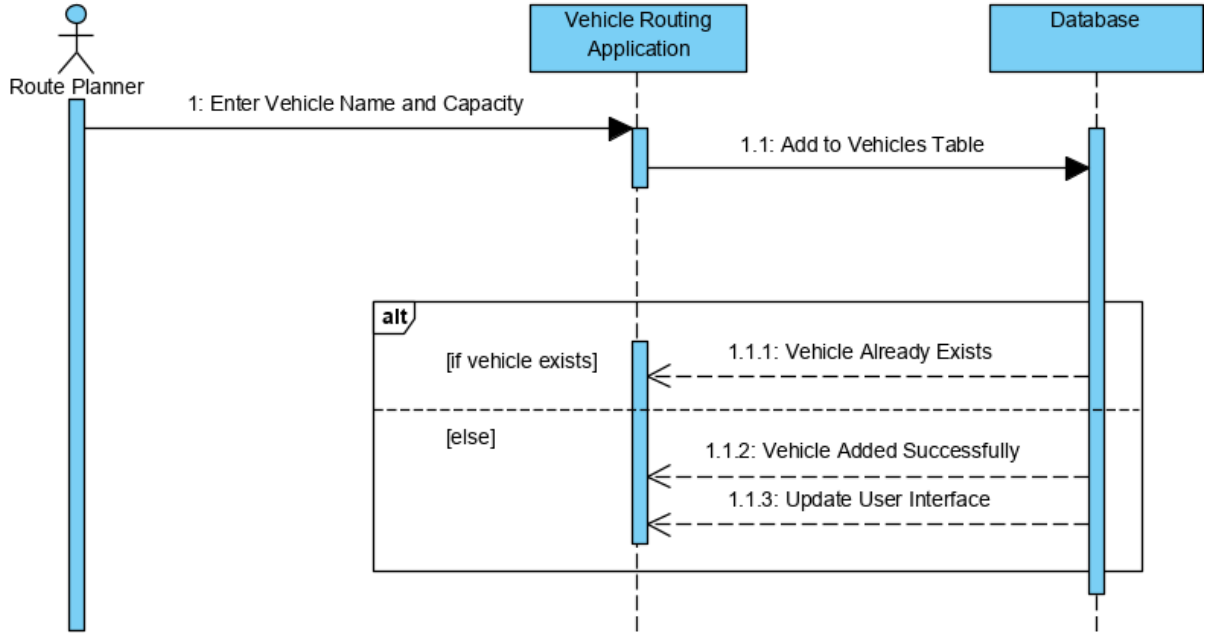


FIGURE 4.6: Adding and Editing Vehicle Sequence Diagram

The sequence of editing an existing vehicle is similar.

Figure 4.7 describes the events in the process of adding a new location.

The user enters the x and y coordinates for a location in the dialog box and clicks the submit button. The subsequent operation is similar to adding or editing a vehicle.

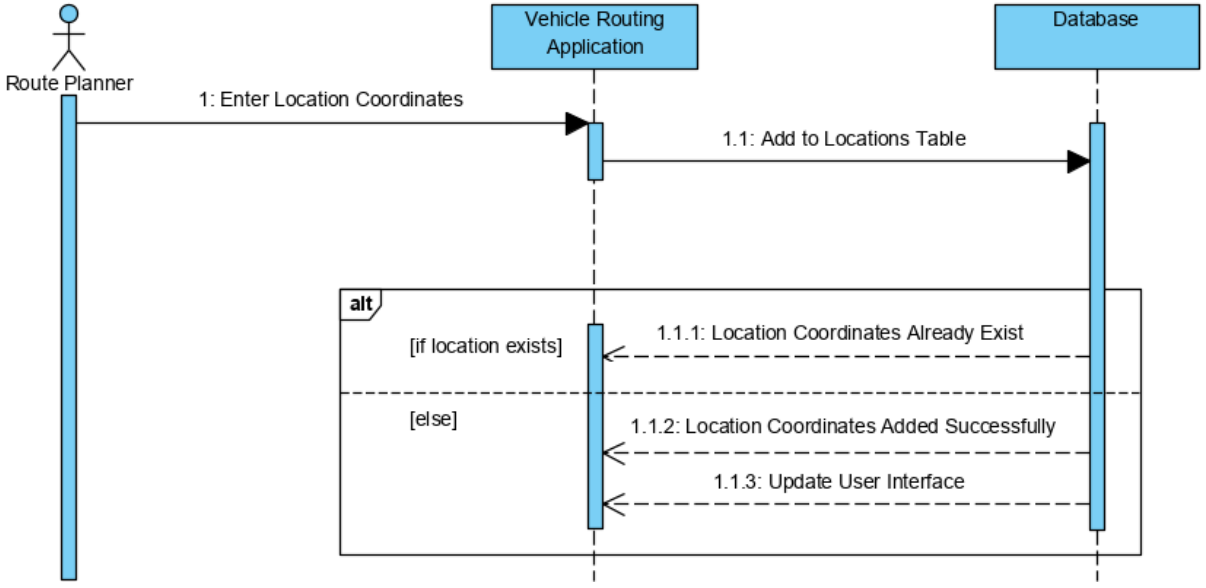


FIGURE 4.7: Adding and Editing Location Sequence Diagram

4.5.2 Add New Algorithm

The sequence of events involved in adding a new algorithm is depicted in Figure 4.8.

The user enters the name of the new algorithm, the required parameters and the code in the dialog box. If an algorithm with the same name already exists, the user is notified to enter a different name. Alternatively, if the algorithm name does not exist, then the new algorithm along with the parameters and code are stored in the Algorithms table. The user interface is then updated.

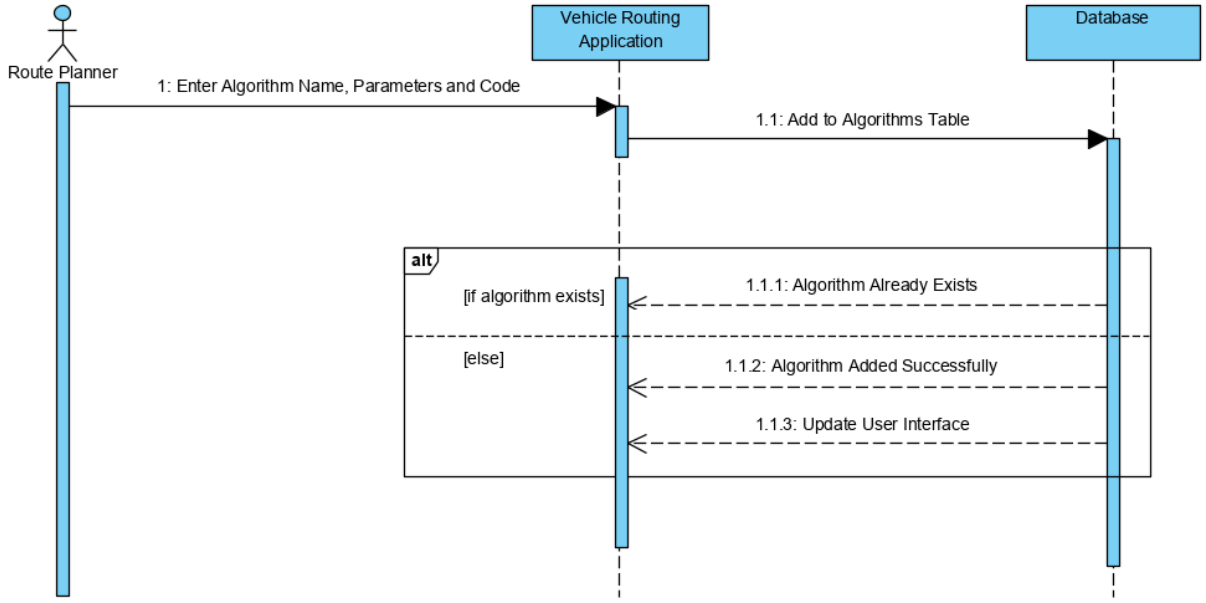


FIGURE 4.8: Adding New Algorithm Sequence Diagram

4.5.3 Delete Vehicle/Location/Algorithm

Figure 4.9 outlines the deletion of a vehicle, location or algorithm.

The user selects the vehicle, location or algorithm to be deleted and clicks the corresponding delete button. The selected entity is then deleted from the table, and the user interface is updated.

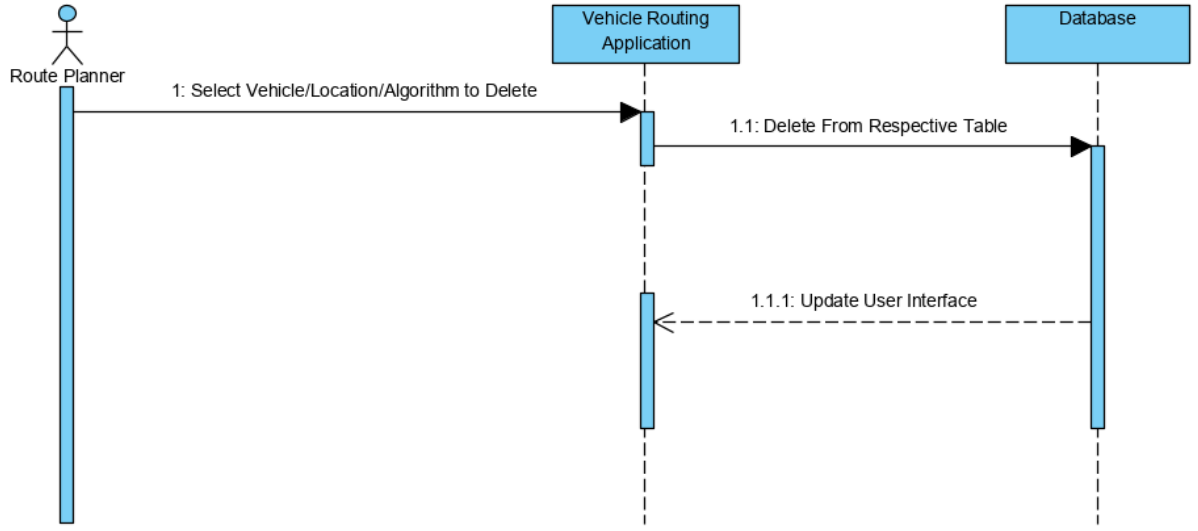


FIGURE 4.9: Delete Vehicle/Location/Algorithm Sequence Diagram

4.5.4 Run Algorithms

The sequence of operations for running an algorithm is described in Figure 4.10.

The user first selects an algorithm and enters the related parameters in the appropriate format. Next, the algorithm execution is initiated. This involves retrieving the locations and vehicles from the tables and returning them to the web application, which then sends the locations, vehicles and user-inputted parameters to the Python algorithm. If a solution exists, the Python algorithm computes a solution route and generates a graph, both of which are displayed on the application. Otherwise, the user is notified that a solution does not exist.

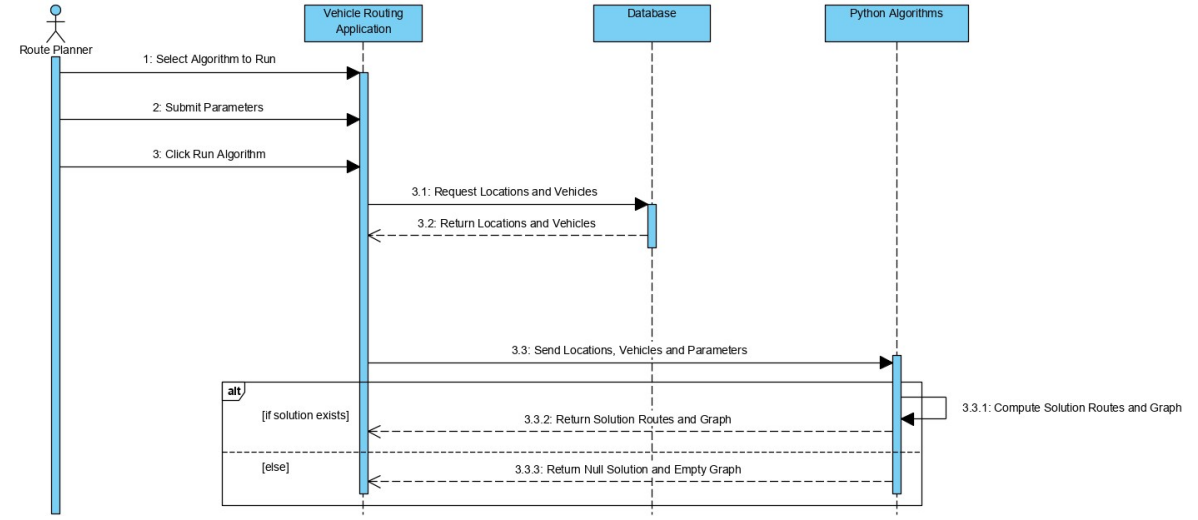


FIGURE 4.10: Run Algorithms Sequence Diagram

Chapter 5

Implementation

The implementation phase involves converting the requirements and design specifications into a functioning application. Actual code is written and compiled during this phase (Bassil, 2012). This section describes the possible approaches for developing the application and the ones adopted for implementing the web application.

5.1 Microsoft Azure vs Google App Engine vs Amazon Web Services

A selection had to be made between Google Cloud Platform (GCP), Amazon Web Services (AWS) and Microsoft Azure for developing the application. Microsoft Azure was the ideal choice as it provides higher availability and redundancy compared to GCP and AWS due to multiple possible deployment regions worldwide. Azure surpasses the other platforms in terms of agility, speed of deployment and operation. It enables reusability of existing resources such as code, libraries, etc. It is flexible and scalable, and therefore, allows the implementation of functionalities and services as and when required. Developers can build and use their own services and need not be concerned about the provision and maintenance of the hardware and software required (Padhy et al., 2011). Azure provides a high level of security and maintains multiple backups of data which means the chances of data loss are low. It enables effective data recovery in case of losses.

5.2 Azure Table Storage vs Azure SQL

For storage, a selection had to be made between Azure Table storage and Azure SQL Database.

Azure Table Storage is a service that stores large amounts of structured, NoSQL (non-relational) data in the cloud with a schemaless design. Azure Tables offer additional security because only

authenticated calls from inside and outside the Azure cloud are accepted. The datasets stored in Azure Tables do not require complex joins and foreign keys, and data is stored in a serialised format such as JSON, enabling efficient querying of data. Scalability is another advantage as new columns can be added to existing tables at any time. Azure Tables also work more effectively with web applications compared to Azure SQL Database (Microsoft, 2018). There are three default columns in each Azure Table, the PartitionKey, RowKey and the Timestamp. The PartitionKey and RowKey together act as the Primary Key of the table. Querying in Azure Table Storage occurs via HTTP GET and POST requests and Request URIs.

For the reasons stated above, Azure Table storage was the preferred choice.

The Azure Table Storage components used by the application are shown in Figure 5.1.

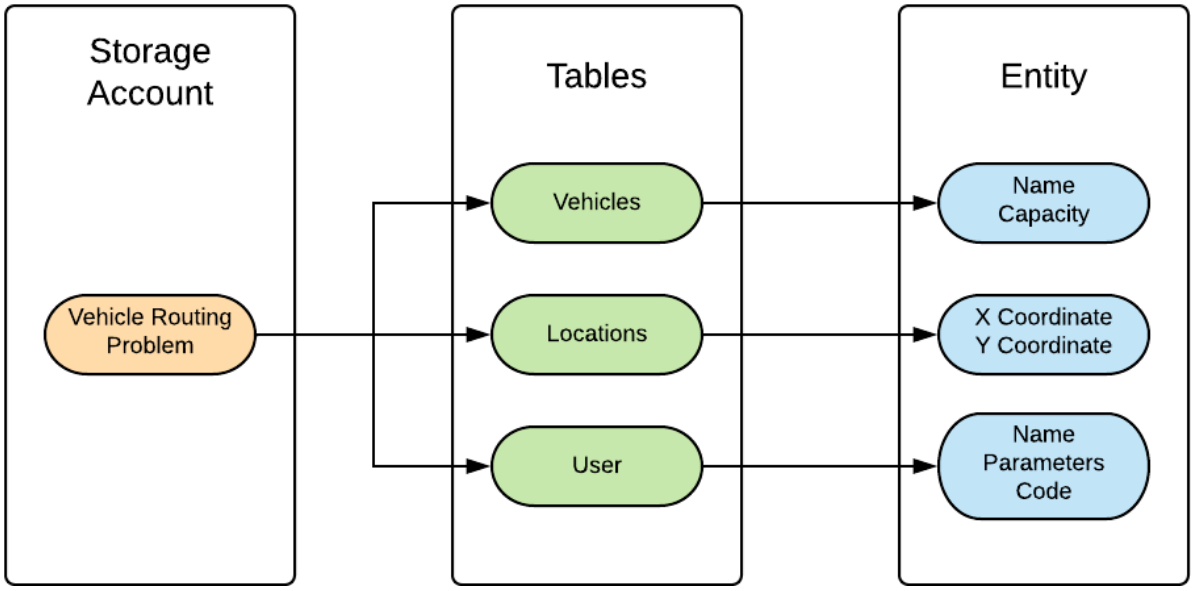


FIGURE 5.1: Azure Table Storage

In the Vehicles table, the Name is stored as the PartitionKey and the Capacity as the RowKey since two vehicles cannot have the same name and capacity. Similarly, in the Locations table, the X Coordinate is stored as the PartitionKey and the Y Coordinate as the RowKey. Storing the values in this manner boosts querying efficiency.

In the Algorithms table, the Name is stored as the PartitionKey and the Parameters as the RowKey. Another column Code stores the Python Code for newly added algorithms as a string.

5.3 Azure Function App

Communication between the web application and Azure Table Storage was achieved with the help of Azure Function App. Multiple Azure Functions can be created inside a single Function App.

Azure Functions are a form of serverless computing through Functions as a Service (FaaS). In serverless computing, the cloud provider dynamically manages the allocation and provision of servers. Certain aspects of the server-side code are written by the developer and run in stateless compute containers that are event-driven and completely managed by the cloud provider. These stateless containers can be triggered by various events such as HTTP requests, file uploads, Table Storage events, etc. The code written by developers is supplied to cloud providers in the form of functions; therefore, FaaS also falls under the category of serverless computing (GlobalDots, 2019).

Using FaaS and serverless computing is beneficial since it offers greater scalability, flexibility and reduces development time. Deploying, changing and updating functions is simplified. It also eliminates the need for server management as servers are managed by the cloud providers (GlobalDots, 2019).

Azure Functions offer a complete package for FaaS, providing services from coding to continuous deployment, integration and monitoring of function usage. The functions are completely independent of each other, thereby reducing coupling in the application. The functions can easily be integrated with Azure Tables for efficient storage, retrieval and querying of data. JavaScript or Node.js was used to implement the functions required for the application. Each Azure Function contains a single main JavaScript function that takes a JSON object as input and returns a JSON object as output.

A Function App called VRPApp was created. Its status can be monitored using the link: `https://vrpapp.azurewebsites.net/`. Several functions performing a variety of tasks were constructed in the Function App; these are listed in Figure 5.2.

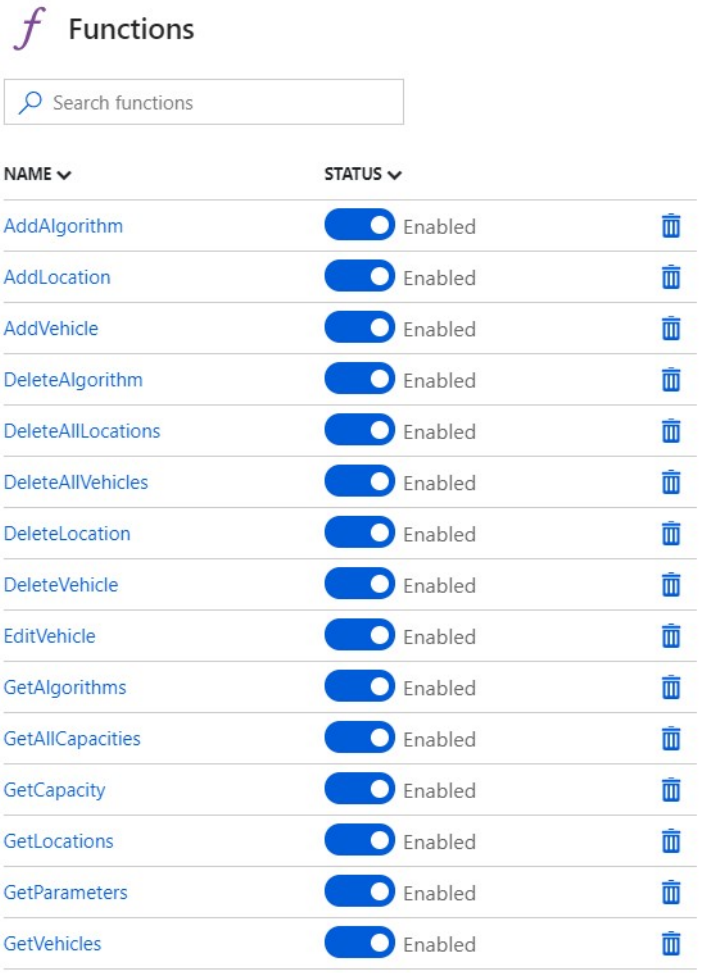


FIGURE 5.2: Azure Functions in Azure Function App

5.4 Client-side

The initial plan was to create a desktop application using Java. However, creating a web-based application is more effective since it can be accessed concurrently by multiple users globally. No additional software is required as all devices have pre-installed browsers. It is easier to update, customise and manage web applications compared to desktop applications.

HTML, CSS and JavaScript were used in conjunction since these enable creation of more interactive, elegant and richer user interfaces. This also simplified and sped up the development process.

HTML is lightweight and user-friendly, and most importantly displays changes and updates almost instantaneously. CSS is also lightweight and is used to style HTML components to give a fresher look to the interface. Creating a CSS file provides greater accessibility and allows easy maintenance, updating and reusability of class designs.

JavaScript is useful in the development of web applications because programs are embedded in web pages and executed on the user's browser instead of the website server, thus improving the overall user experience (Duan et al., 2005). JavaScript is an event-based programming language, where programs can be executed when specific events occur on user interactions. In addition, JavaScript offers a wide range of useful libraries. Another advantage was using jQuery and AJAX requests to communicate with the server-side Python.

The React library was used extensively as it is straightforward and has more support compared to other similar libraries. It boosts productivity and expedites testing and maintenance. React uses a Virtual DOM, which means that elements on the application are rendered faster compared to vanilla JavaScript code (AltexSoft, 2018).

5.4.1 Application Layout

The first step in the development of the client-side was designing the user interface as shown in Figure 5.3.

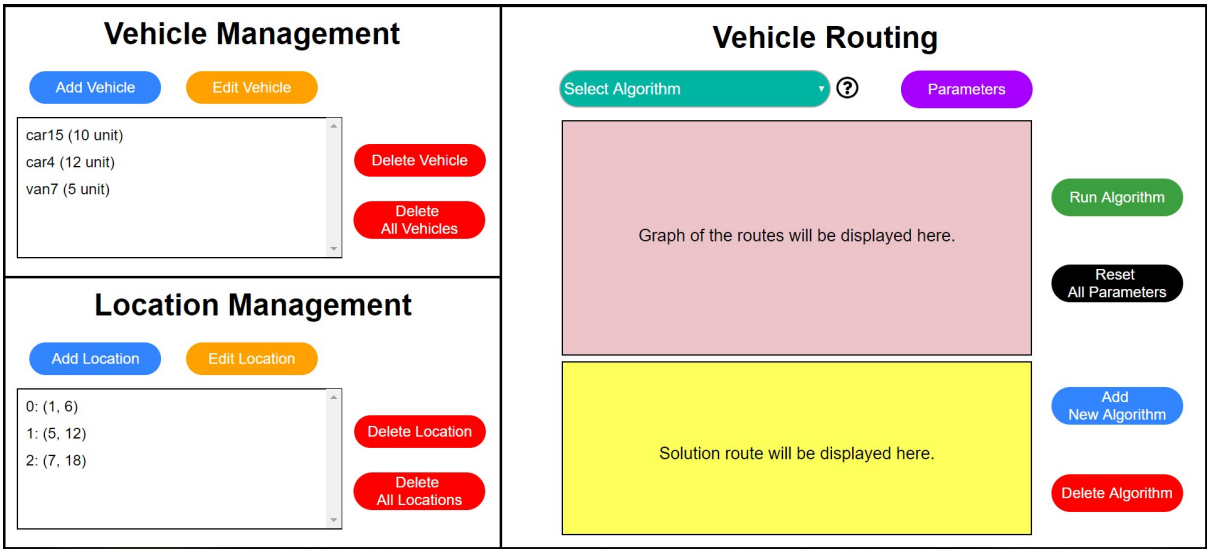


FIGURE 5.3: Application User Interface

5.4.2 Adding and Editing Vehicles

When the 'Add Vehicle' button is clicked, a dialog box (Figure 5.4) pops up, where the user enters the Vehicle Name and Capacity. On clicking the green button in the dialog box, the new vehicle is added to the Vehicles Azure Table.

Vehicle names and their capacities can also be edited. The user needs to select a vehicle to edit from the list before clicking the 'Edit Vehicle' button. The new Vehicle Name and Capacity can

be entered in the dialog box (Figure 5.4). On clicking 'Done', changes are made to the relevant entity, and the list is updated.

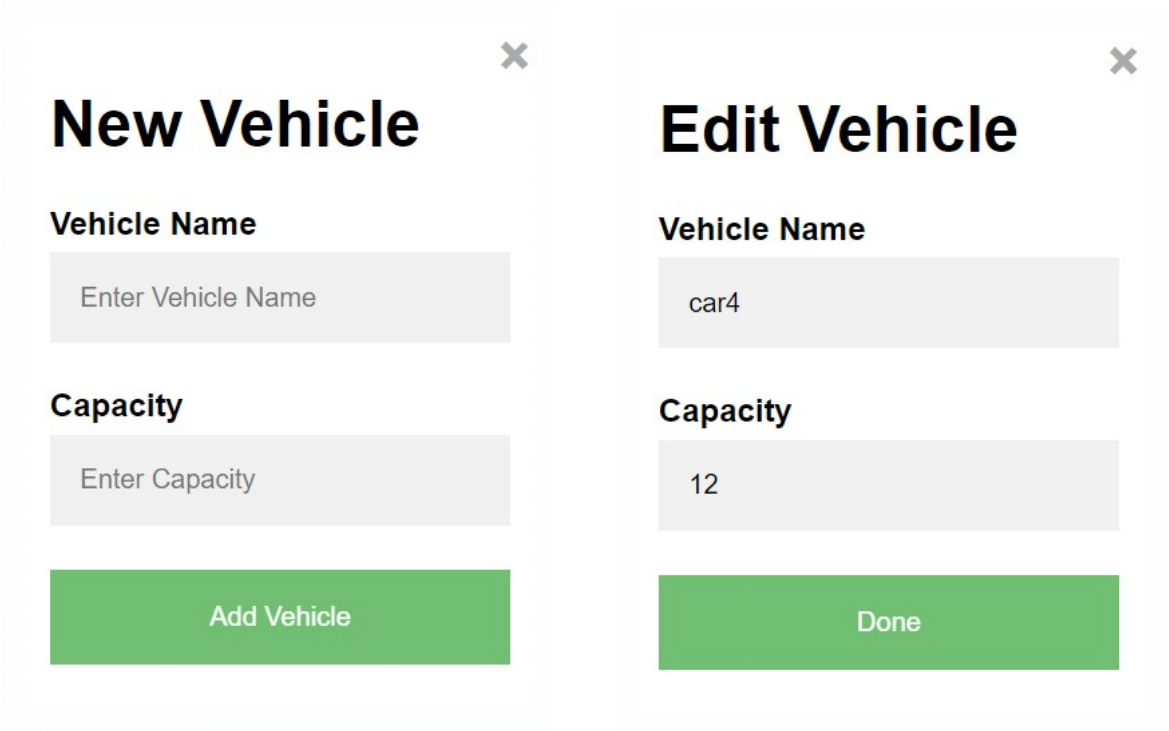


FIGURE 5.4: Vehicle Dialog Boxes

5.4.3 Adding and Editing Locations

When the 'Add Location' button is clicked, a dialog box (Figure 5.5) pops up where the user enters the X and Y Coordinates of the location. On clicking the green button in the dialog box, the new location is added to the Locations Azure Table.

Locations can also be edited. The user needs to select a location to edit from the list before clicking the 'Edit Location' button. The new X and Y coordinates can be entered in the dialog box (Figure 5.5). On clicking 'Done', changes are made to the relevant entity, and the list is updated.

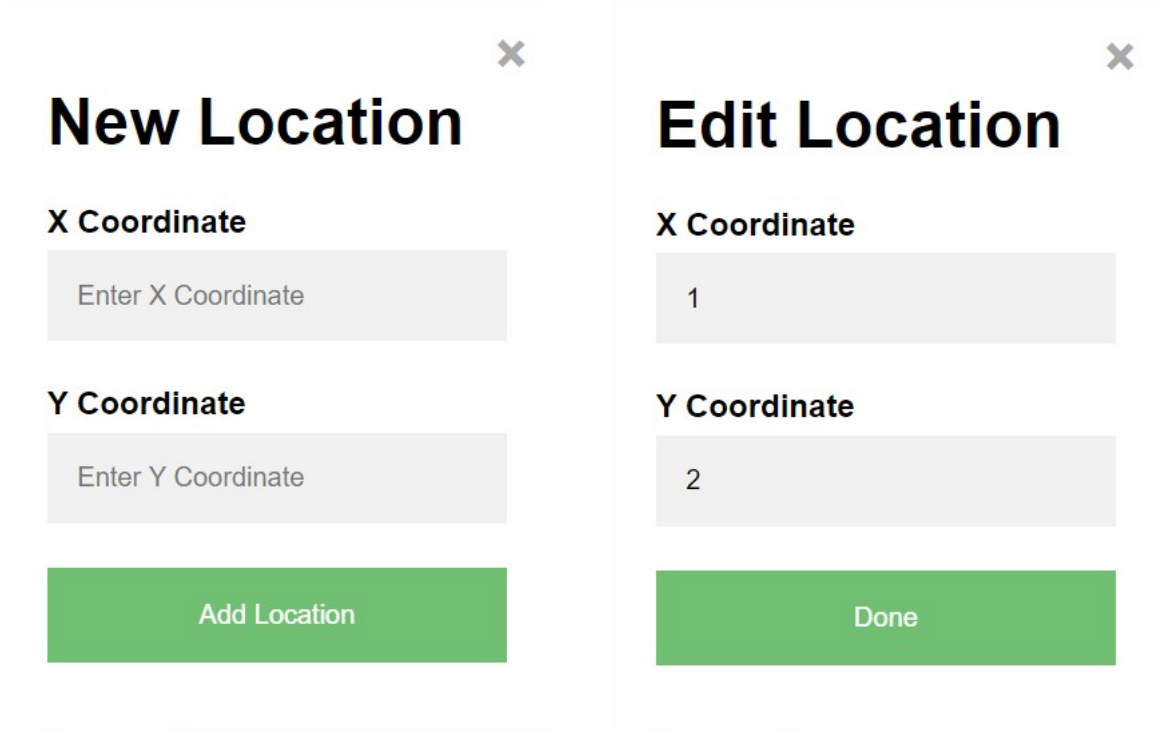


FIGURE 5.5: Location Dialog Boxes

5.4.4 Delete Vehicle/Location

To delete a vehicle or location, the user must select the relevant entity from the list and click the 'Delete Vehicle' or 'Delete Location' button as appropriate. The selected entity is then deleted from the respective Azure table, and the list on the interface is updated.

5.4.5 Delete All Vehicles/Locations

On clicking the 'Delete All Vehicles' button, all the vehicles are deleted from the Vehicles Azure Table and the list on the interface is updated to reflect this.

The 'Delete All Locations' button works similarly.

5.4.6 Select Algorithm

The 'Select Algorithm' dropdown list allows the user to select the required algorithm. The list includes nine default algorithms, as seen in Figure 5.6.

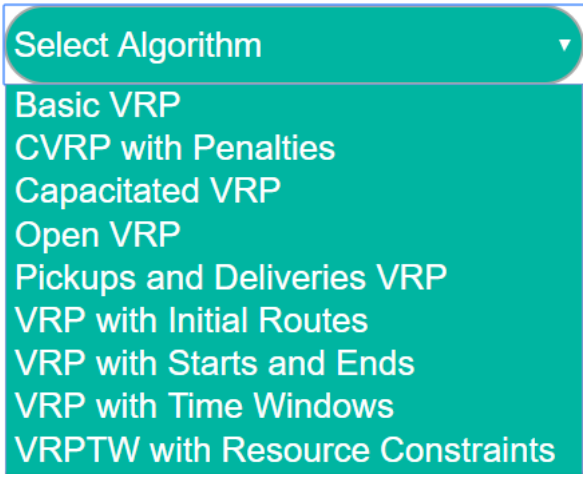
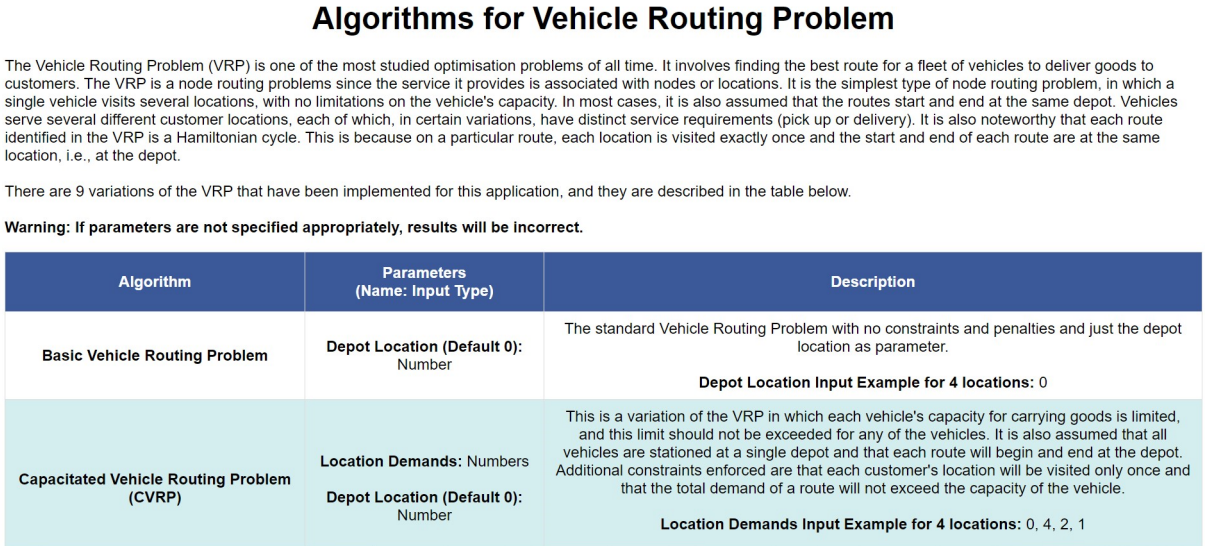


FIGURE 5.6: Select Algorithm Dropdown List

5.4.7 Help Website

On clicking the help icon next to the 'Select Algorithm' dropdown list, a new tab opens which directs the user to the 'Algorithms for Vehicle Routing Problem' help website. This website describes the nine default algorithms and their input parameters.

The help website can be seen in Figure 5.7.



Algorithms for Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is one of the most studied optimisation problems of all time. It involves finding the best route for a fleet of vehicles to deliver goods to customers. The VRP is a node routing problems since the service it provides is associated with nodes or locations. It is the simplest type of node routing problem, in which a single vehicle visits several locations, with no limitations on the vehicle's capacity. In most cases, it is also assumed that the routes start and end at the same depot. Vehicles serve several different customer locations, each of which, in certain variations, have distinct service requirements (pick up or delivery). It is also noteworthy that each route identified in the VRP is a Hamiltonian cycle. This is because on a particular route, each location is visited exactly once and the start and end of each route are at the same location, i.e., at the depot.

There are 9 variations of the VRP that have been implemented for this application, and they are described in the table below.

Warning: If parameters are not specified appropriately, results will be incorrect.

Algorithm	Parameters (Name: Input Type)	Description
Basic Vehicle Routing Problem	**Depot Location (Default 0):** Number	The standard Vehicle Routing Problem with no constraints and penalties and just the depot location as parameter. **Depot Location Input Example for 4 locations:** 0
Capacitated Vehicle Routing Problem (CVRP)	**Location Demands:** Numbers **Depot Location (Default 0):** Number	This is a variation of the VRP in which each vehicle's capacity for carrying goods is limited, and this limit should not be exceeded for any of the vehicles. It is also assumed that all vehicles are stationed at a single depot and that each route will begin and end at the depot. Additional constraints enforced are that each customer's location will be visited only once and that the total demand of a route will not exceed the capacity of the vehicle. **Location Demands Input Example for 4 locations:** 0, 4, 2, 1

FIGURE 5.7: Algorithm Help Page

5.4.8 Parameters

Certain algorithms require the user to specify parameters. To specify the parameters, the user must click the 'Parameters' button. A dialog box is displayed, where the required parameters can be entered. It is noteworthy that the parameters must be entered in the format specified on the help website; otherwise, the algorithms will return incorrect solutions and, in some cases, not even run.

The dialog box for the Capacitated VRP can be seen in Figure 5.8.

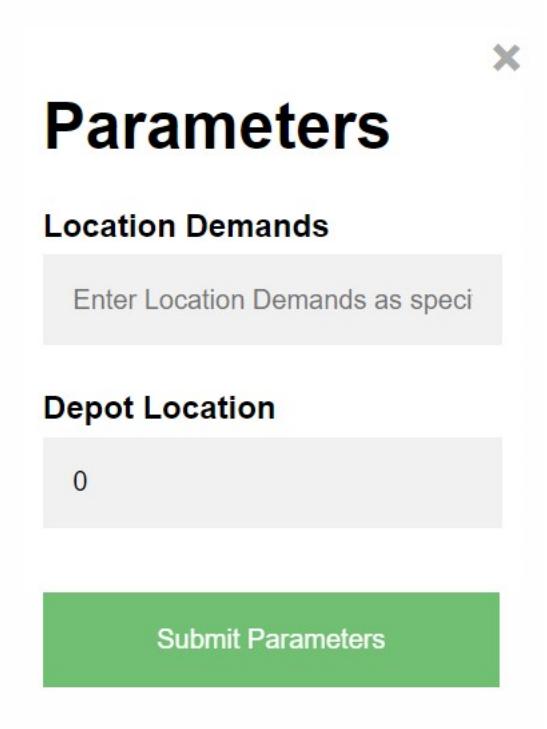


FIGURE 5.8: Parameters Dialog Box

Dialog boxes for other algorithms are similar.

5.4.9 Run Algorithm

On clicking the 'Run Algorithm' button, the application sends the parameters, vehicles, locations, and other relevant information to the appropriate Python server where the solution routes and graph is generated and displayed on the user interface.

The solution route and graph for running the Basic VRP can be seen in Figure 5.9.

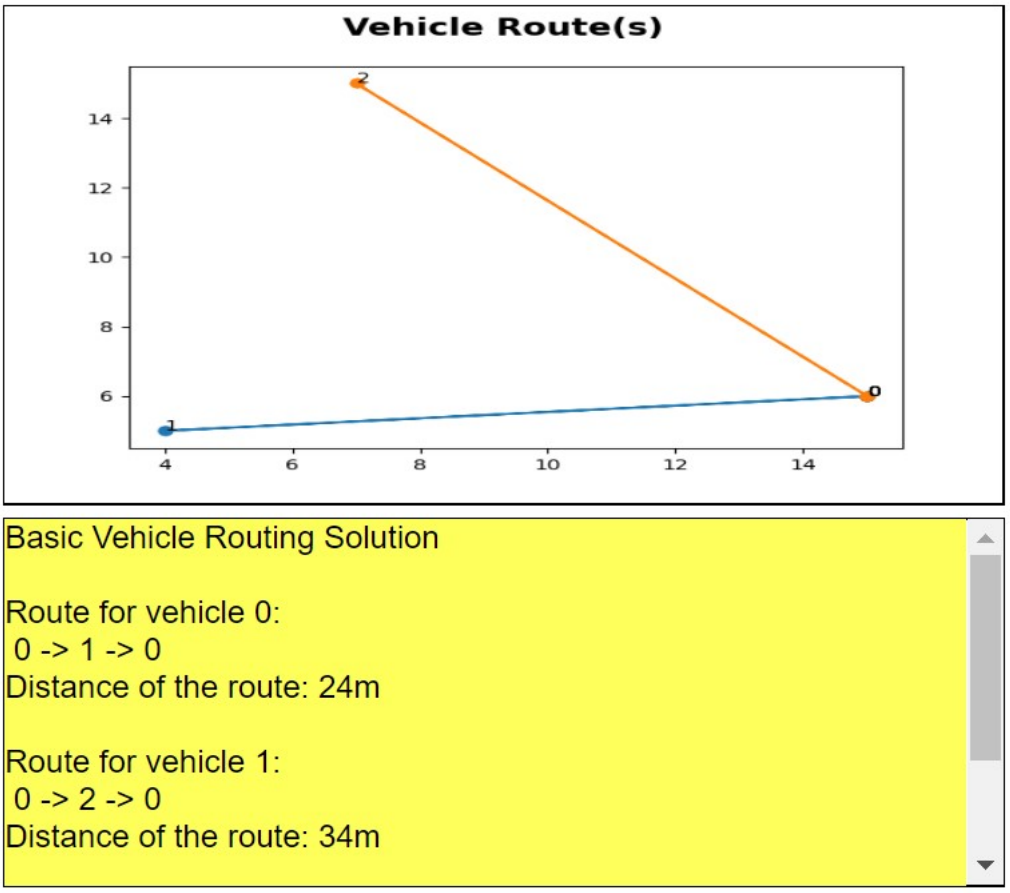


FIGURE 5.9: Solution for Basic VRP

5.4.10 Reset All Parameters

On clicking the 'Reset All Parameters' button, all the parameters are reset to their default values.

5.4.11 Add New Algorithm

When the 'Add New Algorithm' button is clicked, a dialog box is displayed which allows the user to enter the new Algorithm Name, its Parameters and the Python Code. The user can also upload a .txt or .py file which is then loaded into the Code text area of the dialog box. On clicking the 'Submit Algorithm' button, the new algorithm is added to the Algorithms Azure Table, and the interface is updated to indicate this. The code for newly added algorithms can be incorporated into the application at a later stage.

The Add Algorithm dialog box can be observed in Figure 5.10.

FIGURE 5.10: Add Algorithm Dialog Box

5.4.12 Delete Algorithm

To delete an algorithm, the user must select the algorithm from the 'Select Algorithm' dropdown list and click the 'Delete Algorithm' button. The selected algorithm is deleted, and the interface is refreshed. Only newly added algorithms can be deleted; the nine default algorithms cannot be deleted.

5.4.13 Communication with Azure Table Storage

Sending and receiving of data between the user interface and Azure Table Storage was achieved via the Fetch API in JavaScript and functions in the Azure Function App. The fetch function uses the function URL of the Azure Functions to send HTTP requests and JSON data to them. The Azure Functions add, delete, edit or retrieve data from the relevant table and return a JSON response. The response is then processed in JavaScript, and the user interface is notified and updated appropriately.

5.5 Server-side

The initial plan was to implement the algorithms using pseudocode and algorithms found online and in journals. Due to the lack of appropriate sources, the Google OR Tools library was used. Algorithms were implemented with help from the Vehicle Routing Problem section on the Google Developers website (Google Developers, 2020c). The nine algorithms implemented have already been explained in the Literature Review section. In this section, the functions and Python code written for them are described.

A Python module containing some common functions used by all algorithms was created. Some of these functions are listed below:

- **Manhattan Distance:** This function returns the Manhattan distance between two given location coordinates. For two coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance is given as:
$$Manhattan\ Distance = |x_1 - x_2| + |y_1 - y_2|$$
- **Distance Matrix:** A distance matrix displays the Manhattan distance between pairs of locations. This function takes a set of locations and creates a distance matrix, which is then given as input to the VRP algorithms. All algorithms use this matrix except the ones involving time windows.
- **Time Matrix:** A time matrix displays the Manhattan distance divided by the average speed of vehicles between pairs of locations. Algorithms involving time windows use this matrix.
- **Create Graph:** This function creates a graph displaying the routes followed by each vehicle.

5.5.1 Input Parameters for Algorithms

All algorithms take three inputs by default: the set of locations, the number of vehicles and the depot location. The Basic VRP and the Open VRP do not take any other parameters. However, other algorithms require additional parameters as listed below:

1. **Capacitated VRP and Capacitated VRP with Penalties:** Both these algorithms require the vehicle capacities and demands at each location as additional parameters.
2. **Pickups and Deliveries VRP:** This algorithm also takes a list of pairs of location indices. The first number of the pair indicates the location index of the pickup and the second number indicates the location index of the delivery.
3. **VRP with Time Windows:** This algorithm additionally requires a list of time window pairs and average speed of vehicles as input.
4. **VRPTW with Resource Constraints:** This algorithm takes a list of time window pairs, average speed of vehicles, vehicle load time, vehicle unload time and depot capacity as additional inputs.
5. **VRP with Starts and Ends:** This algorithm requires two lists of location indices as input, one for the start location and the other for the end location of each vehicle.
6. **VRP with Initial Routes:** In addition to the default parameters, this algorithm takes lists of routes as input, one route for each vehicle.

5.5.2 Communication with Application Interface

One of the main challenges encountered in the development of the application was sending data from the client-side JavaScript to the server-side Python. This was resolved by converting the Python algorithms into a series of web servers using Flask. Flask is a web microframework for Python that is lightweight, highly flexible and extensible. It includes HTTP request handling and is modular, which means multiple Flask servers can be run simultaneously. This was imperative since nine servers needed to run concurrently, one for each algorithm. Each algorithm runs on a unique server port and communicates with the client-side JavaScript using HTTP POST requests. When the user clicks the 'Run Algorithm' button on the application interface, the JavaScript using jQuery and AJAX sends the data in JSON form to the appropriate Python server. The algorithm on the server returns the solution routes and graph to JavaScript, which are rendered onto the user interface.

To run all nine servers conveniently, a Windows Batch File and a bash script were created, the former for running the servers on Windows and the latter for running on Unix based operating systems. Executing the appropriate file on the relevant operating system will run all the nine servers.

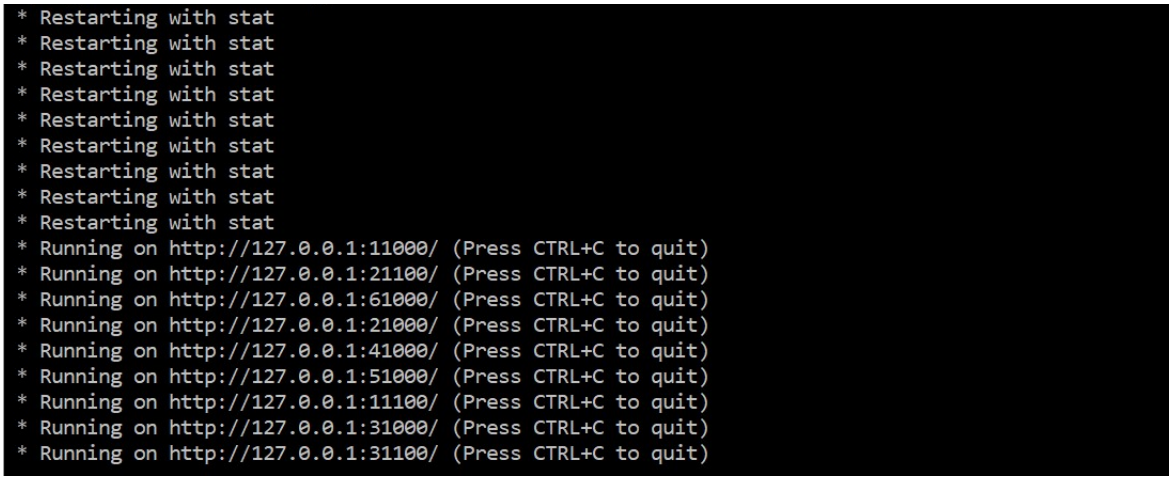


FIGURE 5.11: Python Flask Servers running on Windows

5.5.3 Displaying Graphs on the Interface

An issue encountered was displaying the graphs of the routes generated by the Python algorithms on the web application. This issue was resolved using the imgurpython library. The graph generated in Python can be uploaded to Imgur as an image. The link of the uploaded image is sent to the client-side JavaScript where the graph can then be displayed easily. Uploading the

graph image requires the username, password, client id and client secret, hence a new Imgur account was created to which these graphs are uploaded (Lucid Programming, 2017).

Authentication of the account and uploading of the graph was implemented using the selenium library. These processes occur using a headless browser, therefore the user needs to have either Firefox or Google Chrome installed for the graph to be displayed on the application.

5.6 Azure Web App

The client-side of the application was deployed on Microsoft Azure Web App. It enables smooth deployment and redeployment, which makes adding, deleting and debugging of functionality very easy. Microsoft takes care of server maintenance and ensures the application keeps running smoothly. Another advantage is that there is no need to obtain a domain name as Microsoft provides a URL after deployment to the Web App. Apps hosted on Azure are highly secure since it complies with strict security standards. It is also possible to gain further insight into resource usage, the app's throughput, response times, memory and CPU utilisation and error trends (Microsoft Azure, 2020).

The application can be accessed on this link: `https://vehicleroutingsolution.azurewebsites.net/`.

5.7 Static Help Website

An algorithm help page was created as well, which provides information about the default algorithms and their input parameters. A static website was created since the help page is noninteractive, and the content is constant. This involved uploading the HTML and CSS files to a Blob container on Azure. Blob storage is an object storage solution used to store massive amounts of data and files. It can serve documents, files and data directly to a browser, making it ideal for the deployment of a non-interactive website.

The help page can be accessed from this link: `https://vrpalgorithmhelptable.z33.web.core.windows.net/`.

5.8 Algorithms in Java for Evaluation

For evaluation and comparison of performance and efficiency of different variations of the VRP, the algorithms were implemented in Java using the Google OR Tools library.

Chapter 6

Testing Strategy

The testing phase includes verification and validation of the software. It is the process of scrutinising various features of the developed application, verifying that it meets the original requirements and fulfils the intended objective. Testing allows developers to debug and refine the application (Bassil, 2012).

6.1 Unit Testing

Individual units or components of an application are tested to verify that each unit performs as anticipated. A unit can refer to a particular program, function, etc. (Software Testing Fundamentals, 2019a).

Table 6.1 below and Figure C.1 in Appendix C describe the executed unit tests.

Test	Procedure	Expected Outcome	Result
Calculating the Manhattan Distance	The function is called with two locations as arguments.	The expected Manhattan distance is obtained.	Pass
Calculating the distance matrix	The function is called with a list of locations as an argument.	The expected distance matrix is returned.	Pass
Calculating the time matrix	The function is called with a list of locations and the average speed as arguments.	The resultant time matrix is as expected.	Pass
			Continued

Running the Basic VRP algorithm	The algorithm takes a list of locations, number of vehicles and depot location as arguments.	The string of solution routes returned is as expected.	Pass
Running the Capacitated VRP with insufficient vehicle capacities	The function takes the depot location, number of vehicles, and lists of locations, vehicle capacities and location demands as arguments.	The algorithm fails to find a solution and returns "No solution".	Pass
Incorrect values of time windows for the VRP with Time Windows	The function takes the depot location, number of vehicles, average speed, and lists of locations and time windows as arguments.	The algorithm fails to find a solution and returns "No solution".	Pass
Obtaining graph coordinates	The function takes a list of locations and a string of solution routes as arguments.	The graph coordinates obtained satisfy estimates.	Pass

Table 6.1: Unit Testing

Unit testing was also performed for the functions in the Azure Function App to ensure adding, editing and deleting of entities to the Azure Tables occurs correctly. Additionally, it was useful for testing the interactions between the Azure Functions and the JavaScript functions and between JavaScript functions and the Python Flask servers.

6.2 Integration Testing

Integration testing involves combining individual units and testing them as a group. It helps in determining how efficiently the components work together. This is because components might work properly individually, but integration into the main application can cause errors and bugs (Pearson, 2015).

Screenshots for the tests in Table 6.2 can be found in Appendix D.

Test	**Procedure**	**Expected Outcome**	**Result**
Add a new Vehicle with an existing name	Click the 'Add Vehicle' button. Enter a name already in use, enter any capacity and click the 'Add Vehicle' button in the dialog box.	An alert pops up "Vehicle with this name already exists. Please enter a different name."	Pass
Add a new Location with existing coordinates	Click the 'Add Location' button. Enter the X and Y coordinates of an existing location. Click the 'Add Location' button in the dialog box.	An alert pops up "Location has already been added. Please enter a different location."	Pass
Run Algorithm without Python Flask servers running	Select any algorithm from the dropdown list and click 'Run Algorithm'.	A message is displayed "The server is currently offline. Vehicles, Locations and Algorithms can still be added."	Pass
Run algorithm with insufficient vehicles	Select the 'Basic VRP' algorithm and click 'Run Algorithm'	An alert pops up "Please enter at least 1 vehicle to compute a route"	Pass
Run algorithm with insufficient locations	Select the 'Basic VRP' algorithm and click 'Run Algorithm'	An alert pops up "Please enter at least 2 locations to compute a route"	Pass
Run algorithm with insufficient parameters	Select the 'Capacitated VRP' algorithm and click 'Run Algorithm'.	An alert pops up "Please enter parameters"	Pass

Continued

Add a new Algorithm with an existing name	Click the 'Add New Algorithm' button. Enter an existing algorithm name, any parameters and code. Click 'Submit Algorithm'.	An alert pops up "Algorithm with this name already exists. Please enter a different name."	Pass

Table 6.2: Integration Testing

6.3 System Testing

In System Testing, the entire application is tested as a whole. Testing of the application is carried out to ensure it meets all the requirements established at the Requirements Analysis stage (Software Testing Fundamentals, 2019b).

Screenshots for the tests in Table 6.3 can be found in Appendix E.

Test	**Procedure**	**Expected Outcome**	**Result**
Add a new Vehicle	Click the 'Add Vehicle' button. Enter a name and capacity and click 'Add Vehicle' in the dialog box.	A new vehicle is created and displayed in the Vehicle Management section	Pass
Delete Vehicle	Select a Vehicle from the list and click the 'Delete Vehicle' button.	The vehicle is deleted, and the updated list is displayed.	Pass
Edit a Location	Select a Location from the list and click the 'Edit Location' button. Perform the required changes and click 'Done'.	The location is edited successfully, and the list is updated.	Pass
Delete All Locations	Click the 'Delete All Locations' button. An alert asking for confirmation pops up, click 'OK'.	All the locations are deleted, and the list is empty.	Pass
			Continued

Add a New Algorithm	Click the 'Add New Algorithm' button. Enter the name, parameters and code for the algorithm and click 'Submit Algorithm'.	The algorithm is created and added to the Select Algorithm dropdown list.	Pass
Delete Algorithm	Select an algorithm other than the default algorithms and click the 'Delete Algorithm' button.	The algorithm is deleted, and the Select Algorithm dropdown list is updated.	Pass
Run Algorithm	Ensure the Python Flask servers are running. Select one of the default algorithms. Click the 'Parameters' button and enter the relevant parameters. Click 'Submit Parameters' followed by 'Run Algorithm'.	If a solution exists, the solution routes and a graph of the routes are displayed on the interface.	Pass

Table 6.3: System Testing

Chapter 7

Project Management

Project management and planning was an imperative part of the project. It helped to manage the workload and time more effectively, so I was able to accomplish all the goals specified at the start of the project. This section contains details about time management, incremental planning and version control for the application.

7.1 Gantt Chart

I used Gantt charts for planning and scheduling of tasks and tracking the progress of the project. The first Gantt chart (Figure 7.1) displays a planned, approximate timeline for the completion of tasks. Initially, I had also planned to perform validation and verification using Event-B models and use PDDL to search for optimal routes. However, this was altered due to the circumstances arising from the COVID-19 outbreak. The second Gantt chart (Figure 7.2) illustrates the actual timeline representing the real progress made.

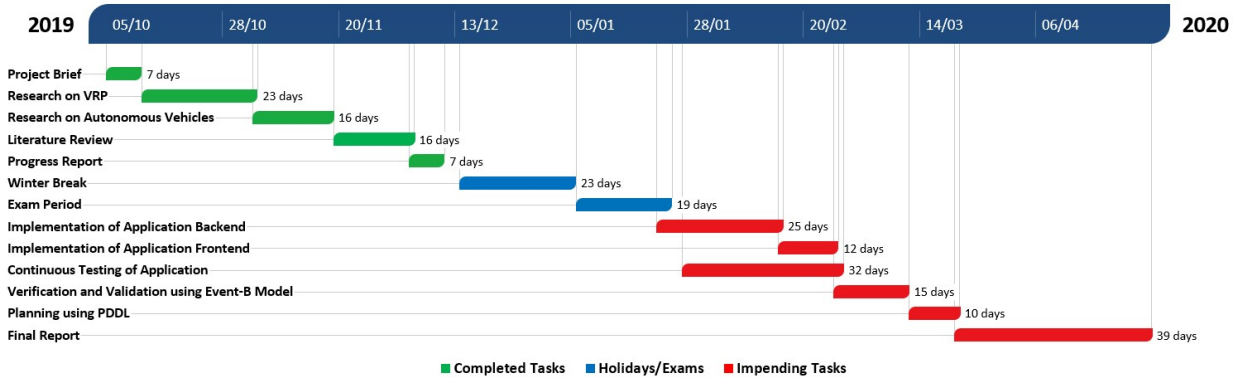


FIGURE 7.1: Gantt Chart representing Planned Project Progress

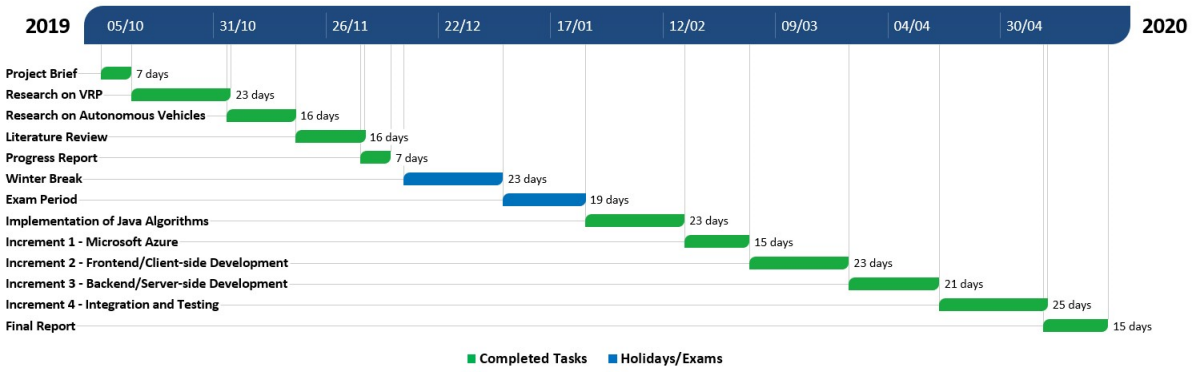


FIGURE 7.2: Updated Gantt Chart representing Actual Project Progress

7.2 Sprint Plans

The sprint planning aspect of the Agile methodology was utilised for the planning of tasks. Incremental sprint plans were created to track progress and identify completed tasks and the ones remaining at each stage. The sprint plans were revised periodically based on the progress and requirement of additional features. Trello was used due to its simplistic layout, and my experience of using it. The sprint plans are given in Figure 7.3.

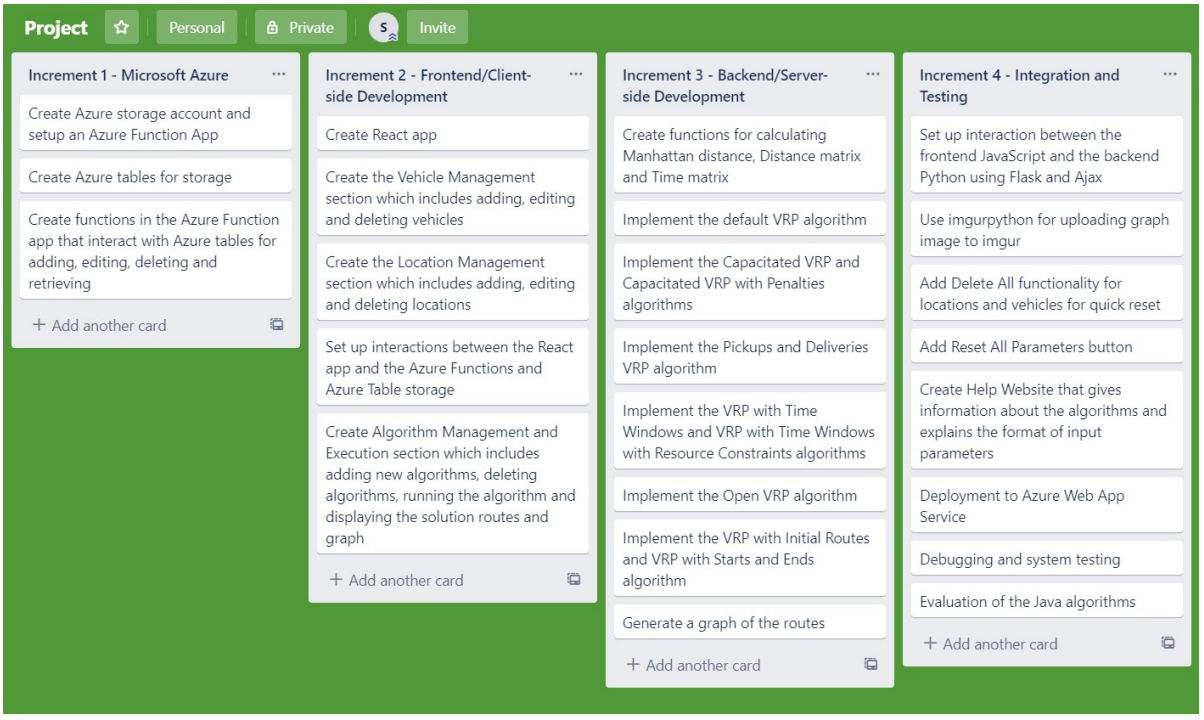


FIGURE 7.3: Sprint Plans

7.3 GitHub

A private GitHub repository was used for version control to manage the code and maintain online backups for recovery in case of malfunctions. This also enabled me to track changes and revert to a previous version of the application in extreme circumstances.

7.4 Risk Analysis

Risk analysis is the process of identifying factors that may affect the project, quantifying them and determining mitigation strategies to overcome them. This section analyses the potential risks and subsequent mitigation techniques involved in the development of this project.

Probability is the likelihood of some damage or risk occurring. This can vary between 1 and 5 with 1 being the least probable, and 5 the most probable. Severity is the extent of damage that a risk can cause and can range from 1 to 5, with 1 being the least severe, and 5 the most severe. Risk Exposure is calculated using the formula Risk Exposure = Probability × Severity.

Risk	**Probability (0-5)**	**Severity (0-5)**	**Risk Exposure**	**Mitigation**
Inability to work due to illness, accident, etc.	1	3	3	Work ahead of schedule. Reduce scope of the project (slightly less functionality)
Inefficiency in using React due to inexperience	2	1	2	Use vanilla HTML/CSS and JavaScript instead
Damage caused to computer	1	5	5	Recover files from backups and use an alternate computer
Considerable amount of work from other modules	3	2	6	Proper planning and time management to ensure all work is completed
				Continued

Obstacles in implementing certain features of the system	2	3	6	Attempt to overcome the issue. If this proves too challenging, exclude the feature from the system
Changing project specifications	2	1	2	Create a web-based application instead of an offline application for better usability
Loss of work	1	5	5	Maintain backups online on GitHub and offline on an external hard disk
Failure to implement specific tasks such as route planning using PDDL in the project	2	1	2	Prioritise tasks that are more necessary and implement them first
Bugs in the system	2	4	8	Rigorous testing of code. Recover from backups created in extreme cases

Table 7.1: Risk Analysis

7.5 Supervisor Meetings

Meetings with my supervisor were scheduled every 1-2 weeks, and were extremely valuable. Even during university closure due to the COVID-19 outbreak, we had regular online meetings using Microsoft Teams, where we discussed the progress made on the project. We agreed to omit Event-B modelling and PDDL planning from the project due to the prevailing circumstances.

Chapter 8

Critical Evaluation

Once the development and testing of the application are completed, the application needs to be evaluated. This section examines the performance of the application as well as the efficiency, scalability and effects of constraints on the VRP.

8.1 Evaluation of the Application

Overall, the implementation of the application has been a huge success since all the key project goals set at the beginning were achieved. The application allows users to run different variations of the VRP based on their requirements. Users can add vehicles and locations on which the algorithms will run. Certain algorithms require additional parameters that must be specified. Users are able to add new algorithms which can be incorporated into the application conveniently at a later stage. Once the user initiates the running of an algorithm, the algorithm calculates the routes that are as close to optimal as possible for the given input. The solution routes and a graph illustrating the route taken by each vehicle are displayed on the application interface. A solution is obtained as long as valid parameters are entered. Vehicles and locations can be edited and deleted quickly with the edit and delete features. The delete all and reset all parameters functionality allows the user to effortlessly reset all the input parameters and run the algorithms with fresh input.

8.2 Evaluation of the VRP Algorithms

This section analyses the performance of the various VRP algorithms by varying the problem size. The evaluation criteria are different for each algorithm and are displayed in the following tables. In all cases, the execution times are taken as averages of 5 executions.

8.2.1 Evaluation of the Basic VRP

Evaluating the Basic VRP requires the number of vehicles to be kept constant. For the execution times in Table 8.1 and Figure 8.1, the number of vehicles is considered as 3.

Number of Locations	Execution Time (in ms)
5	87
20	114
30	208
40	371
50	573
70	1326
80	3192
100	5856
130	11390
150	20039

Table 8.1: Basic VRP with variable number of locations

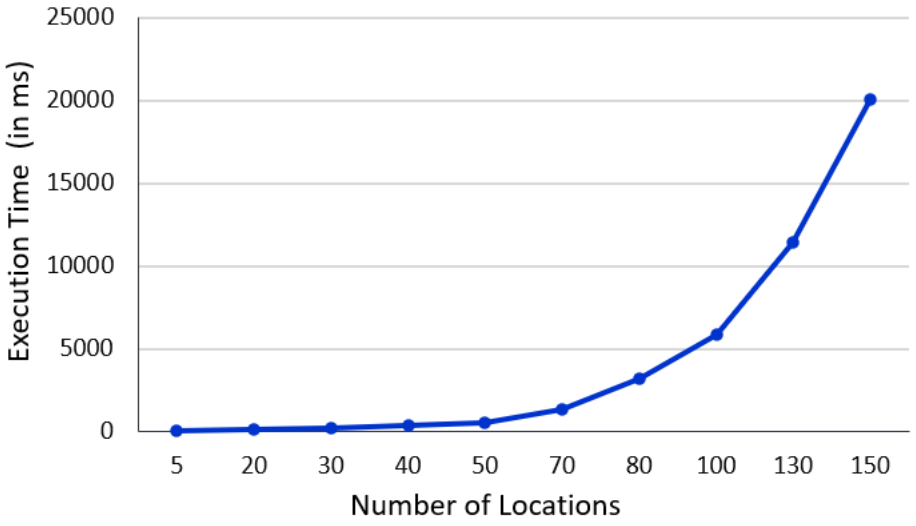


FIGURE 8.1: Graph for Basic VRP with variable number of locations

Keeping the number of vehicles constant, as the number of locations increase the algorithm takes relatively longer to find a solution. The algorithm is quite efficient since it can find a solution for up to 100 locations in less than 6 seconds.

For evaluation of the Basic VRP, the number of locations can also be kept constant. For the data in Table 8.2 and Figure 8.2, the number of locations is considered as 100.

Number of Vehicles	**Execution Time (in ms)**
5	3749
10	3756
15	3890
20	3903
25	3962
30	3979

Table 8.2: Basic VRP with variable number of vehicles

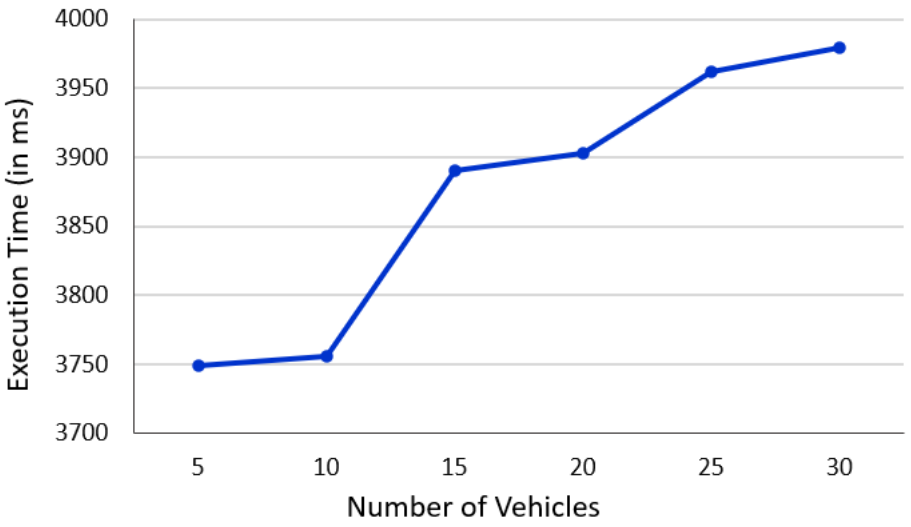


FIGURE 8.2: Graph for Basic VRP with variable number of vehicles

If the number of locations is kept constant, increasing the number of vehicles has a marginal effect on algorithm performance. There is not much difference between the timings for finding a solution for 5 vehicles and for 30 vehicles. Another interesting point is that several vehicles were unused in executions involving more than 10 vehicles.

Evaluation of the Open VRP, VRP with Initial Routes and the VRP with Starts and Ends show similar trends.

Since varying the number of vehicles has no significant effect on the performance of the algorithm, therefore, the data for this evaluation criteria has not been included for other algorithms.

8.2.2 Evaluation of the Capacitated VRP

For evaluating the Capacitated VRP, the number of vehicles and the vehicle capacities is kept constant. For the statistics in Table 8.3 and Figure 8.3, the number of vehicles is taken as 3, and the vehicle capacities as 100, 150 and 200, respectively.

Number of Locations	Execution Time (in ms)
5	88
20	93
30	96
40	104
50	118
70	145
80	206
100	297
130	431
150	591

Table 8.3: Capacitated VRP

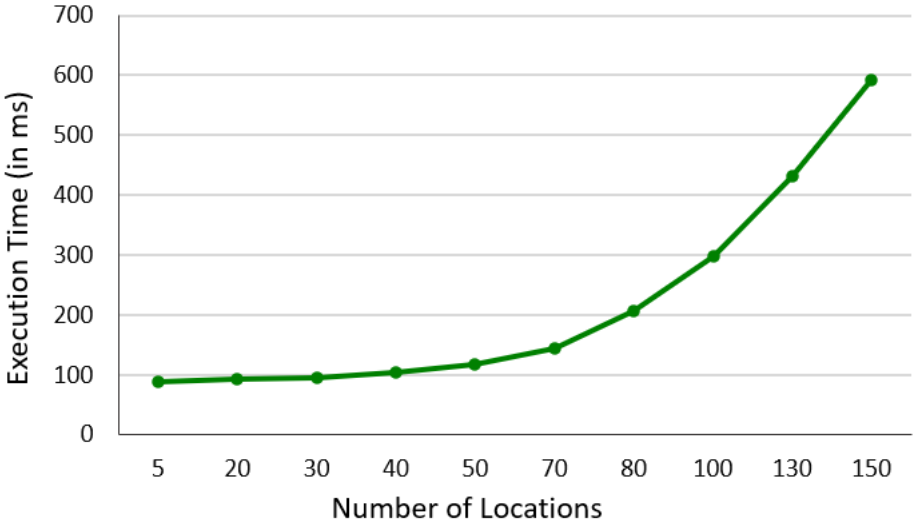


FIGURE 8.3: Graph for Capacitated VRP

The additional constraint of vehicle capacities evidently improves the performance of the algorithm compared to the Basic VRP. Even for a large number of locations, the algorithm is able to find a solution in approximately half a second. However, it is noteworthy that if the location demands exceed the combined capacities of the vehicles, the algorithm is not able to find a solution.

The CVRP with Penalties has similar performance.

8.2.3 Evaluation of the Pickups and Deliveries VRP

For evaluating the Pickups and Deliveries VRP, the number of vehicles and the number of pickups and deliveries is kept constant and taken as 3 for the calculations in Table 8.4 and Figure 8.4. However, the location indices in the pickups and deliveries vary.

Number of Locations	Execution Time (in ms)
5	92
20	111
30	182
40	465
50	661
70	1871
80	2473
100	6579
130	15577
150	23645

Table 8.4: Pickups and Deliveries VRP

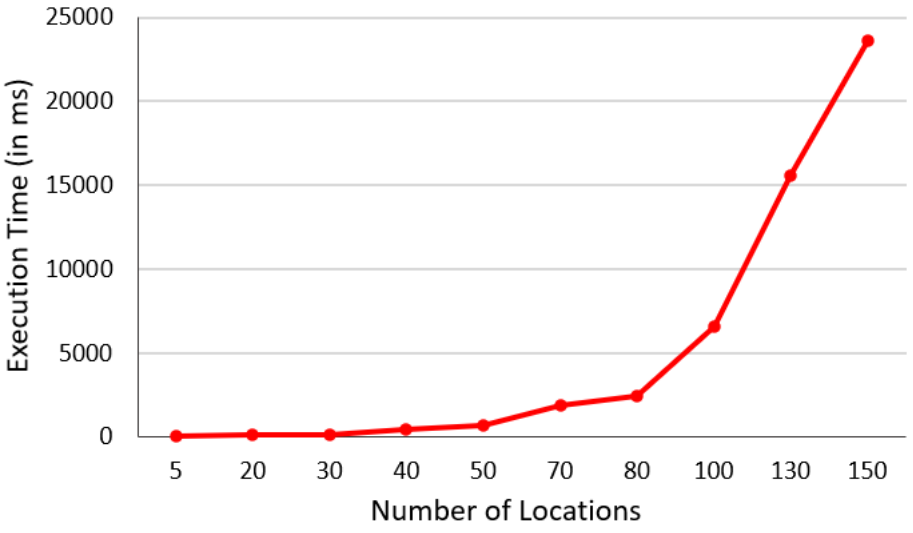


FIGURE 8.4: Graph for Pickups and Deliveries VRP

This algorithm is much slower compared to the Basic VRP and the Capacitated VRP. For 150 locations, the algorithm takes almost 23.6 seconds. This is significantly longer than the time taken by the Capacitated VRP (5.9 seconds) and the Basic VRP (20.3 seconds). The algorithm often fails to find a solution for the input of particular pickups and deliveries.

8.2.4 Evaluation of the VRP with Time Windows

For evaluation of this algorithm, the number of vehicles is kept constant and considered as 3 for the data in Table 8.5 and Figure 8.5. The number of time windows increases with the number of locations, as one time window needs to be specified for each location.

Number of Locations	Execution Time (in ms)
5	85
7	89
10	91
15	92
17	94

Table 8.5: VRP with Time Windows

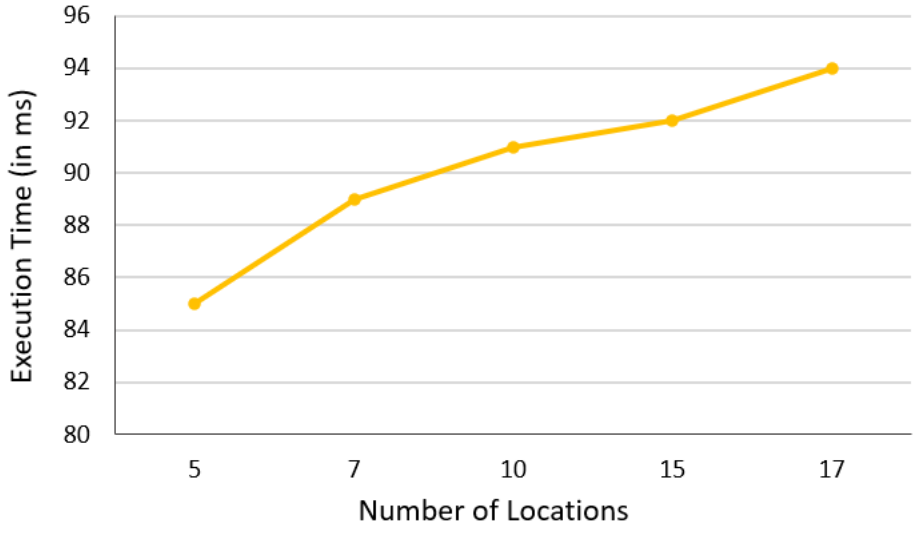


FIGURE 8.5: Graph for VRP with Time Windows

The algorithm performance is adept, and a solution can be found promptly for most cases. However, evaluation for time windows for greater than 17 locations becomes extremely difficult as no solution is found in many cases. Hence extending the number of locations for this algorithm is more complex compared to the other problems.

The evaluation of the VRPTW with Resource Constraints is similar.

8.2.5 Performance Comparison of Various Algorithms

Graphs comparing the performance of the various algorithms are given in Figure 8.6 and Figure 8.7.

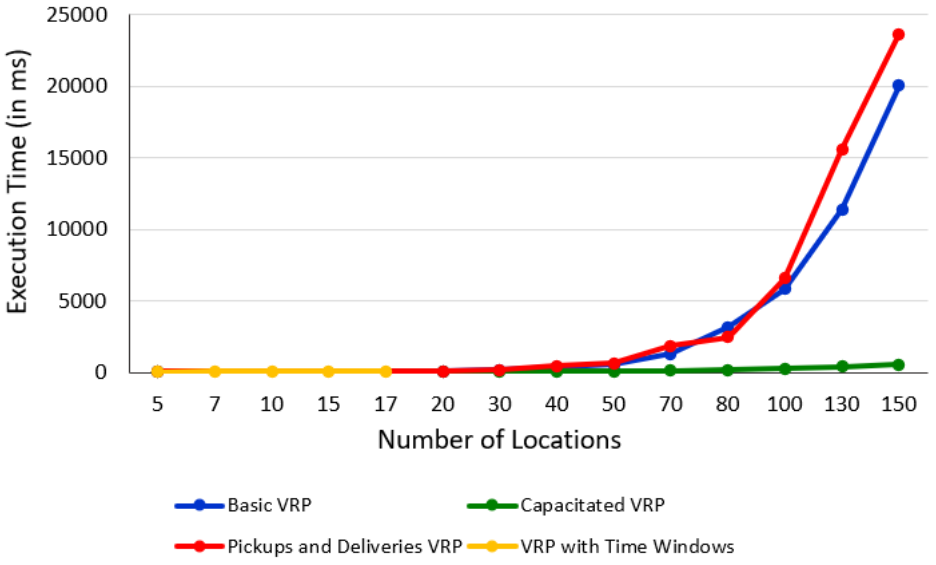


FIGURE 8.6: Linear Graph comparing Algorithm Performance

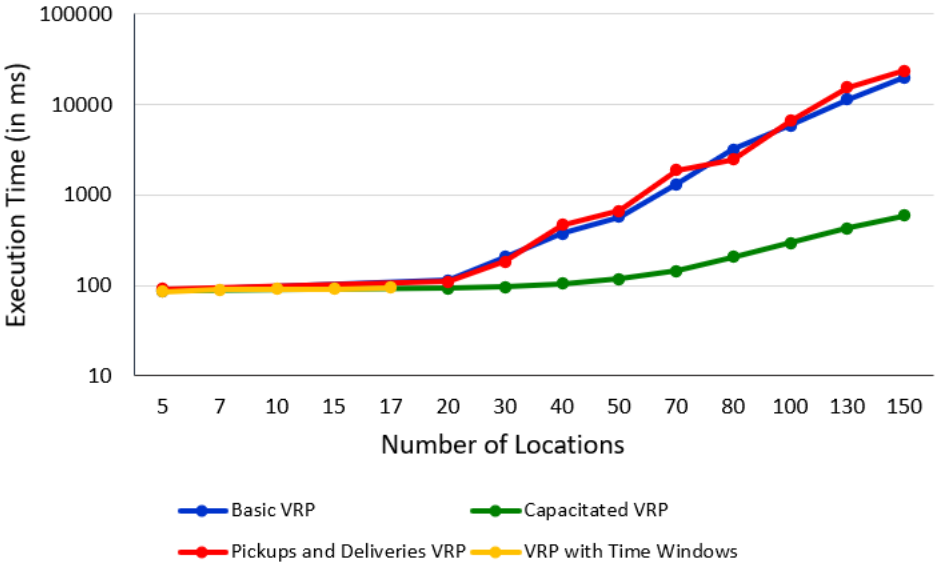


FIGURE 8.7: Logarithmic Graph comparing Algorithm Performance

It is evident from the graph that certain algorithms perform better than others. Applying specific constraints is more favourable than others. The Capacitated VRP is the most efficient,

whereas the Pickups and Deliveries VRP is the least efficient. Even though the VRP with Time Windows is quite efficient, it is not scalable since the algorithm fails to find a solution for more than 17 locations. This is because specifying the time windows becomes extremely difficult for a large number of locations.

Increasing the number of locations while keeping the number of vehicles constant has a significant impact on the performance of the algorithms.

Chapter 9

Conclusion and Future Work

The developed application simulates and analyses the Vehicle Routing Problem and its popular variations. It serves as a prototype solution for generating routes by running the appropriate algorithms. The application operates smoothly, is user-friendly and scalable as new algorithms can be implemented at any stage with ease.

Analysis of the various algorithms offers an insight into the field of route planning. The project studies the performance of the algorithms by varying the number of vehicles, locations and parameters such as vehicle capacities, time windows, etc. The algorithms are efficient and identify solutions for most inputs. The performance analysis and evaluation highlight the benefits of applying certain constraints and using specific algorithms over others. The evaluation shows that there is a considerable effect on the efficiency of the algorithms with an increase in the number of locations.

VRP is an NP-hard problem with no optimal solutions. Hence the algorithms search a small solution space and find a solution that is close to an optimal solution. Applying certain constraints makes it difficult for the algorithms to provide solutions for a substantial number of locations.

There are some areas where the future extension of the application can be accomplished:

- **Vehicle Parameters:** Incorporate more real-life parameters such as vehicle fuel/battery capacities, individual speeds, unavailability of customers at locations, etc.
- **Google Distance Matrix API:** Extend the application to include the Google Distance Matrix API, which would enable route planning of real-world locations.
- **Eliminate Python Flask Severs:** Combine the Python server-side, and client-side React app, thus eliminating the need to run the Python Flask servers separately.

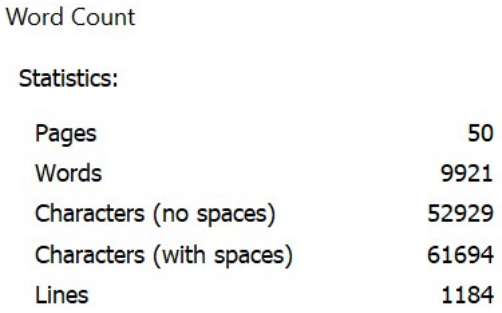
Autonomous vehicles are the future of transportation and have numerous benefits. Since they are entirely software-controlled, therefore, the developed VRP application can be readily integrated to function with autonomous vehicles.

Overall, I am delighted with the results of this project and trust that I have been able to offer an intriguing analysis of the Vehicle Routing Problem, its variations and the benefits of efficient route planning to organisations and the environment.

Appendix A

Word Count

The word count of the body calculated by Foxit is given below.



Word Count

Statistics:

Pages	50
Words	9921
Characters (no spaces)	52929
Characters (with spaces)	61694
Lines	1184

FIGURE A.1: Report Word Count

Appendix B

Project Brief

Simulation of Vehicle Routing Problem

Suyash Dubey (sdd1n17)

Supervisor: Dr Thai Son Hoang (tsh2n14)

October 2019

B.1 Problem

Vehicle Routing Problems (VRP) are combinatorial optimisation problems that deal with the transportation of people or goods between depots and customers. The goal is to identify the optimal set of routes for multiple vehicles while minimising operating costs and maximising customer satisfaction. One of the assumptions made is that the vehicles deliver goods from a single central depot to customers that have placed relevant orders. Finding an optimal solution to VRPs is NP-hard. Therefore in general, heuristics and metaheuristics are used to solve VRPs in practical applications due to the scope of the problem.

B.2 Goals

The goal of the project is to solve the VRP using a convoy of vehicles to simulate the route and delivery system. The vehicles will operate as autonomous systems, without any direct human interaction. Human involvement will only be required if a problem arises. The delivery route can either be established by the user or generated by various heuristic algorithms.
The application will be written using Java, where the convoy of vehicles and simulation of the delivery system will be programmed and displayed.

B.3 Scope

Programming vehicles to perform the deliveries is out of the scope of this project and hence the project will be completely software-based. The aim is to develop an application that is simple to use and scalable so that it can be used easily to plan delivery routes and can also be extended to other similar routing problems. One possible extension can be creating an Event-B model and using ProB simulation to ensure that the vehicles and routes are selected appropriately and ensure the safety of the vehicles.

Appendix C

Unit Testing Screenshot

The unit tests performed can be seen below.

```
test_basic_vrp (__main__.TestVRP) ... ok
test_capacitated_vrp_fail (__main__.TestVRP) ... ok
test_distance_matrix (__main__.TestVRP) ... ok
test_graph_coordinates (__main__.TestVRP) ... ok
test_manhattan_distance (__main__.TestVRP) ... ok
test_time_matrix (__main__.TestVRP) ... ok
test_time_windows_no_solution (__main__.TestVRP) ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.019s

OK
```

Figure C.1: Unit Tests

Appendix D

Integration Testing Screenshots

Screenshots for the integration tests described in Section 6.2 are shown below.

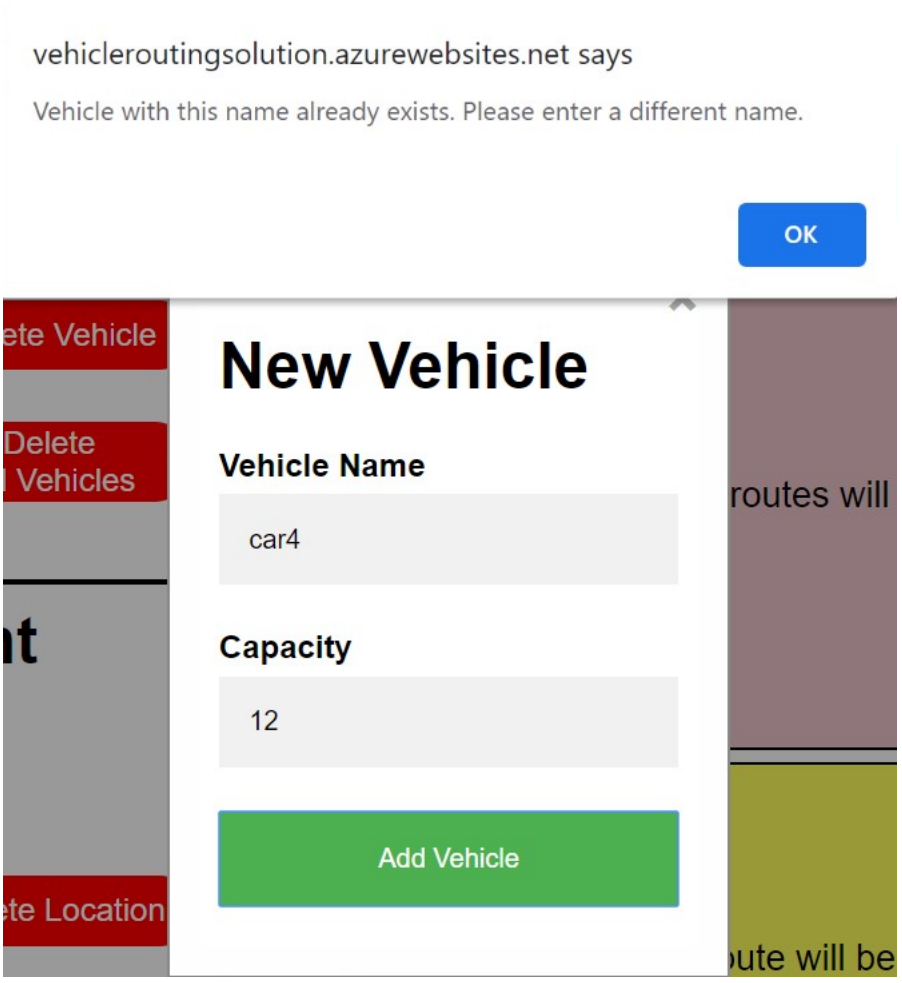


FIGURE D.1: Add a new Vehicle with an existing name

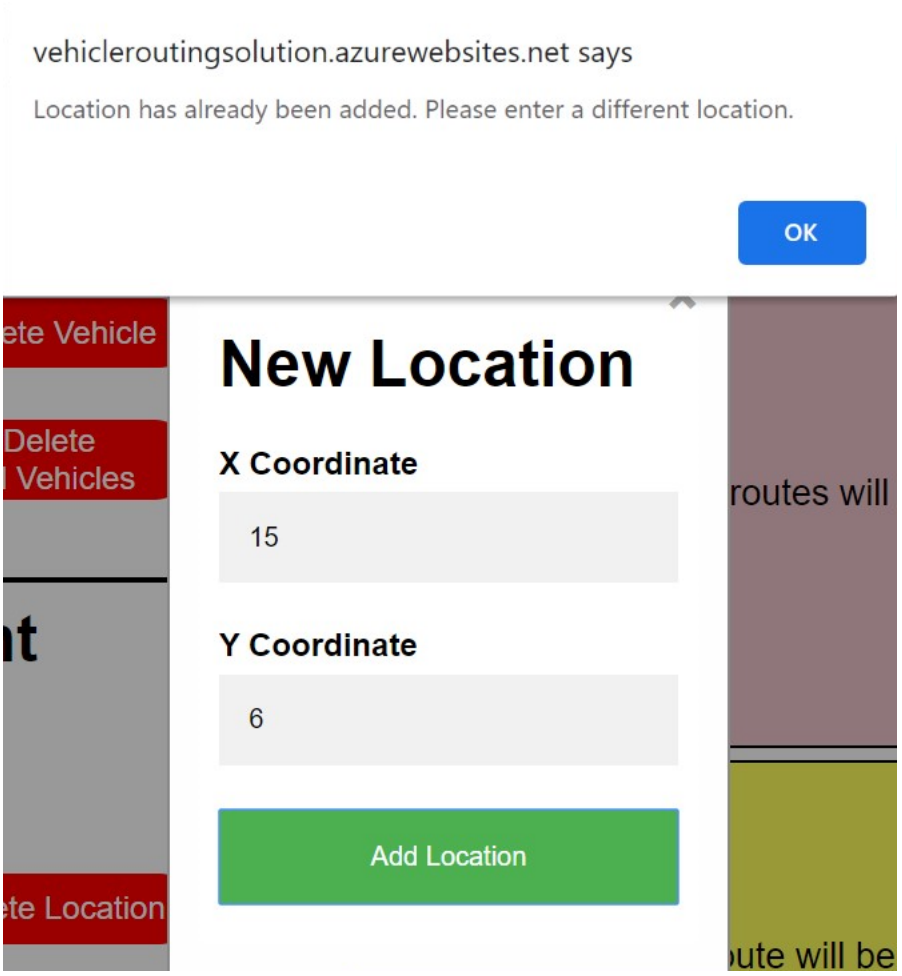


FIGURE D.2: Add a new Location with existing coordinates

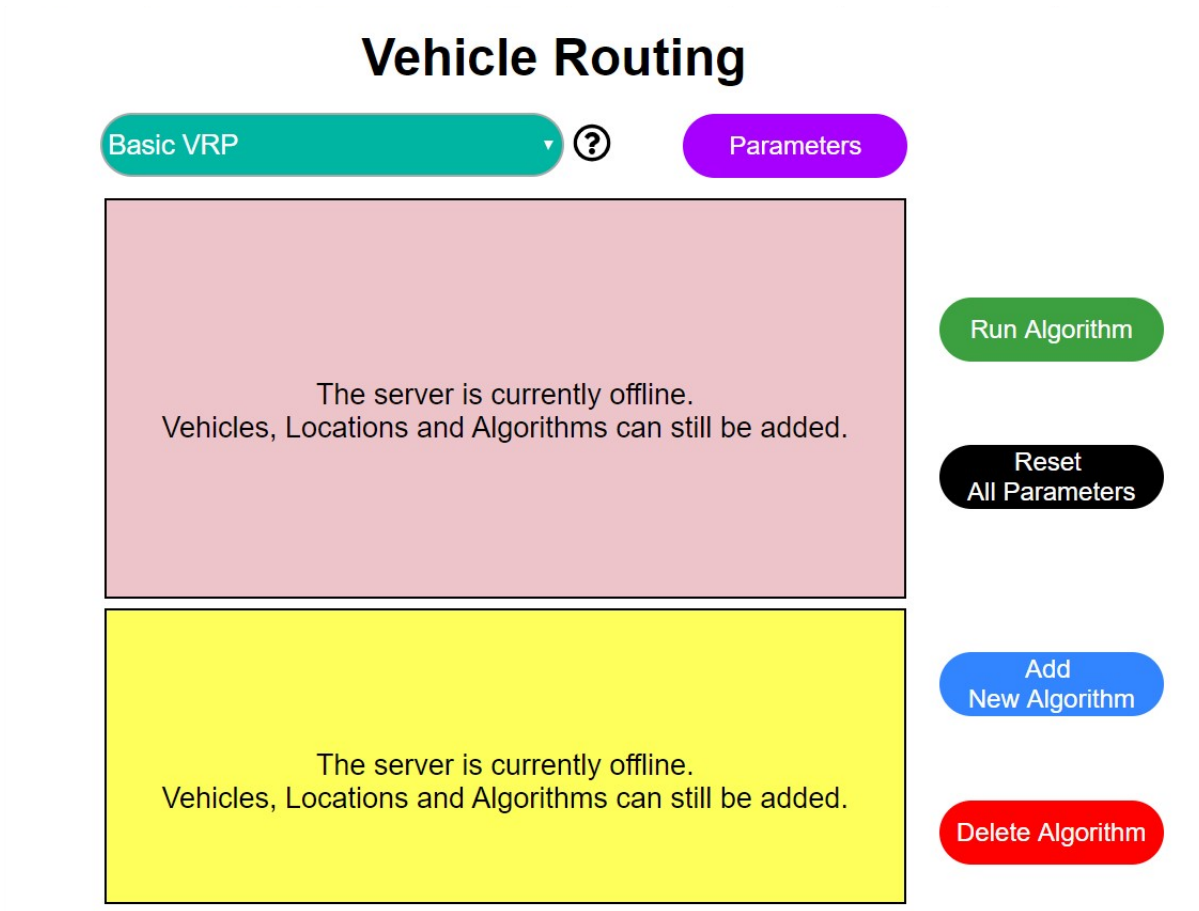


FIGURE D.3: Run Algorithm without Python Flask servers running

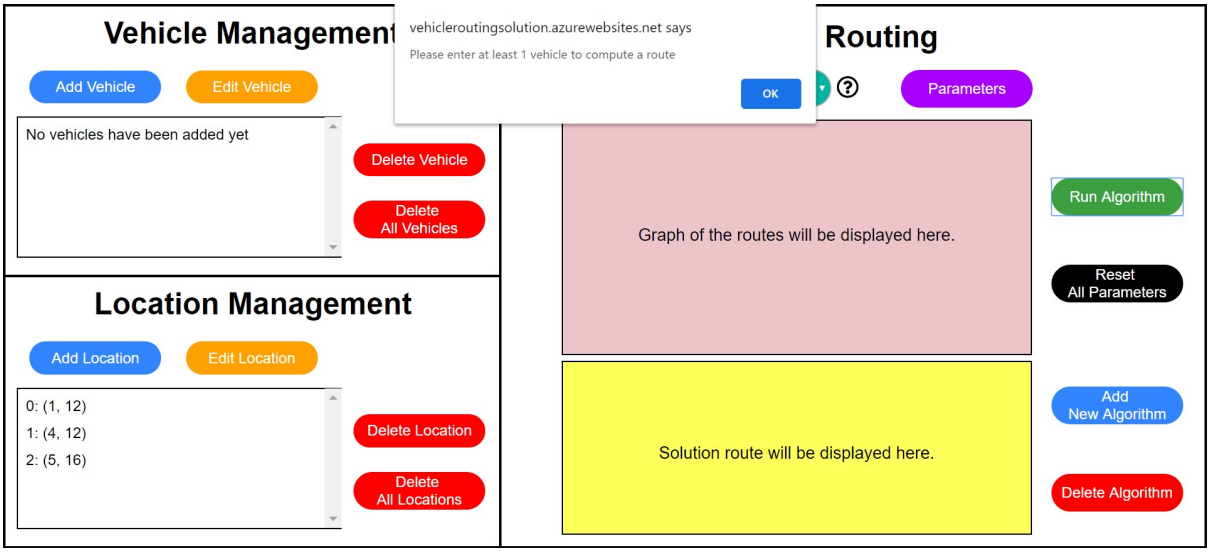


FIGURE D.4: Run algorithm with insufficient vehicles

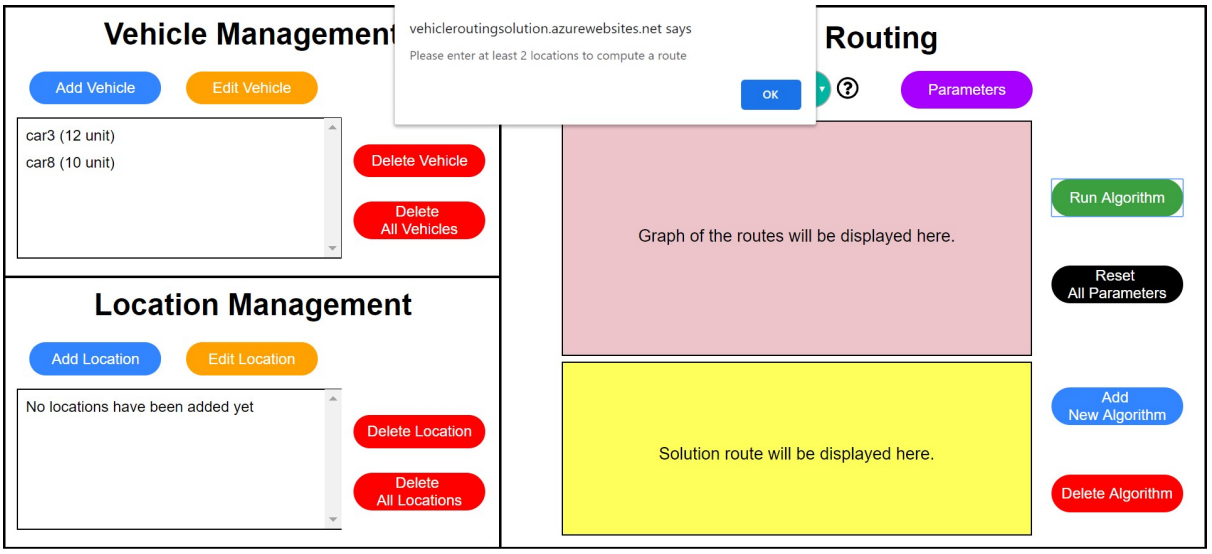


FIGURE D.5: Run algorithm with insufficient locations

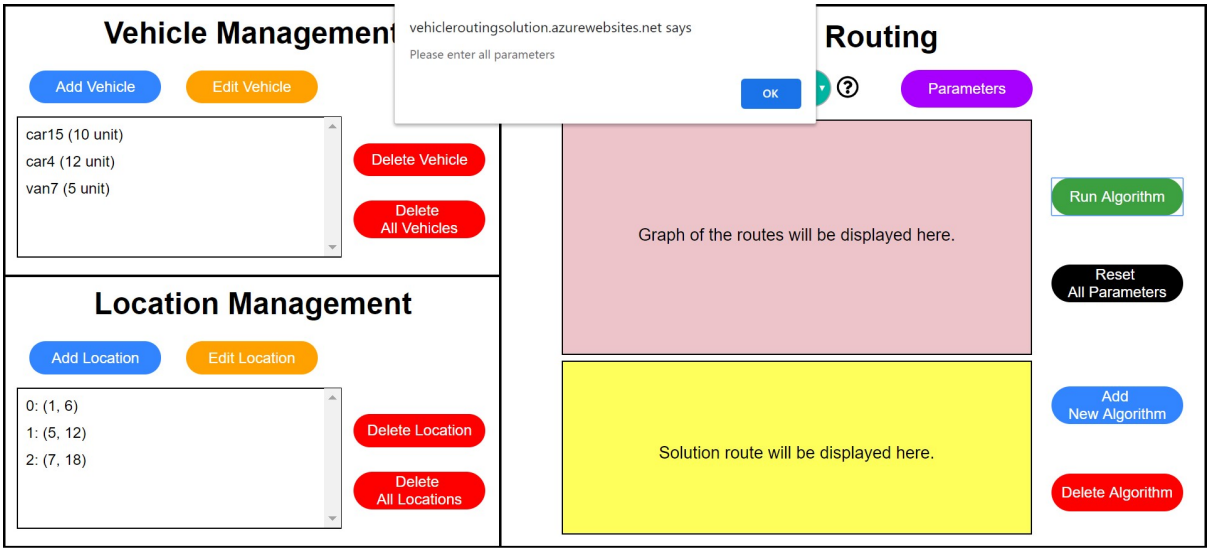


FIGURE D.6: Run algorithm with insufficient parameters

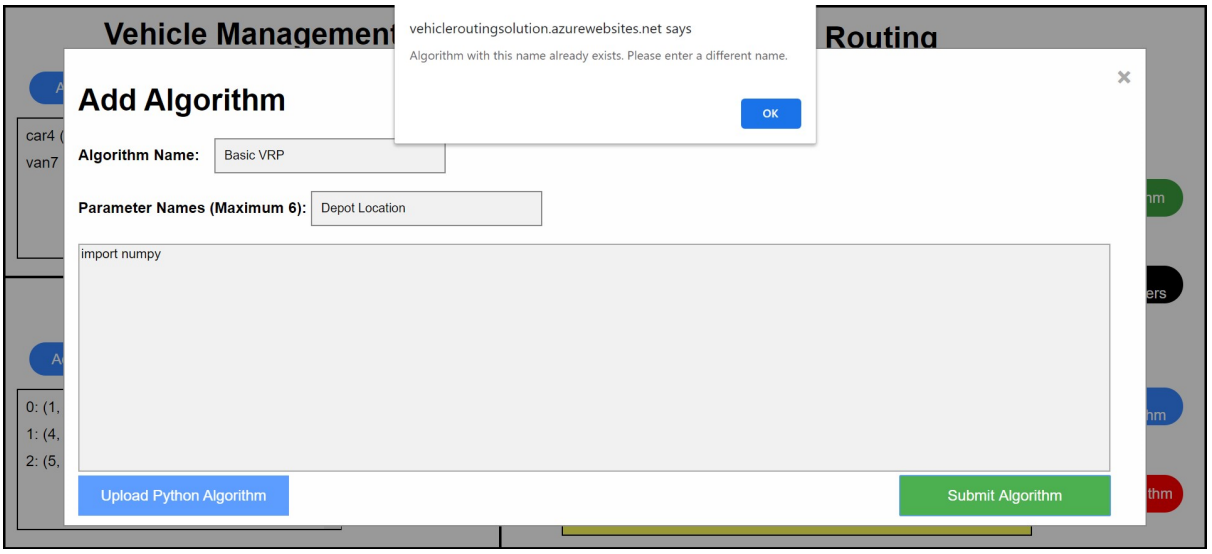


FIGURE D.7: Add a new Algorithm with an existing name

Appendix E

System Testing Screenshots

Screenshots for the system testing performed in Section 6.3 are shown over the following pages.

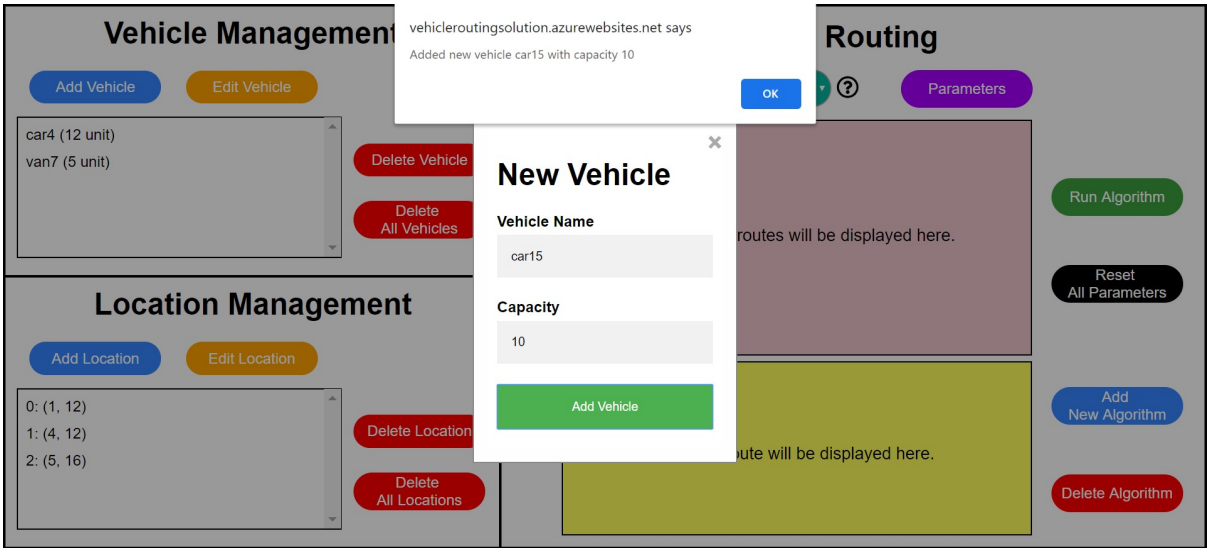


FIGURE E.1: Add a new Vehicle 1

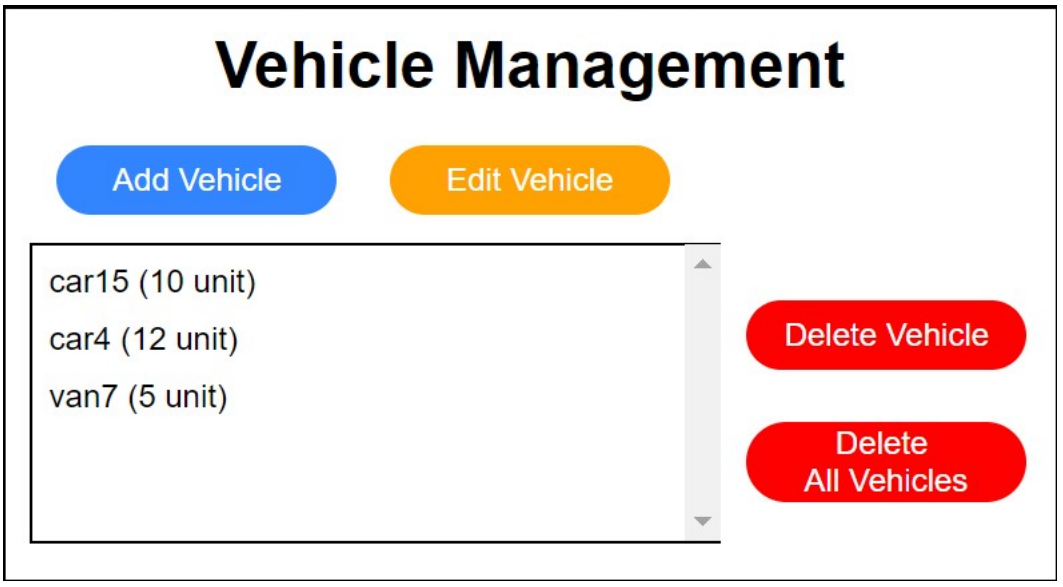


FIGURE E.2: Add a new Vehicle 2

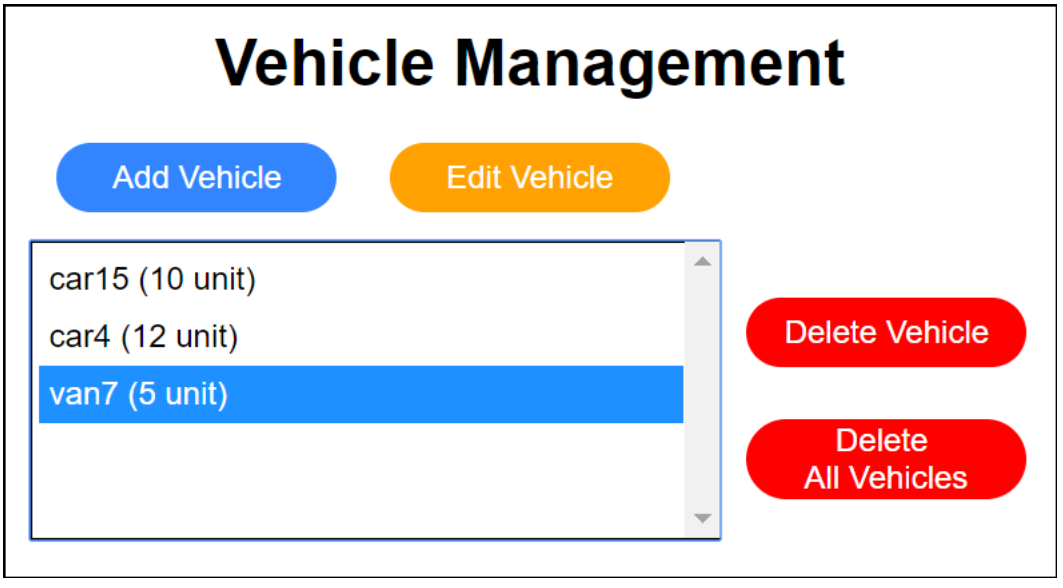


FIGURE E.3: Delete Vehicle 1

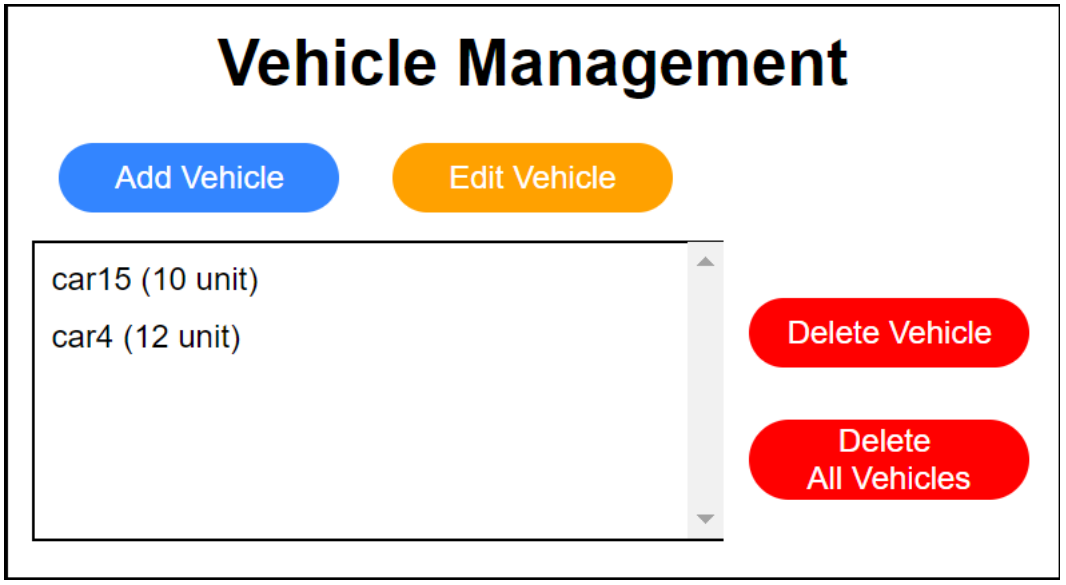


FIGURE E.4: Delete Vehicle 2

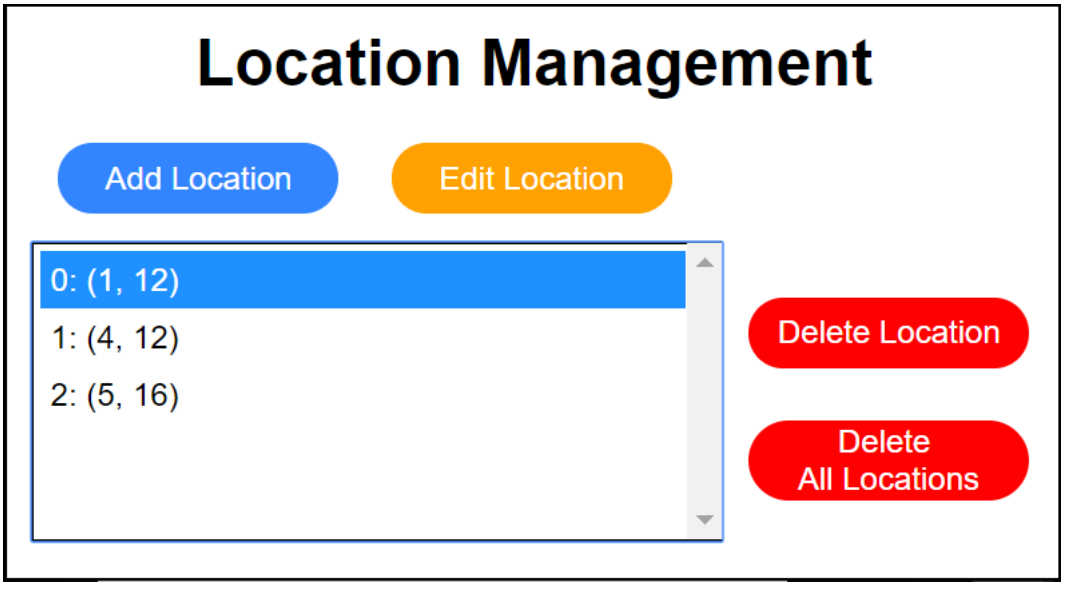


FIGURE E.5: Edit a Location 1

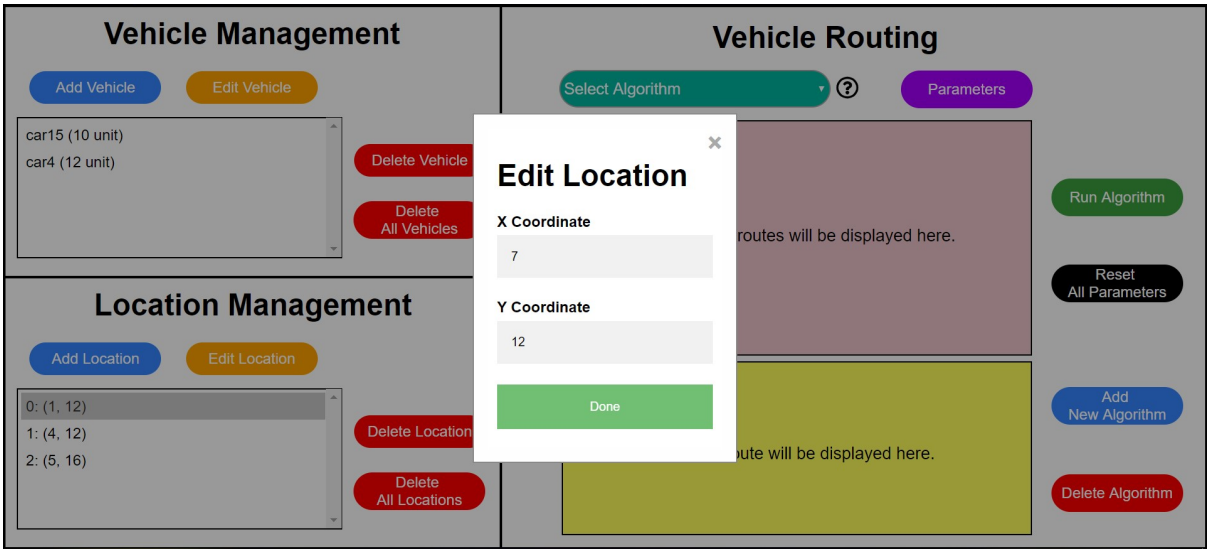


FIGURE E.6: Edit a Location 2

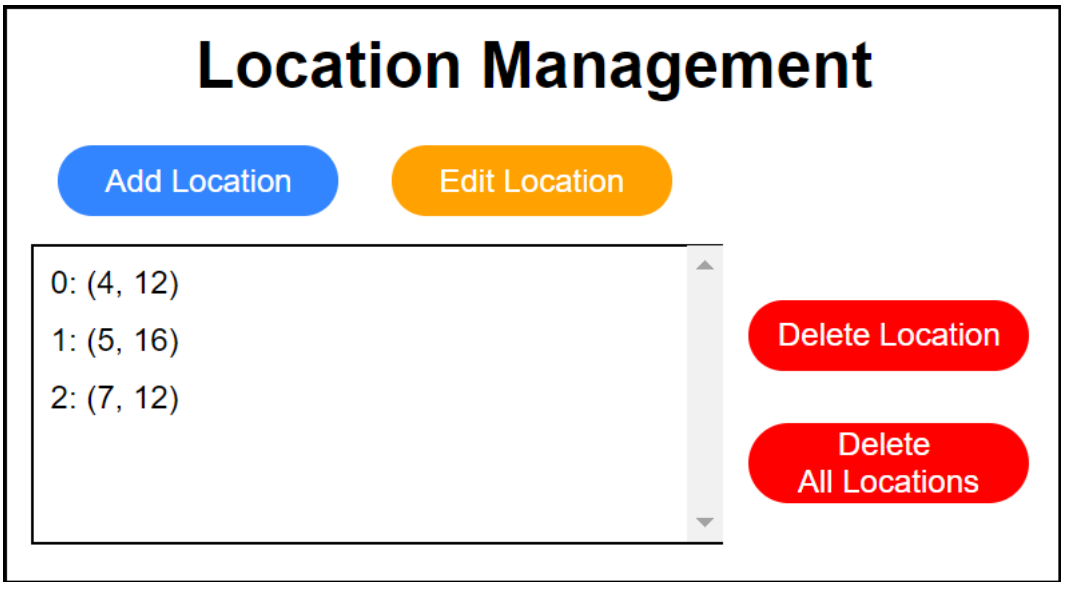


FIGURE E.7: Edit Location 3

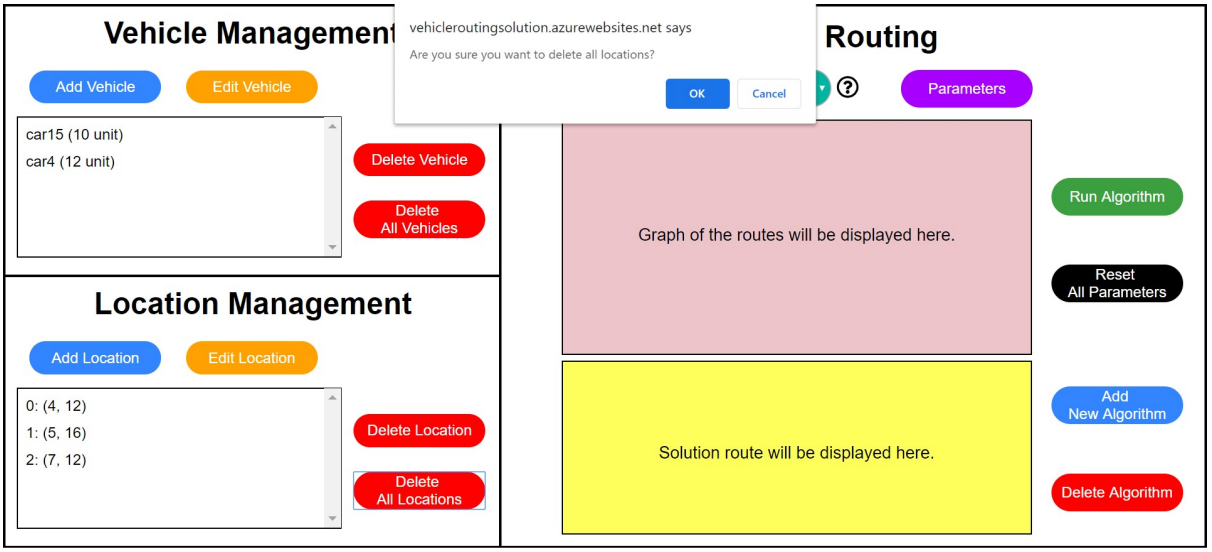


FIGURE E.8: Delete All Locations 1

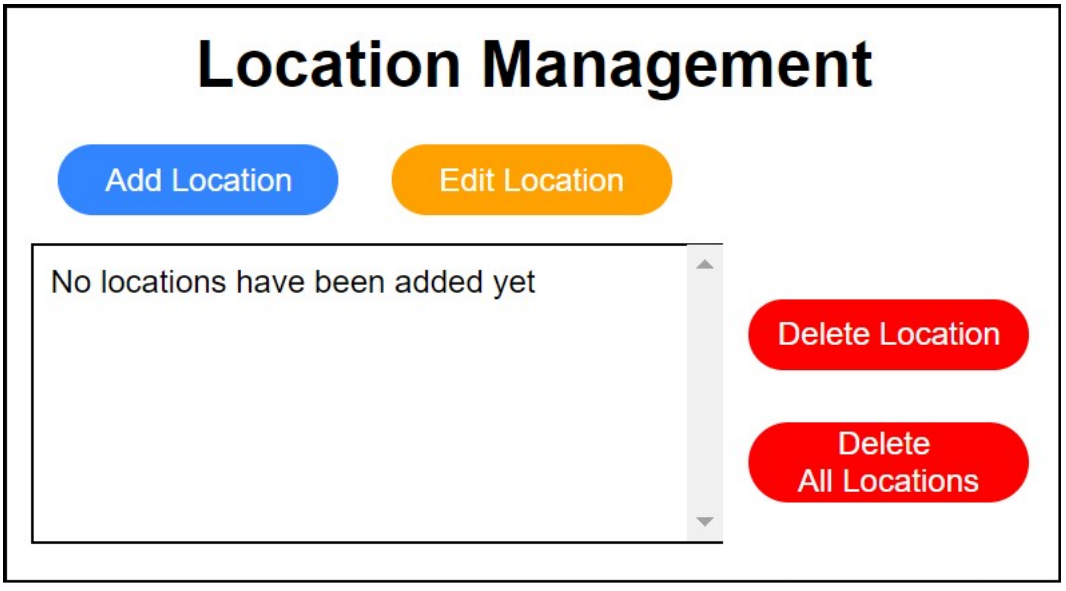


FIGURE E.9: Delete All Locations 2

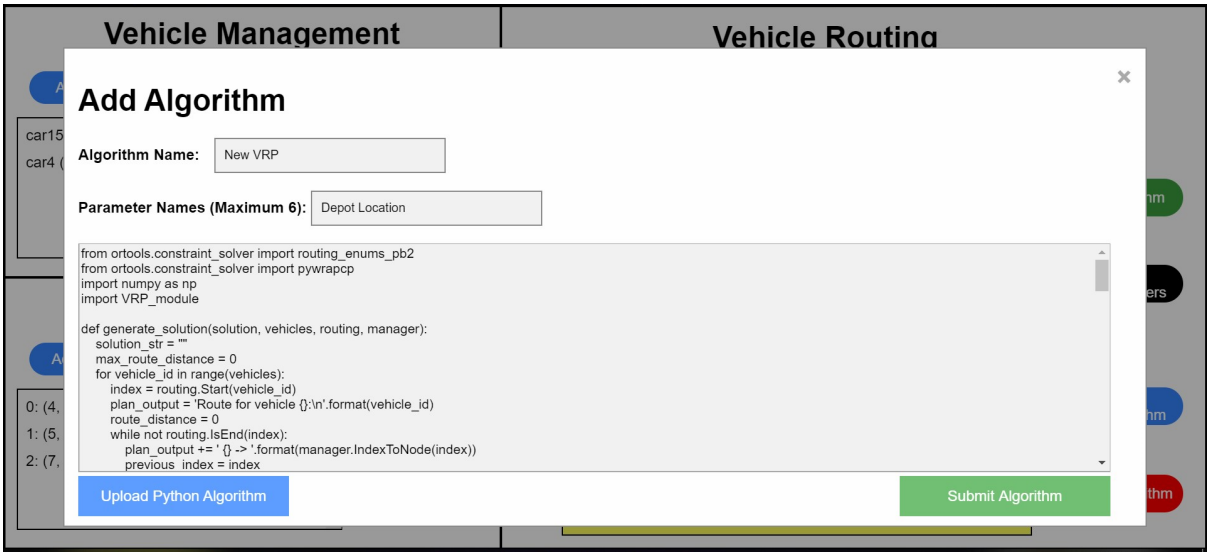


FIGURE E.10: Add a New Algorithm 1

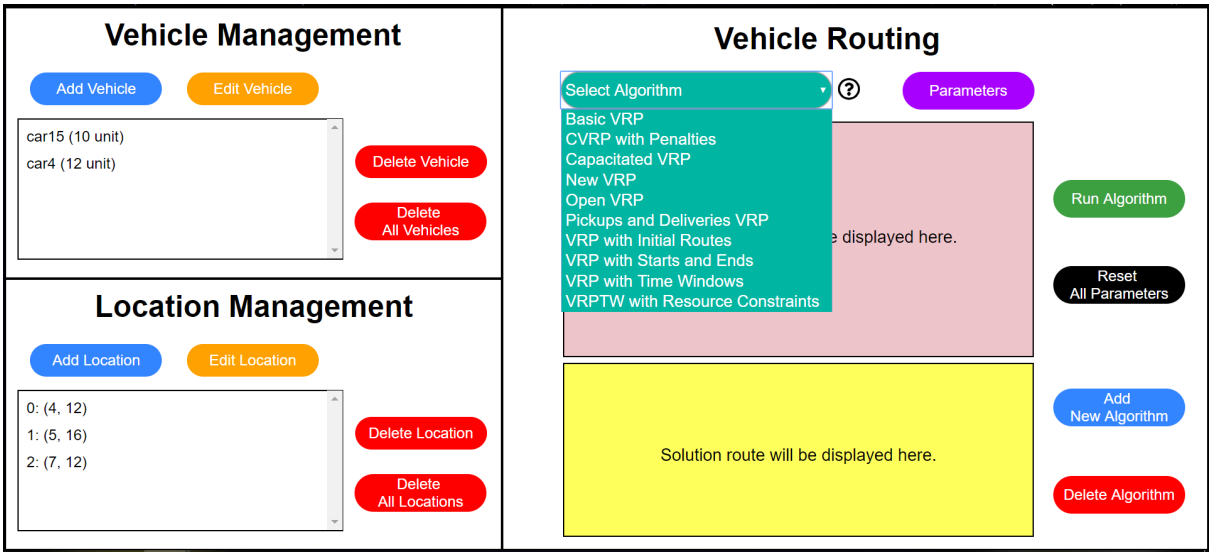


FIGURE E.11: Add a New Algorithm 2

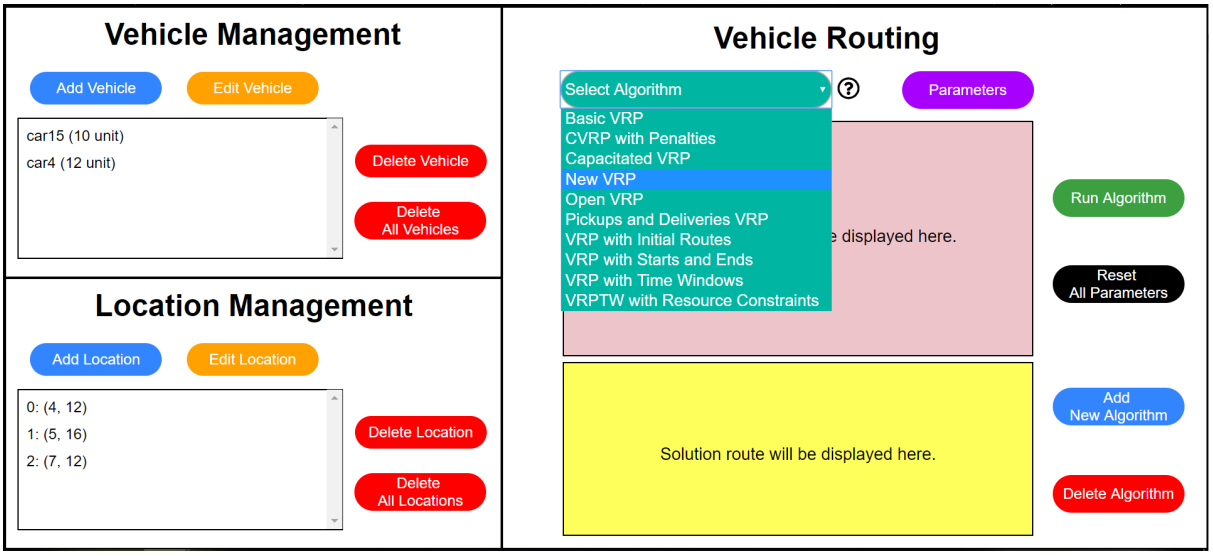


FIGURE E.12: Delete Algorithm 1

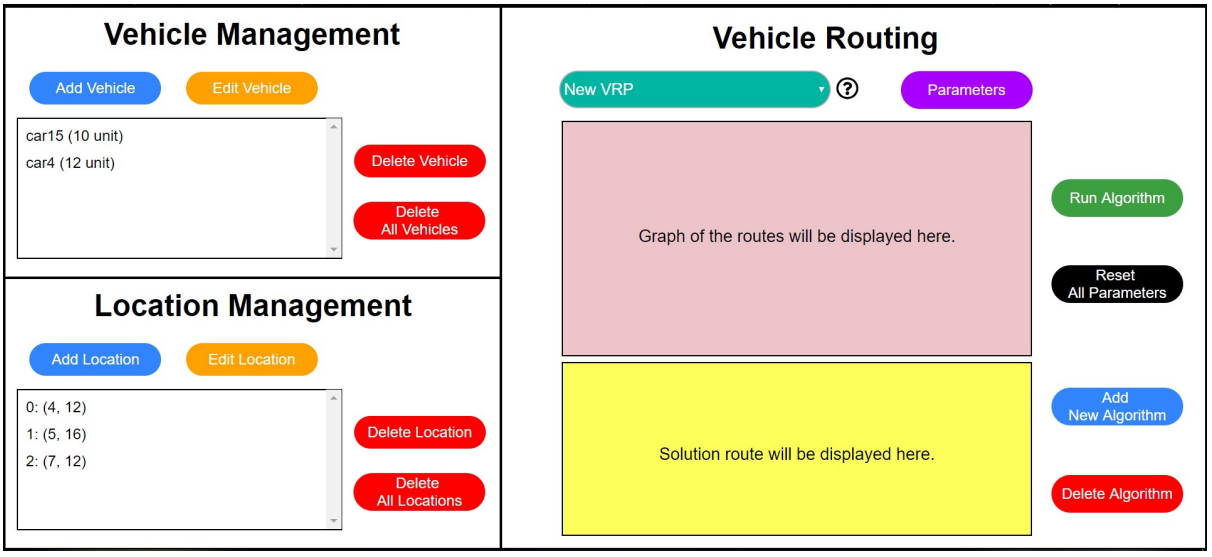


FIGURE E.13: Delete Algorithm 2

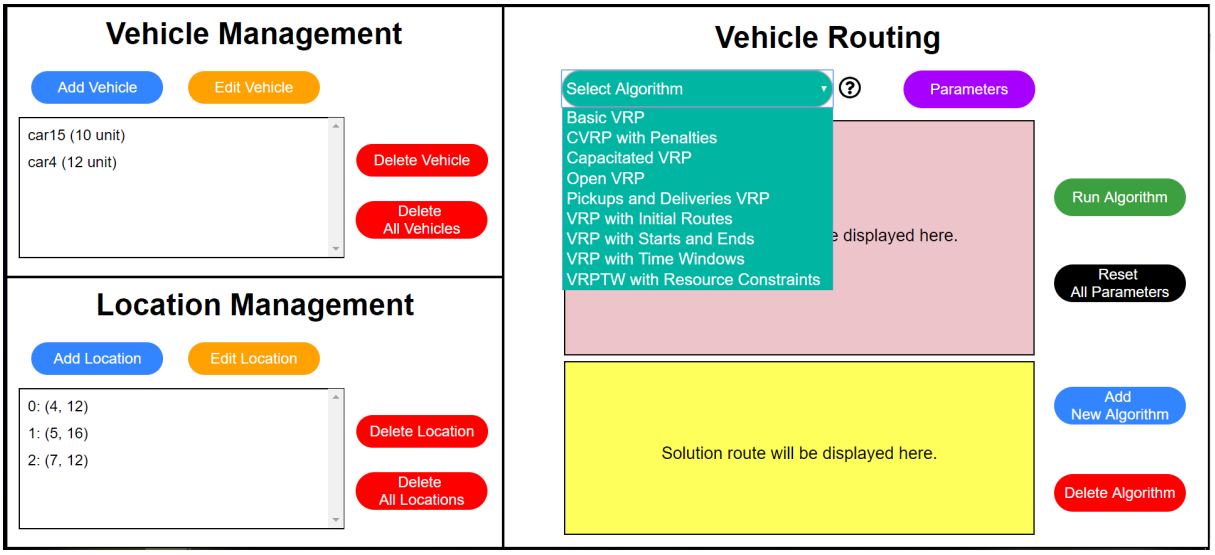


FIGURE E.14: Delete Algorithm 3

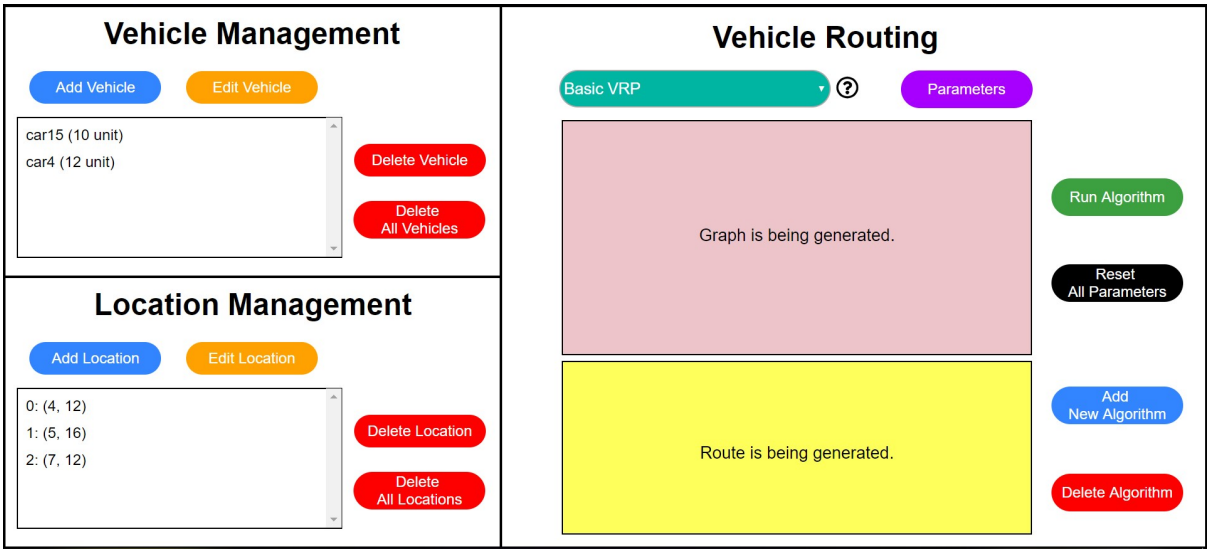


FIGURE E.15: Run Algorithm 1

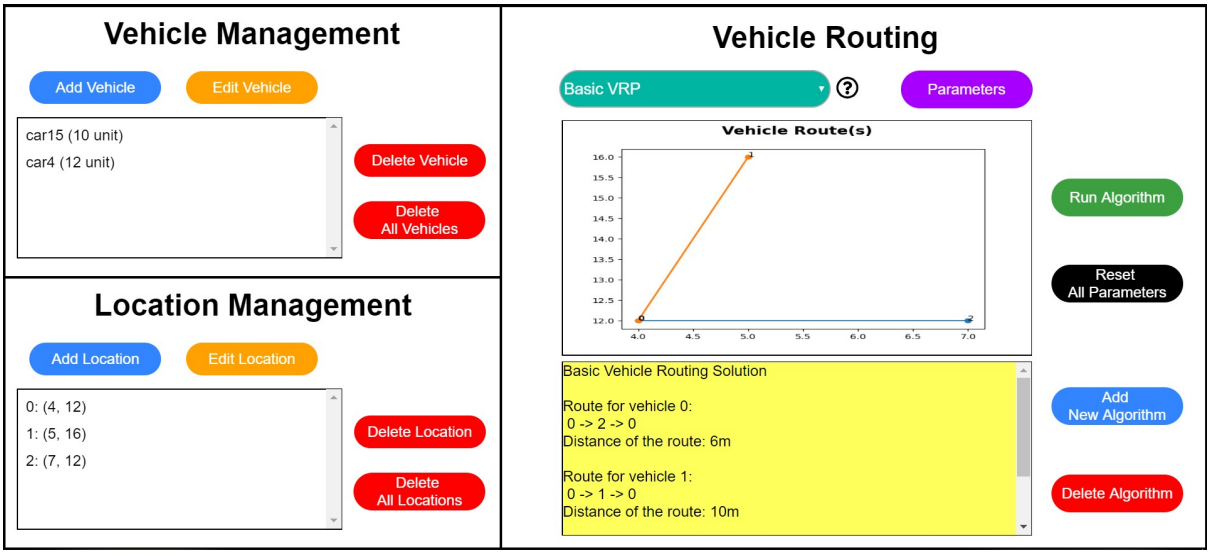


FIGURE E.16: Run Algorithm 2

Bibliography

AltexSoft, 2018. *The Good And The Bad Of Reactjs And React Native*. [online] Available from: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/.

Andriansyah, Nissa Prasanti, and Prima Denny Sentia, 2018. Pickup and delivery problem with LIFO, time duration, and limited vehicle number. *MATEC Web of Conferences*, 204, 07003. Available from: https://www.researchgate.net/publication/327794861_Pickup_and_delivery_problem_with_LIFO_time_duration_and_limited_vehicle_number.

Celeste Arbogast, 2017. Illinois Grainger College of Engineering: *Experiments show that a few self-driving cars can dramatically improve traffic flow*. Available from: https://grainger.illinois.edu/news/21938.

Youssef Bassil, 2012. A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering Technology*, 2 (5). Available from: https://arxiv.org/abs/1205.6904.

Zuzana Borčinova, 2017. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review*, 8 (2), 463-469. Available from: https://hrcak.srce.hr/file/285563.

José Brandão, 2004. A tabu search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, 157 (3), 552-564. Available from: https://www.sciencedirect.com/science/article/pii/S0377221703002388.

Bernd Bruegge and Allen H. Dutoit, 2013. *Object-oriented Software Engineering*, 3rd ed. Boston: Prentice Hall.

Krzysztof Bruniecki, Andrzej Chybicki, Marek Moszynski, and Mateusz Bonecki, 2016. Evaluation of Vehicle Routing Problem Algorithms for Transport Logistics Using Dedicated GIS System. *2016 Baltic Geodetic Congress (BGC Geomatics)*. Available from: https://ieeexplore.ieee.org/document/7548015.

Steve Burbeck, 2012. *Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC)*. Available from: http://www.dgp.toronto.edu/~dwigdor/teaching/csc2524/2012_F/papers/mvc.pdf.

Yanquing Q. Duan, John S. Edwards, and Mark X. Xu, 2005. *Information Management*, 42 (6), 799-811. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0378720604001193?casa_token=jU5sBPQLJqUAAAAA:1upstXxDbmzGyqRedJqn1xYvSb8Tlz-rm98T_1cDcMACFQuji8TohdyTSM_Ay3-GoNUtS0I.

European Environment Agency, 2019. *Increase in EU greenhouse gas emissions hampers progress towards 2030 targets.* [online] Available from: https://www.eea.europa.eu/highlights/increase-in-eu-greenhouse-gas.

Hongyi Ge, Tong Zhen, Yuying Jiang, and Yi Che, 2010. An Intelligent Solution for Open Vehicle Routing Problem in Grain Logistics. *Lecture Notes in Electrical Engineering*, 389-396. Available from: https://link.springer.com/chapter/10.1007/978-3-642-14350-2_49.

GlobalDots, 2019. *Serverless Computing Explained.* [online] Available from: https://www.globaldots.com/serverless-computing-explained.

Bruce L. Golden, S. Raghavan, and Edward A. Wasil, 2008. The Vehicle Routing Problem: Latest Advances and New Challenges. *Operations Research/Computer Science Interfaces.*

Google Developers, 2020a. *Penalties And Dropping Visits.* [online] Available from: https://developers.google.com/optimization/routing/penalties.

Google Developers, 2020b. *Resource Constraints.* [online] Available from: https://developers.google.com/optimization/routing/cvrptw_resources.

Google Developers, 2020c. *Vehicle Routing Problem.* [online] Available from: https://developers.google.com/optimization/routing/vrp.

Chrysanthos E. Gounaris, Panagiotis P. Repoussis, Christos D. Tarantilis, and Christodoulos A. Floudas, 2011. A Hybrid Branch-and-Cut Approach for the Capacitated Vehicle Routing Problem. *Computer Aided Chemical Engineering*, 507-511. Available from: https://www.sciencedirect.com/science/article/pii/B9780444537119501024.

Geir Hasle, Knut-Andreas Lie, and Ewald Quak, 2007. *Geometric Modelling, Numerical Simulation, and Optimization.* 397-431.

Manar I. Hosny, 2011. Heuristic Techniques for Solving the Vehicle Routing Problem with Time Windows. *International Proceedings of Computer Science and Information Technology*, 13. Available from: https://www.researchgate.net/publication/258262705_Heuristic_Techniques_for_the_Vehicle_Routing_Problem_with_Time_Windows.

Philip Kilby and Paul Shaw, 2006. Vehicle Routing. *Handbook of Constraint Programming*, 801-836. Available from: https://www.sciencedirect.com/science/article/pii/S1574652606800271.

Lucid Programming, 2017. YouTube: *Python and the imgur API: Image Upload.* Available from: https://www.youtube.com/watch?v=MyCr8kPT3OI&t=1s.

Microsoft, 2018. *Introduction To Table Storage In Azure.* [online] Available from: https://docs.microsoft.com/en-us/azure/storage/tables/table-storage-overview.

Microsoft Azure, 2020. *Web App Service.* [online] Available from: https://azure.microsoft.com/en-us/services/app-service/web/.

NHTSA, 2019. *Automated Vehicles for Safety.* [online] Available from: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

Ohio University, 2019. *The Future of Driving in the United States.* [online] Available from: https://onlinemasters.ohio.edu/blog/the-future-of-driving/.

Rabi Prasad Padhy, Manas Ranjan Patra, and Suresh Chandra Satapathy, 2011. X-as-a-Service: Cloud Computing with Google App Engine, Amazon Web Services, Microsoft Azure and Force.com. *International Journal of Computer Science and Telecommunications*, 2 (9). Available from: https://raw.githubusercontent.com/MarketLeader/Toko-Chetabahana/master/docs/cloud/analysis/cloud-computing.pdf.

LaTonya Pearson, 2015. *The Four Levels Of Software Testing.* [online] Segue Technologies. Available from: https://www.seguetech.com/the-four-levels-of-software-testing/.

Software Testing Fundamentals, 2019a. *Unit Testing.* [online] Available from: http://softwaretestingfundamentals.com/unit-testing/.

Software Testing Fundamentals, 2019b. *System Testing.* [online] Available from: http://softwaretestingfundamentals.com/system-testing/.

Synopsys, 2019. *What is an Autonomous Car? – How Self-Driving Cars Work.* [online] Available from: https://www.synopsys.com/automotive/what-is-autonomous-car.html.

Paolo Toth and Daniele Vigo, 2002. *The Vehicle Routing Problem.* Philadelphia: Society for Industrial and Applied Mathematics.