Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Management Science and Information Systems Studies

Final Year Project Report

Clustering in School Bus Route Optimisation for Good Travel Software

Diarmuid Coffey

April 2021

TRINITY COLLEGE DUBLIN

Management Science and Information Systems Studies

Project Report

GOOD TRAVEL SOFTWARE

Clustering in School Bus Route Optimisation

April 2021

Prepared by: Diarmuid Coffey ***Supervisor: Simon Wilson***

DECLARATION

I declare that the work described in this dissertation has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism ‘Ready, Steady, Write’, located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

I declare that the report being submitted represents my own work and has not been taken from the work of others save where appropriately referenced in the body of the assignment.

Signed: _____

Diarmuid Coffey

April 2021

ABSTRACT

The aim of this project was to develop techniques for Good Travel Software to cluster students into bus stop locations where more than one student can be picked up and dropped off to improve routing efficiency. A secondary goal of this project was to investigate techniques that could be used to simplify the resultant bus routes and ensure that stops are placed on suitable, non-minor roads with sidewalks. These goals have been achieved using a distance-constrained clustering algorithm, integrated with several community sourced routing and mapping APIs, that can impose hard walking constraints on students and evaluate the feasibility of bus stop locations.

PREFACE

This project was requested by Good Travel Software. Good Travel Software (GTS) was established in 2010 and has become a world leader in Mobility as a Service. It develops software solutions for the car, bicycle and scooter rental markets, and works with municipalities on optimal routing for their fleets. This project was focused on the routing aspect of their business, aiming specifically to improve the bus routing for one of Good Travel Software's clients, the school district of Halifax, Canada.

The final system meets the terms of reference for the project. Techniques have been successfully developed to perform distance constrained clustering to limit walking distances for students, and to ensure the sensible and optimal placement of bus stops, avoiding very minor roads, cul-de-sacs, and choosing locations with sidewalks. An analysis of two 'test states' was undertaken to explore the quality of the obtained solutions.

I would like to sincerely thank Professor Simon Wilson for his input, encouragement, and feedback over the course of the project. I would also like to thank my housemates and my family, especially my father.

GOOD TRAVEL SOFTWARE

Clustering in School Bus Route Optimisation

April 2021

TABLE OF CONTENTS

APPENDIX

1. INTRODUCTION

1.1 The Client

Good Travel Software (GTS) is one of the world's leading providers of Mobility as a Service software. Their software has been used to book over 300 million miles of journeys across the globe, through large and small car sharing services. Founded in 2010, GTS provides feature rich, highly configurable software that focuses on optimisation, aiming to reduce operating costs by increasing fleet utilisation and routing optimality.

GTS's focus for the future is to develop a multi-modal system, where a person can book a journey with multiple legs of transport across several different vehicle types, for example, bike to tram to bus, on one application, using one ticket. They believe multi-modal transport is a key part of the solution to the problems of congestion and pollution that afflict cities across the globe, and are on the forefront of the battle to use technology to address this challenge.

1.2 Project Background

One of the software solutions that GTS has developed is aimed at optimising school bus routes for school municipalities. These school districts must collect students from their homes each morning, bring them to their school by the beginning of class, and return them in the evening. GTS provides routing software that can take the addresses of pick-up and drop-off locations and propose routes that achieve the commute in as optimal a way as possible.

In certain school districts, students are only allowed to walk a certain distance from their house to their assigned bus stop. GTS would like to implement a system that uses the inputted addresses of students and produces suitable bus stops that satisfy these distance constraints. This would allow Good Travel Software to produce school bus routes from the ground up, ensuring optimality at every step. Even minor reductions in daily operating cost can result in significant end-of-year cost savings for the school districts in question.

Good Travel Software would also like to develop techniques to ensure that the potential bus stop locations follow other constraints. Some school districts necessitate that their students are picked up and dropped off at locations featuring pavements. Ensuring that the bus stops are not placed on very minor roads or cul-de-sacs can have a remarkable effect on routing efficiency. This again can cascade into considerable cost reduction for Good Travel Software's clients.

In this project, GTS would like to implement these new features into their system; clustering the students into bus stops that obey student walking distance constraints, placing these bus stops in suitable locations on permissible roads, and routing the buses through these stops in a close-to-optimal way.

1.3 Terms of Reference

The produced software must be able to take in the addresses of students in co-ordinate form and produce bus stops that satisfy the maximum walking distances that the school district has permitted.

A method to adapt the placement of these bus stops to improve routing efficiency is required. By moving the stops to different locations while remaining inside the walking distance constraints, costs can be reduced, and route efficiency can be improved. This method must be able to detect and minimise the use of cul-de-sacs and very minor roads.

Once these features have been introduced, techniques to identify whether a selected location has a sidewalk should be investigated. This feature will allow GTS to automate a time consuming, manual checking process, improving process efficiency.

During the project the terms of reference remained largely unchanged from the initial brief. Sidewalk checking, identified as a task that should only be pursued should time permit, was successfully implemented.

1.4 Summary of Remaining Chapters

- Chapter 2 provides an overview of the bus stop creation process that was developed in this project, illustrating the environment within which the process operates.
- Chapter 3 outlines the work that has been done to design and implement the bus stop creation process. The decision-making process and software engineering methodology are discussed, and the developments used to achieve the final pipeline are explained in detail.
- Chapter 4 examines the techniques designed to improve routing efficiency. The solution produced with these improvements on the sample dataset is compared to the base solution to illustrate the effects.
- Chapter 5 contains the final conclusions. It also sets out the final recommendations, and areas for further improvement and development.

2. SYSTEM OVERVIEW

This chapter gives an overview of the bus stop creation process. It outlines the system's purpose and its objectives and explores the technical environment within which it operates.

2.1 System Objectives

Outlined in Section 1.2, the purpose of the clustering process serves to automate a time-consuming process for Good Travel Software. Currently, the clustering of students into bus stops is done semi-manually, with human oversight needed to check if all students are walking below the distance constraint. The produced system will save on working hours. The system will also allow GTS to evaluate the placement of the produced bus stops in an efficient, adaptable manner. Road types, maximum permissible moved distance, and sidewalk presence checks can be easily adapted to allow GTS to meet the demands of the school district at hand.

The methods developed to improve routing efficiency are simple and can be shown to have a marked effect on the overall distance travelled by the buses while staying within the given constraints. This idea is further examined in Chapter 4. The two objectives listed above were achieved together in a cohesive, flexible data pipeline that achieves as optimal a solution as possible.

Explain the objectives of the system: cluster the students into appropriate bus stops, ensure the bus stops are on suitable roads, and then implement methods to reduce the travel time of the buses, thus improving efficiency.

2.2 System Overview

The completed data pipeline is a combination of methods from three Python files: clustering.py, routing.py and busStopCheck.py. The methods in these files can be linked together to take an input of student addresses through the clustering process (using methods contained in the clustering.py and busStopCheck.py files) and the routing process (contained in routing.py), with a CSV (Comma Separated Values) file of bus routes being produced at the end of the continuous process in a form that can be uploaded directly to the visualisation software.

A system overview diagram has been included in Appendix C.3, p. C.3 which outlines the inputs and outputs that each method takes to create the data pipeline, and the order in which to call them. The User Documentation included in Appendix C.2, p. C.1 provides a complete set of configuration instructions.

The students' co-ordinates are used to perform an iterative form of K-Means clustering that will produce bus stop locations, existing solely on the x-y co-ordinate plane, that obey the maximum walking distance constraint as the crow flies. This process is explained in Section 3.5. These bus stops must then be snapped to suitable locations on the road network using the Overpass API which is further discussed in Section 3.4.

This snapping process will allow stops to be moved up to 200 metres from the initial stop location from the K-Means clustering. As such, it is necessary to now re-evaluate the students' walking distances. The GraphHopper API, also discussed in Section 2.3, is used to create a

student-to-stop walking distance matrix. Reassigning the students to their closest stops using this matrix, the number of students now walking over maximum permissible distance can be found. These students are sent back into the K-Means clustering process, in a loop that sees more bus stops incrementally added until the process either reaches convergence and no improvement is being seen between iterations, or there are zero students violating the walking distance constraint.

Should convergence occur with several students still allocated walking distances above the constraint, bus stops are added directly on the students' co-ordinates and snapped to the nearest permissible road in a final effort to improve walking distances. At this point, the process of stop amalgamation begins, with the aim of improving routing efficiency while keeping the students beneath the constraint. This process is discussed in Section 4.3.

The final stage of the pipeline involves using Google's OR-Tools, a powerful open-source software suite built to tackle vehicle-routing problems, to produce a close-to-optimal set of routes through the created bus stops. This solution set is used to calculate the students' total travel times and is then turned into a CSV form that is ready to be uploaded to Good Travel Software's visualisation tool. This process is discussed in Section 4.2.

2.3 Technical Environment

As described in Section 2.2, the data pipeline has been implemented using 3 Python 3.7.9 files. Good Travel Software already used Python to perform their routing using a Python wrapper of Google OR-Tools, so it was decided that Python was the most suitable language to perform clustering. Several other key packages and Application Programming Interfaces (APIs), used throughout the process, are discussed below.

The Overpass API

The Overpass API, formally known as OSM (OpenStreetMap) Server-Side Scripting, is a read-only API that allows read-only access to OSM map data. OpenStreetMap is an ambitious, collaborative worldwide project that aims to create a crowdsourced map of the world. Inspired by the success of Wikipedia, OpenStreetMap encourages users to contribute to the project by updating the database with information about the infrastructure in their area. The OpenStreetMap project proved to be integral to the development of this system. Queries to the Overpass API to retrieve OpenStreetMap data must be written in Overpass QL (Query Language), an imperative, procedural programming language written with a C style syntax. The Overpass API was used to evaluate whether a location was suitable for a bus stop, taking a query using the 'requests' package and returning a JSON (JavaScript Object Notation, a popular method to transfer data) object that is dealt with using the json package. Both packages are discussed below, and the stop evaluation process is further discussed in Section 3.5.

The GraphHopper Directions API

GraphHopper is an open-source routing library that provides several routing APIs and is based on OpenStreetMap data. One of such offerings, the Distance Matrix API, allows large many-

to-many distance matrices to be calculated. This API is used multiple times in the pipeline, to create inter-bus stop distance matrices, and to create student-to-stop walking matrices. This free tier of this service has a minutely and daily credit limit that is discussed in Section 3.6.

Google OR-Tools

Google OR-Tools is an open-source software specialising in ‘combinatorial optimisation’. Good Travel Software already uses OR-Tools vehicle routing library to solve the vehicle routing problem that results from the created bus stops. OR-Tools takes a distance matrix of the stops, the number of students at each stop and the capacity of the buses and returns a set of close-to-optimal routes, detailing the order in which the stops should be visited, and the distance travelled by each bus. The OR-Tools process is discussed in Section 4.2.

Other Python Packages

Various other Python packages became necessary to include in the final pipeline. The NumPy package, designed to handle large, multi-dimensional matrices and arrays, is used throughout the project to store the bus stops and students. The ‘pandas’ library is used for its comprehensive library of tools to read and write data to and from CSV formatted files. The Python ‘time’ module allows running time to be monitored for time-consuming functions and provides crucial waiting periods that prevent the system from violating the GraphHopper credit limit per minute, referred to as the “minutely” limit in their documentation. This is explained further in Section 3.6.

The iterative distance constrained clustering system explained in Section 3.5 involves the KMeans function from the module sklearn.cluster and the KneeLocator from the kneed repository. Geopy.distance is also used here to obtain as-the-crow-flies distance from the co-ordinate data. The process of requesting and processing responses from the Overpass API necessitated the inclusion of the requests library, the json module and the KDTree class (from the scipy.spatial package).

The use of these packages is explained further in Sections 3.4 and 3.5, in the functions within which they are called.

3. DESCRIPTION OF THE WORK DONE

This chapter outlines the development of the data pipeline, its features and the reasoning behind the decisions taken. Section 3.4 focuses on the development of the Overpass API querying system to move the bus stops onto the road network, while Section 3.5 describes the distance constrained clustering algorithm used to create the bus stops and assign the students.

3.1 Software Requirements

Initial discussions with the client, as discussed in Section 2.1, outlined the two primary objectives of the system. The designed system should perform distance constrained clustering, choosing legitimate stop locations, and implement methods to improve routing efficiency in the resultant bus routes. The terms of reference distilled these aims into several requisite features that guided the development process:

- The system must produce bus stops that keep student walking distances below the maximum walking constraint where possible. In the sample data, this walking constraint was 400 metres but the permissible walking distance is configurable.
- The bus stop locations must be placed on suitable road types, avoiding non-minor roads and cul-de-sacs. This was further clarified to mean non-private primary, secondary, tertiary, and 2 lane-residential roads.
- Methods should be developed to improve route efficiency. This request was further clarified to concern the exploration of methods to adjust the placement of bus stops to more-optimal positions within the walking distance constraints.
- The implementation must avoid using any paid APIs, existing solely on free, non-subscription software.
- The development of a method to identify whether a chosen location included a sidewalk was requested.

As the development process continued, an additional request that the final output was compatible with Good Travel Software's visualisation software was accepted and implemented.

3.2 Software Engineering Methodology

The nature of these software requirements meant that an agile development methodology suited this project. Agile software development focuses on an incremental, iterative development process that implements client input at every iteration (Biju, 2008). This allowed the success of different methods to be evaluated and adapted upon as development proceeded.

At the beginning of the process, metrics to quantify improvements were established with the client's oversight. The delivery process then involved creating a solution to the problem, and then evaluating that solution along these metrics to decide whether the adaptation had a noticeable improvement compared to the last iteration. Weekly meetings with the client were

used to discuss the success of previous implementations, and to direct the future evolution of the pipeline. The agile development process allowed inventive original solutions to be pursued throughout the engineering process.

3.3 Software Selection

This project started with several software suggestions from the client themselves. Their pre-existing routing methods were written in Python and made use of Google OR-Tools. Consequently, any further methods developed to cluster and improve route efficiency should also be written in Python. The specific version of Python used was Python 3.7.9.

Snapping to Roads

Moving hypothetical bus stops to real-world locations on the road network was one of the most important choices. There were many platforms that provided this feature but were ruled out for various reasons. Google provides a road API that performs this task to the nearest road. However, this feature uses a pay-as-you-go pricing model. While GraphHopper provides a map matching feature, it's "minutely" and daily limits made its implementation unworkable.

The Overpass API was the perfect solution to the task as it was able to achieve this goal without introducing paid software to the system. Its query system, explained in Section 3.4, is highly customisable, allowing other features like sidewalk checking and bus stop road prioritisation to be built straight into the querying system. These features are discussed in Sections 3.4 and 4.3, respectively. The Overpass API is also free to use, making it the best choice for this task.

Distance Matrices

The decision for which API to use to create the necessary distance matrices was influenced by choices already made by the client. The decision not to use subscription software meant that Google Maps Distance Matrix API was unsuitable due to its pay-as-you-go pricing (Google, 2021). GraphHopper was chosen for this task as the client was already using this service for their routing calculations. GraphHopper's free tier allocates 500 credits a day, which, when used sparingly, was enough for the confines of this project (GraphHopper, 2021). The free tier does however enforce a credit limit per minute, or "minutely" limit on requests. Methods requesting the GraphHopper API were surrounded by 60 second waiting periods to avoid violating this limit.

3.4 Querying the Overpass API

The process of snapping stops to suitable locations on the road network is a key part of this project. This involved quantifying what made a location suitable for a bus stop and exploring methods to search for a location that satisfied these criteria.

The final implementation makes use of data from the OpenStreetMap project, accessed using the Overpass API. Learning to use this API was a non-trivial task. Thankfully, a web-based tool called Overpass turbo provides a 'query wizard', shown in Figure 3.4.1, that assists with the development of queries by converting simple, human-readable search terms into functional Overpass queries.

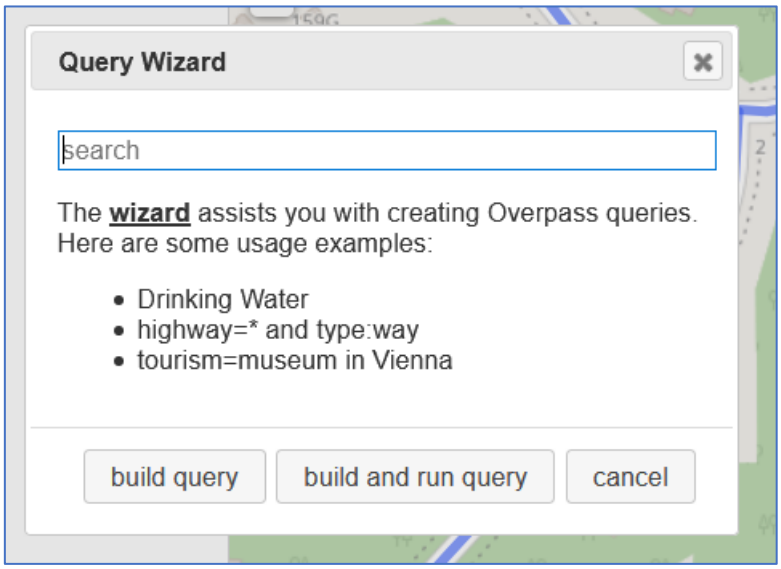

FIGURE 3.4.1 – Overpass Turbo's Query Wizard

Quantifying a Suitable Road within OpenStreetMap data

The first step to developing an Overpass API query to achieve this task was to determine what made a road suitable for a bus stop. It is expected that this will change drastically between different municipalities, as, not only will different school districts require different needs, but the practices of OpenStreetMap road tagging may vary between different areas.

OpenStreetMap data features three different types of basic data structure: 'nodes', 'ways', and 'relations'. These structures are further described through attached tags. There is no formal guideline for the nature of tags; instead, the community agrees on certain key and value combinations for the most popularly used tags. 'Ways' in OpenStreetMap are used to represent linear features, including roads, paths, rivers and walls. It became necessary to explore the 'ways' in the surrounding area of the sample dataset (set in Halifax, Canada) to create a list of suitable and unsuitable tags.

For this process, roads in the area of the sample dataset were examined using Google Street View and OpenStreetMap. Figure 3.4.2 illustrates this process, with a suitable bus stop location on Larry Uteck Boulevard displayed on both Google Street View and OpenStreetMap.

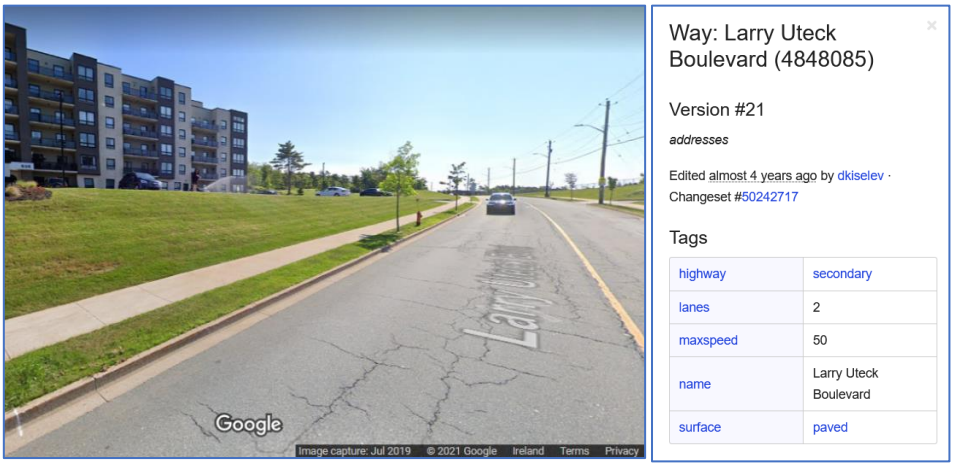

FIGURE 3.4.2 – Larry Uteck Boulevard in Halifax Canada on Google Street View (L) and its tags on OpenStreetMap (R)

List of Relevant OpenStreetMap Tags

- **"highway":** The highway key is the main key used for identifying roads and paths. It is used to identify ways ranging from motorways (highway = motorway) to residential roads that people live on (highway = residential) to access roads like car parks and alleys (highway = service) and is also used to describe nodes that represent turning circles at the bottom of cul-de-sacs (highway = turning_circle). From the exploration of the Halifax area, it was decided that all primary ways, and certain secondary, tertiary, and residential ways were suitable for bus stops.
- **"access":** The access tag is used to denote roads that exist solely for private use (access = private) or business entries. It is necessary to check locations for this tag to make sure the bus stop is not placed in a gated community or on a road with restricted access.
- **"lanes":** The lanes tag describes the number of lanes on the road in question. This is useful to assess the characteristics of a residential road, as one lane residential roads in the sample dataset were not suitable for bus stops.
- **"sidewalk":** The sidewalk tag identifies a road that has a sidewalk. This tag is useful to decide whether a location has a sidewalk. However, the use of this tag is varied and inconsistent. This is discussed in Section 3.7.

The Designed Overpass Query

```
[out:json][timeout:25];
(way["highway"="residential"](around: "search_distance", "lat", "lon");
 way["highway"="tertiary"](around: "search_distance", "lat", "lon");
 way["highway"="secondary"](around: "search_distance", "lat", "lon");
 way["highway"="primary"](around: "search_distance", "lat", "lon");
 node["highway"="turning_circle"](around: "search_distance", "lat", "lon");
);out body ids geom;
```

FIGURE 3.4.3 – The Overpass API query designed to return ways suitable for bus stop locations in Overpass QL

Figure 3.4.3 features the query that has been designed to return ways suitable for bus stop locations. [out:json] requests that the output return in JSON format. The last line of the query specifies the information about the objects that is desired. In this query, 'body' returns the tags, whereas 'ids' and 'geom' return the co-ordinates of the ways, needed to return the final bus stop location. The process of building this query in the pipeline is handled by busStopCheck.get_roads_coords_query(), which takes in a co-ordinate location and a search distance, and returns a string that contains the desired query in Overpass QL.

Checking the JSON Response

The JSON object returned by this query can on occasion feature ways that are not suitable for bus stops, as the other tags, like "access" and "lanes" are not featured in the query. As such, the JSON object only features possible ways, which must be further explored to certify suitability. This checking process, illustrated in Figure 3.4.4, is performed in the busStopCheck.return_suitable_location_outSearch() method.

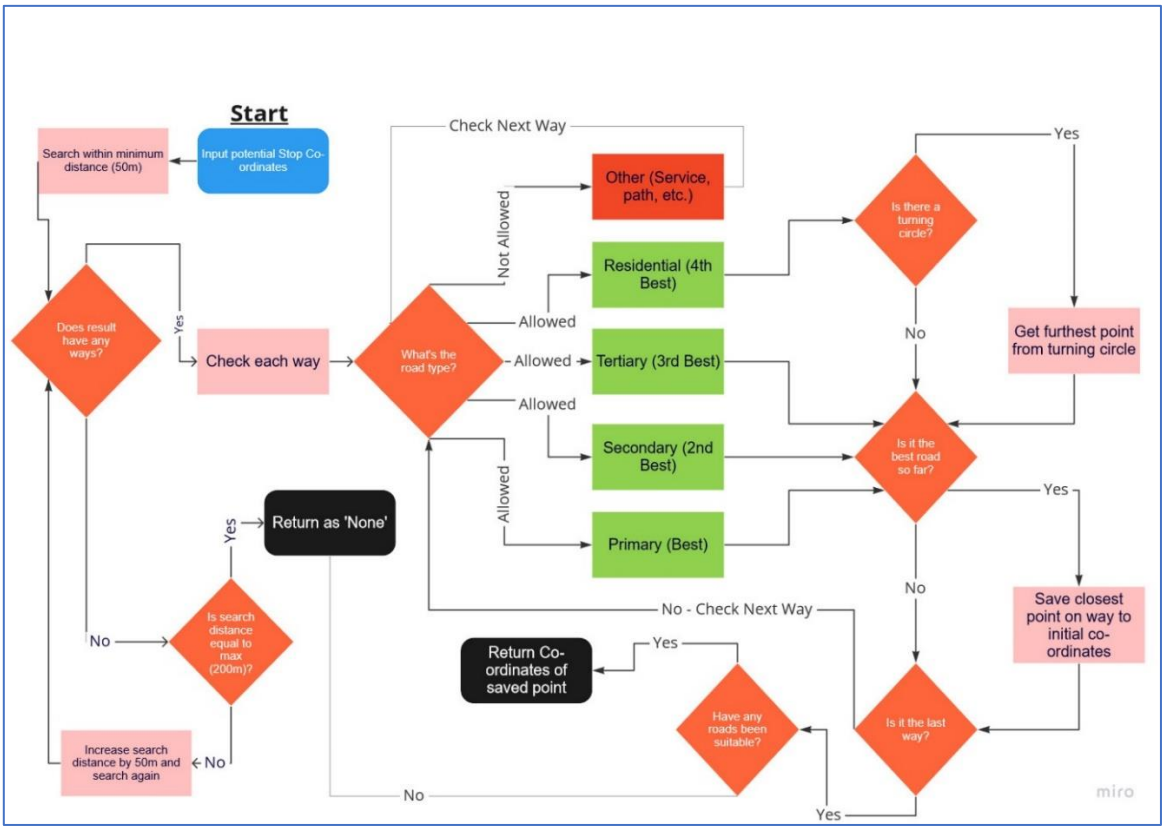

FIGURE 3.4.4 – The checking process used to decide whether a location is suitable for a bus stop

The JSON response is checked for suitability. Each way (the OpenStreetMap term for a road) is examined individually to determine whether it is suitable for a bus stop. This process differs for each highway type; primary roads are immediately accepted, secondary and tertiary roads are checked for access tags, and residential roads are checked for access and lane tags. The final implementation also features a priority system intended to improve route efficiency, discussed in Section 4.3.

Iterative Search System

The final Overpass API query implementation is surrounded in an iteratively increasing search system that aims to find the most suitable road to the hypothetical bus stop location created by the distance constrained clustering method (described in detail in Section 3.5). A query is initially sent to the Overpass API with a search distance of 50 metres. Should no roads be returned or should the roads that are returned be unsuitable for bus stops, the search distance is increased by 50 metres and resent to the API. This process will continue until either a suitable road is found, or the search distance reaches the maximum permissible distance of 200 metres. If the maximum permissible distance is reached, the initial co-ordinates will be returned, along with a road type value of 'None'.

Return Closest Point on Chosen Road

The iterative search system discovers the most suitable road for the hypothetical bus stop location to be moved to. However, the road (an OpenStreetMap 'way' object) consists of several co-ordinate nodes linked together to form the street. The closest point between the chosen way and the initial bus stop location is found by turning the way into a K-D Tree by the scipy.spacial.KDTree method from the scipy.spacial package. A K-D Tree (short for k-dimensional tree) is a form of binary partitioning tree that can be used to quickly discover the closest point.

The closest point on the way is returned along with the road type that the stop will be placed on. This method of returning the closest distance is not used if the chosen way is a residential road and a turning circle was detected during the search process. This is due to a method developed to improve route efficiency that is discussed in Section 4.3.

Checking for Sidewalks

The Overpass API response was useful for implementing the sidewalk checking feature. However, sidewalks exist in many different forms in the OpenStreetMap data. Sidewalks can feature as a 'sidewalk' tag on an existing residential, tertiary, secondary or primary way, or as a 'footway' way, or a 'foot' tag on a path way. The process implemented in busStopCheck.sidewalk_check_stop() checks for all these kinds of sidewalk within 15 metres of the given co-ordinates. Unfortunately, OpenStreetMap data is inconsistent in its use of these sidewalk tags. This is discussed further in Section 3.7.

3.5 Designing the Stop Location Algorithm

This section outlines the design of the constituent parts of the stop location algorithm. It gives details on the methods and reasoning behind the functions themselves, explains how the functions fit into the pipeline, and how the data is changed by the function itself.

Distance Constrained Clustering

The distance constrained clustering process used in clustering.add_extra_stops() is an adapted version of the K-means clustering process. K-means clustering is an unsupervised machine learning algorithm. The K-means clustering process aims to partition the data points into K clusters, with each cluster containing similar data points. It involves 4 steps, repeated until the cluster centroids stabilise and there is no change to the centroids between iterations. The steps are as follows:

- Assign k initial cluster centroids randomly.
- Calculate the distance between the data points and the cluster centroids
- Allocate each data point to the closest cluster centroid.
- Evaluate each cluster, recalculating the cluster centroid as the mean of all the members of the cluster.

This K-means process will produce a specified number of clusters. The adapted, distance constrained K-means clustering process that is used in this project involves clustering the data points, or students' home address co-ordinates, into K centroids, or bus stop co-ordinates. These bus stops are then evaluated individually, checking each students' distance, as the crow flies, from their assigned bus stop using the geopy.distance package. Should any of the students be more than the maximum permissible walking distance from these bus stops, the number of clusters, or K, is incremented by one, and the K-means process is repeated.

The result of this loop is that K bus stops are created. Each student is assigned to a bus stop that is within the maximum permissible walking distance as the crow flies. There are two problems with these bus stops. The first is that they exist solely on the X-Y co-ordinate plane. These stops must be moved to positions on the road network that buses can reach, and that satisfy the criteria for bus stop locations. This issue is solved using the Overpass API in the snapping to roads feature, discussed in Section 3.4.

The second problem is that the bus stops have been created to ensure that the straight-line distance between student and stop is below the walking constraint. In practice, students must walk along roads to the stops, meaning that these initial bus stops may not satisfy the constraints. This issue is also compounded when the stops are snapped to the road network, as this process may again create student-stop allocations that violate the permissible distance. An initial solution to this problem involved using GraphHopper's Distance Matrix to evaluate the walking distances between each student and every stop. This solution quickly violated GraphHopper's minutely and daily access limits, discussed further in Section 3.6. The current solution uses Euclidean distance (straight line distance) to close in on an adequate bus stop configuration, which is then improved using the Student Reassignment method.

Student Reassignment

The student reassignment method takes in students, snapped bus stops, and the student-to-stop walking matrix from the GraphHopper Distance Matrix API. For each student, the walking matrix is searched to find the closest stop along the road network. The student is then reassigned to that stop. The difference between the theoretical straight line walking distance and the empirical walking distance supplied by GraphHopper is significant. This process usually leads to several students now walking over the walking distance constraint to get to their closest bus stop, and sometimes leads to stops that have now been left completely vacant. These redundant stops are dropped from the solution set, and the students that are still walking over the walking constraint (stored in the 'overs' array) are sent back through the stop creation system. This process is described in the Stop Creation Loop section.

The Stop Creation Loop

The Stop Creation Loop (clustering.stop_creation_loop()) handles the process of iteratively reducing the number of students still walking over the walking constraint by creating extra bus stops. The loop is illustrated in Appendix C, p. C.5. The process makes use of several of the previously described functions. At the beginning of each loop, the number of 'overs' from the last loop (every student is classified as an over in the first loop) are sent to the distance constrained clustering process. The resultant bus stops are snapped to the road network and combined with the bus stops from the previous iteration. This means that the number of stops in the solution grows in every iteration of the stop creation loop, improving the solution by lowering the student walking distances. A walking matrix is formed using the GraphHopper Distance Matrix API, and this walking matrix is used to reassign the students to their closest bus stop and identify the 'overs'. These 'overs' are sent back through the loop.

Intuition would suggest that eventually this loop would bring all student walking distances below the maximum walking distance. However, issues inherent in the OpenStreetMap data that are further discussed in Section 3.7 can prevent this from happening. If one of these issues is encountered, the loop will continue to send the same students to the distance constrained clustering method, creating new stops while the number of students violating the constraint remains the same. This is combatted by keeping track of the number of 'overs' between iterations. If this number remains the same for two consecutive loops, or if the number of students over the constraint becomes zero, then the loop is exited.

The results from the stop creation loop, when performed on the sample dataset from Halifax, Canada, can be seen in Table 3.5.1. In loop 3, the number of students walking over 400 metres has remained constant for two loops, so the add final stops method is called. This method is explained later in this section.

Table 3.5.1 – Results from the stop creation loop performed on the sample dataset.

Loop Number	Number of Bus Stops (Total)	Number of bus stops (Introduced in this loop)	Average walking distance (m)	Number students still walking over 400 metres
1	211	211	163m	76
2	242	31	131m	32
3	242	0	131m	32
Add Final Stops	251	9	126m	24

Add Final Stops

If two consecutive iterations of the stop creation loop have the same, non-zero number of students violating the walking distance constraint, the method clustering.add_final_stops() is called. This function attempts to create a unique bus stop for each student remaining over the distance constraint directly on their co-ordinates. The absence of the clustering step allows this method to avoid the pitfalls of the previous two iterations, and acts as a last-ditch method to discover any potentially suitable locations. Like in the stop creation loop, these stops are sent to the GraphHopper Distance Matrix API to create the walking matrix needed for the student reassignment process.

Form Output in Visualisable Form

The client requested that the final output from the clustering and routing process be suitably formatted to be entered into their visualisation software. The function routing101.routes_to_csv handles this process using information from the routes and the bus stops. The final format is described in Appendix D, p. D.1.

3.6 Issues Encountered

Several issues were encountered during the development of the stop creation pipeline. Most of these issues were able to be negated using creative problem-solving but it is important to mention them as guidelines for future development.

GraphHopper API Daily and Minutely Limits

The client's use of free software was a prudent decision, as all necessary tasks were able to be achieved using non-subscription offerings. However, GraphHopper's Distance Matrix API uses a daily and minutely credit limit that necessitated innovation. At several stages in the development of this project, the stop creation loop would max out the daily credit, preventing any further distance matrices to be created that day. There were also situations where the loop attempted to create an inter-stop distance matrix and a student-to-stop walking matrix within the same minute, this time contravening the minutely limit.

Minutely credit limit infringements were negated by implementing 60 second waiting periods between GraphHopper API requests using the 'time' Python module. Daily limit transgressions

were harder to avoid. As discussed in Section 3.5, straight-line distance calculated by the geopy.distance module was used to approximate walking distances where possible, reducing GraphHopper distance matrix requests to a tolerable level.

GraphHopper API Routing Inconsistencies

The GraphHopper Distance Matrix API does not store any distance calculations. This can sometimes mean that two distance matrices can disagree on the distance between the exact same two points. This issue did not introduce any difficulty to the development of the pipeline, but it is worth mentioning that it may mean that optimal solutions may vary slightly between iterations.

Overpass API: JSON Decode Errors and Response Latency

The use of the Overpass API in this project can result in the processing of several hundred individual queries. While the Overpass API is optimised for large-scale data consumers, who may need up to 10 million elements in several minutes, the response times experienced throughout the development of this project were inconsistent. Some queries returned in three seconds, while others took up to 90 seconds. This made the snapping to roads process by far the most time-consuming step of the pipeline, sometimes taking up to 3 hours to move all suggested bus stops to suitable locations.

The responses received from the Overpass API sometimes tended to cause a JSON Decode Error, causing the whole script to crash. It is unclear what caused this error but sending the same query back to the Overpass API usually resulted in success. As such, the Overpass query processing system is surrounded by a try/catch exceptions loop, preventing these sporadic JSON Decode Errors from derailing the entire pipeline.

Inconsistent OpenStreetMap data

The Overpass API makes use of OpenStreetMap data. The crowd-sourced nature of this data can lead to inconsistencies that may provide inconclusive results. A key example of this exists in the sample dataset in Halifax, Canada. Several students live in an area that is surrounded on 3 sides by Glen Arbour Golf Course. The access road to their house would seem suitable for a bus stop, and there is evidence on Google Maps Street View of public vehicles making use of the streets. However, the ways describing these roads have been given the ‘golf-cart=designated’ tag due to their proximity to the golf course. As a consequence of this, the closest suitable location for these students to be given a bus stop is stop 239, situated an average of 714 metres away from the students. This issue also leads to the maximum walking distance in the sample dataset, as one student must walk 2105 metres to get to this stop.

OpenStreetMap contributors also underutilise the sidewalk tag. There are several examples in the area of the sample dataset of ways that visibly feature a sidewalk that is not represented in the tags in any form. Figure 3.4.2 is an example of this, as Larry Uteck Boulevard, a road that can be seen on Street View to feature a sidewalk, has no sidewalk tag in OpenStreetMap. This issue caused the designed sidewalk checking system to be inconsistent, a feature that, at best, could only be used as a guideline.

4. IMPROVEMENTS TO ROUTE EFFICIENCY

This chapter discusses the methods developed to improve route efficiency. The aims of the developed techniques and their implementation are explained. The resulting improved solution is then compared against a solution that has not undergone these techniques to analyse its effectiveness.

4.1 Problem Description

The success of Good Travel Software's solutions is based in their pursuits of optimal efficiency. By making fractional reductions in certain parameters, GTS can provide solutions that greatly reduce operating costs for their customers. For example, minimal reductions in total distance travelled by buses while operating the commute can, over the course of the school year, amount to significant savings in fuel costs for the client.

GTS's routing software takes in the locations and demands of bus stops, and outputs a close-to-optimal set of routes to complete the task. In the context of this problem, Good Travel Software would like to investigate methods that can be implemented into their clustering and routing methods to reduce route complexity, increasing efficiency, and reducing costs by minimising bus travelled distance. The improvements have been focused on the bus stop creation process, aiming to develop techniques to move bus stops within the walking distance constraints to improve the overall solution.

Chapter 3 described the new distance constrained clustering method used to create a suitable set of bus stops within the maximum permissible walking distances provided by Good Travel Software's client. While the solutions produced by these techniques produced an adequate solution that obeyed the constraints where possible, it became apparent through investigation that there was scope within the provided constraints to find solutions that greatly improved routing efficiency. This problem involved investigating improvements to achieve this goal, aiming to adapt the bus stop location process to factor in routing optimality to the final solution.

The techniques developed to achieve this goal are described in Section 4.3, with the improved process, implemented on the sample dataset from Halifax, Canada, compared to the original solution in Section 4.4.

4.2 Problem Background

The Capacitated Vehicle Routing Problem

The vehicle routing problem was first introduced in a paper called "The Truck Dispatching Problem" by Dantzig and Ramser in 1959 (Dantzig, 1959). The problem discussed in that paper involved optimising the routing of a fleet of gasoline delivery trucks between a terminal and several service stations, but the methods and procedures introduced in that paper have been studied and developed into highly sophisticated metaheuristics (Laporte, 2009). The most famous version of the vehicle routing problem is known as the "Travelling Salesman" problem and involves only one 'salesman' or vehicle visiting a set of points or nodes and returning to the starting node in the most optimal manner. The vehicle routing problem is a generalised form of this problem that features several vehicles. The introduction of several vehicles makes the problem far more computationally intensive. Exact solutions can be achieved for small vehicle routing problems, but heuristic algorithms are used in practice (a

heuristic is an approach that is not guaranteed to be optimal, but is sufficient as an immediate approximation).

In this situation, the routing task is formulated as a capacitated vehicle routing problem. The nodes are the bus stops, and their demands are the number of students to be picked up at each stop. The school buses provided by the school district have a maximum capacity of 70 students. The school buses visit the bus stops in as optimal a route as possible, collecting students along the way, and returning to the 'depot' or school at the end. The aim of the vehicle routing problem is to find the most optimal solution, in this case taking optimality to mean least total distance. The package chosen by Good Travel Software to solve this capacitated vehicle routing problem is Google OR-Tools.

Google OR-Tools

The Operations Research and Optimisation team at Google develop and provide a set of optimisation tools called OR-Tools. This open-source software suite is one of the most powerful tools in the field of problem solving, featuring a selection of solvers for linear programming, graph algorithms, vehicle routing problems and knapsack algorithms (Doodlers, 2019).

Good Travel Software uses the Python wrapper for OR-Tool's vehicle routing problem constraint solver. The constraint solver is provided with a data model featuring a distance matrix of the stops, the number of available vehicles, the capacities of the vehicles, and the number of students at each stop. OR-Tools then returns a close-to-optimal set of bus routes. Each bus route contains an ordered set of stops, starting and ending at the school, with the cumulative distance, cumulative time, and cumulative demands (in this case, number of students on the bus) denoted at each stop. This process is depicted in Appendix C.3, p. C.5.

4.3 Ideas and Implementation

As mentioned in Section 4.1, the ideas created to improve routing efficiency focus on the changing placement of the bus stops to more suitable locations. Several methods have been developed to achieve this goal.

Road Priority System using OpenStreetMap Data

The road priority system is built directly into the bus stop creation system, in busStopCheck.return_suitable_location_outSearch(). The principle of the priority system is simple. As detailed in Section 3.4, the hypothetical bus stops created by the distance constrained clustering process are snapped to the road network using the Overpass API. This process involves searching around the co-ordinates of the hypothetical bus stop for OpenStreetMap way objects. The road priority system aims to guide the placement of the bus stop within this search distance.

As explained in Section 3.4, ways suitable for bus stops in the OpenStreetMap data feature highway tags of either 'residential', 'tertiary', 'secondary', or 'primary'. These roads have a natural hierarchy; primary roads are the biggest, followed by secondary, then tertiary, and finally residential. The road priority system uses this natural hierarchy to guide the placement of stops when multiple ways are returned within the search distance. The result of this process means that, where possible, bus stops are naturally shifted upwards onto more efficient roads.

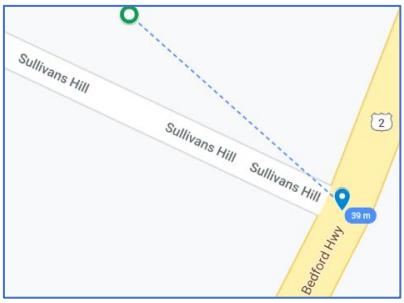

FIGURE 3.4.1 – Priority System in Action

An example of this is contained in Figures 4.3.1 and 4.3.2. Without the priority system, the unsnapped bus stop would be moved straight to Sullivans Hill, a road with a residential tag. However, the priority system suggests moving the bus stop to Bedford Highway, a primary road only 39 metres away. As can be seen from the Google Street View screenshots, this is a prudent decision, as Bedford Highway is a better location for a bus stop

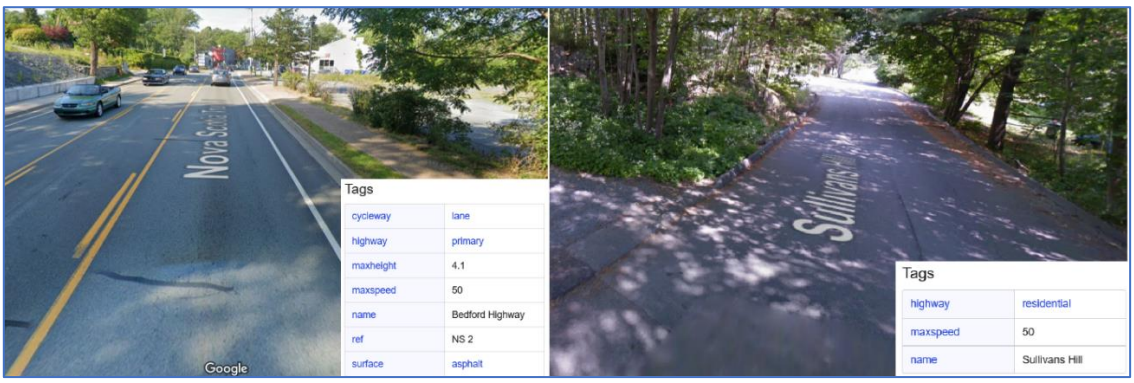

FIGURE 3.4.2 – Comparison of Bedford Highway and Sullivans Hill

Stop Amalgamation

The stop amalgamation process is contained in the clustering.stop_amalgamation() function. Its aim is to improve routing efficiency through reducing the number of bus stops by migrating students to the most efficient bus stop within their walking distance. It takes four inputs: the students, bus stops, an inter-stop distance matrix and a student-to-stop walking matrix. The general principle of the stop amalgamation process is that stop A is placed more efficiently for the routing process than stop B if the average distance between stop A and every other stop in the inter-stop distance matrix is lower than the average distance between stop B and every other stop in the matrix. This assumption was discussed with the client.

The stop amalgamation process operates in a loop. It evaluates each stop individually, gathering all the students that have been assigned to that stop. It then examines each student individually, checking which stops are within the permissible walking distance for that student using the walking matrix. If a stop is found that is within the students' walking distance, its average distance to the rest of the stops in the distance matrix is compared to that of the original stops. If the newfound stop has a lower average distance, the student will be migrated to the new stop, provided their original stop can be completely vacated in the amalgamation process. If the original stop cannot be vacated, it is not worth making students walk further to a different stop when the bus must still visit their original stop.

The benefits of this process are best described using an example like the one in Figure 3.4.3. Before stop amalgamation, St. George's Boulevard contained 4 bus stops within a 500 metre stretch of road to collect 7 students. After the stop amalgamation process, 2 stops were able to be completely vacated and dropped from the solution set. While three students are now walking further to get to their stop, having fewer bus stops will improve overall efficiency.

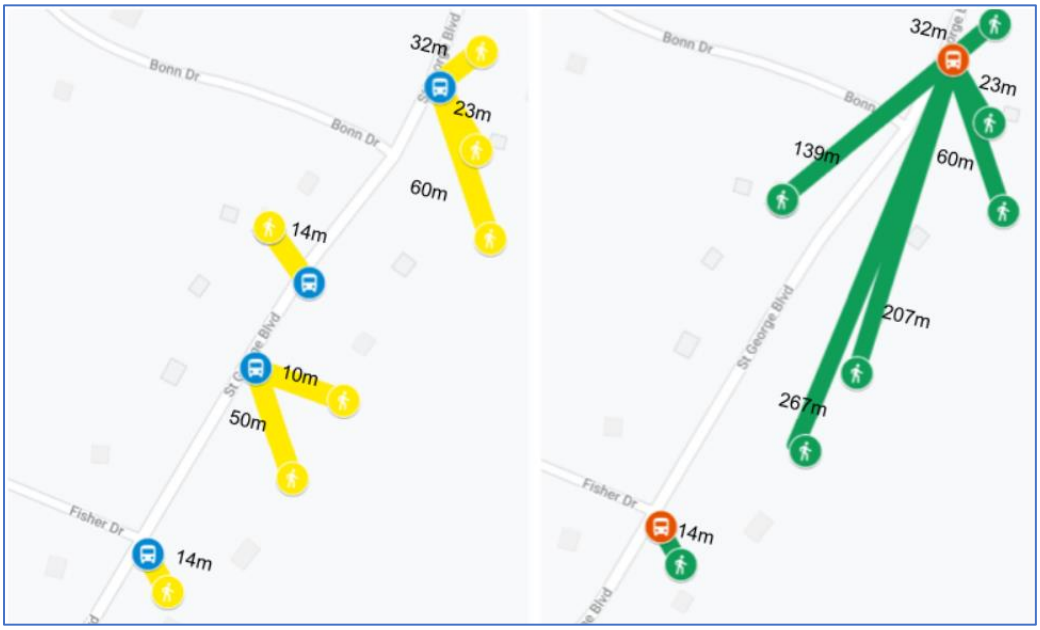

FIGURE 3.4.3 – St George Boulevard before (L) and after (R) stop amalgamation, with student walking distances and assigned stops illustrated.

Furthest Point from Turning Circle System

The furthest point from turning circle system is another augmentation that slots directly into the busStopCheck.return_suitable_location_outSearch method. Turning circles are contained in the OpenStreetMap as nodes with the highway=turning_circle tag, and represent widened, circular areas at the end of residential cul-de-sacs where cars can turn. This method makes use of turning circles to detect cul-de-sacs in residential roads.

If the Overpass API query returns a turning circle, a flag is raised. The road priority system continues until the final stage, usually where the closest point to the initial queried location on the most suitable road is calculated. If the most suitable road is residential, and a turning circle has been detected, then instead of finding the closest point to the query (which may be on a cul-de-sac), the furthest point from the turning circle is returned instead. This furthest point from the turning circle will be the point where the cul-de-sac meets the rest of the road network, ensuring that the bus stop is placed in a more optimal location, avoiding the cul-de-sac altogether.

4.4 Analysis of Test Solutions

This section will compare two solutions across several statistics. The first solution will be produced on the sample dataset in Halifax, Canada using the methods outlined in Chapter 3. The second solution will take the first solution's bus stops and apply the stop amalgamation process developed in Section 4.3. The road priority and furthest point from turning circle systems are built into the Overpass API query system and were used for both sets. The two solutions will be compared across several relevant statistics. A brief description of the two solution sets can be seen in Table 4.4.1.

Table 4.4.1 – The Base Solution without route efficiency improving techniques compared to the Amalgamated Solution

	Base Solution	**Amalgamated Solution**
Number of Bus Stops	251	199
Average Walking Distance	126.33m	161.30m
Total Student Walking Distance	92.095km	117.589km
Number of Students walking over 400 metres	24	24
Average Student Travel Time*	28 mins	25 mins
Total Student Travel Time *	337 hours	307 hours
Average Bus Route Length	34.696km	32.077km
Total Bus Travel Distance	381.644km	352.854km
Average Bus Travel Time	52 mins	48 mins
Total bus Travel Time	9 hours 34 mins	8 hours 51 mins
Number of Buses Used	11	11
Most Empty Seats on a Route	16 empty seats	19 empty seats
Average Number of Students at a Bus Stop	3.01	3.71
* Student travel time = time taken while walking + time spent on bus		

Description of Sample Data

The sample dataset consisted of 729 students living in Halifax, Canada attending Charles P Allen High School. The students' student numbers, home co-ordinates and grades were provided. The student numbers were used as unique identifiers for each student. The grades were unused for this project, as all students fell within the same walking distance constraint. The maximum permissible walking distance for this dataset was 400 metres. This was decided by the school district.

The latitudes of the students' home addresses ranged between 44.692° and 44.782° North. The longitudes ranged between 63.643° and 63.856° West (West longitudes are conventionally represented as negative numbers). There are 638 unique addresses, with some addresses housing multiple students. 558 addresses have only one student, 75 addresses house two students and there are two addresses with three students. The remaining three addresses have four, five and six students.

Subsetting of Data for Walking Distance Distribution

A tagging issue in the OpenStreetMap data, explained in Section 3.7, led to a situation where several students were forced to walk considerable distances above the maximum walking distance constraint to get to their bus stop. These students are all assigned to stop 239. The students from this stop have been excluded from the student walking distance distribution that appears later in this section. The reasons for this are twofold; these students represent an outlier in the distribution, and their presence does not accurately represent the results of the designed process. Including them in the graph renders the visualisation unreadable, reducing the weight of the conclusions that can be drawn from the graph. This is visualised in Appendix F.1, p. F.1.

Overall View of Both Solutions Using Visualisation software

Figure 4.4.2 contains the routing solutions produced before and after stop amalgamation using Good Travel Software's visualisation tool. There is a noted reduction in route complexity, with fewer stops present in the amalgamated solution leading to more streamlined routes.

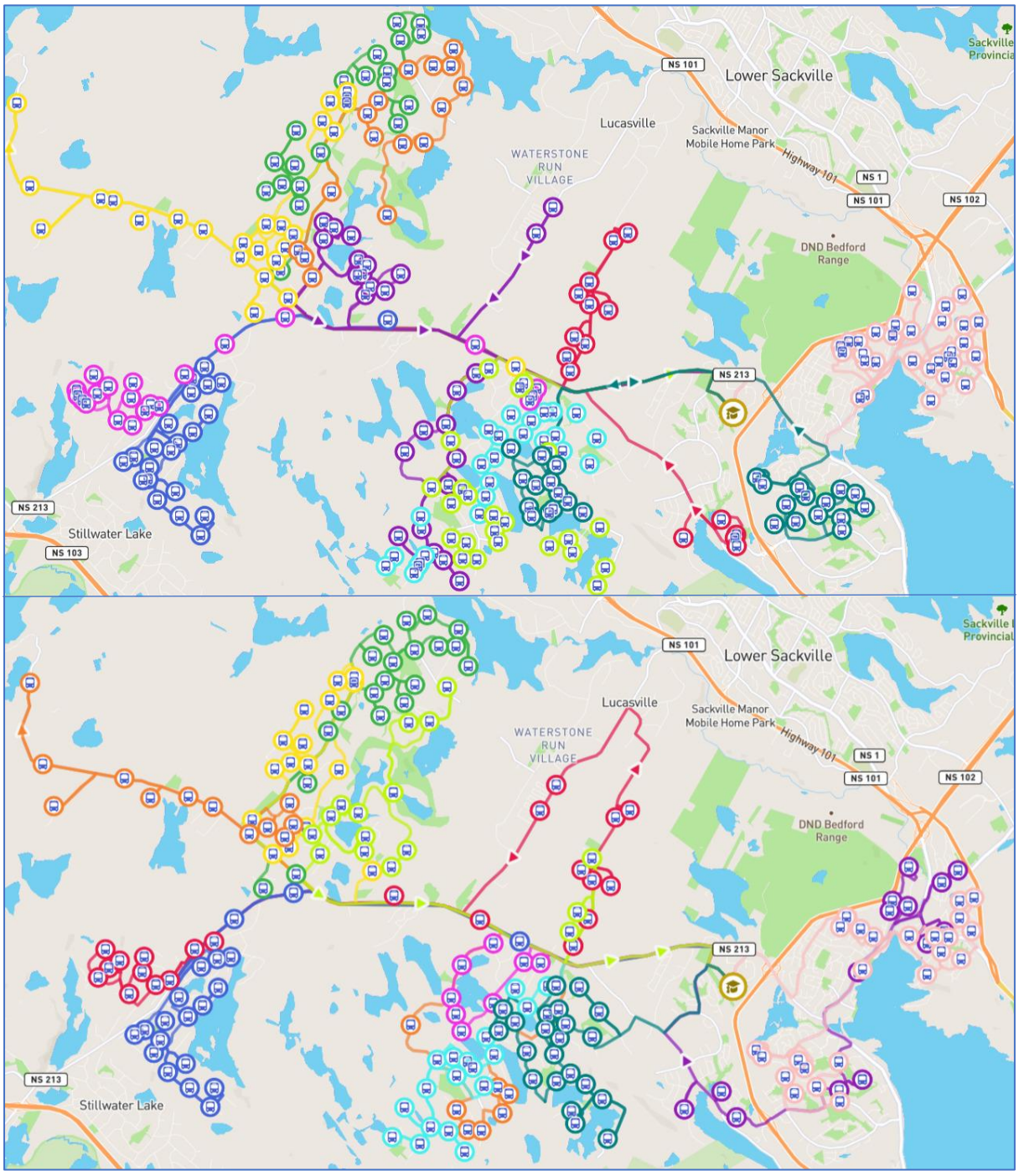

FIGURE 4.4.2 – A visualisation of the base solution (top) and the amalgamated solution (bottom) using Good Travel Software's visualisation tool

Comparison of Bus Routes

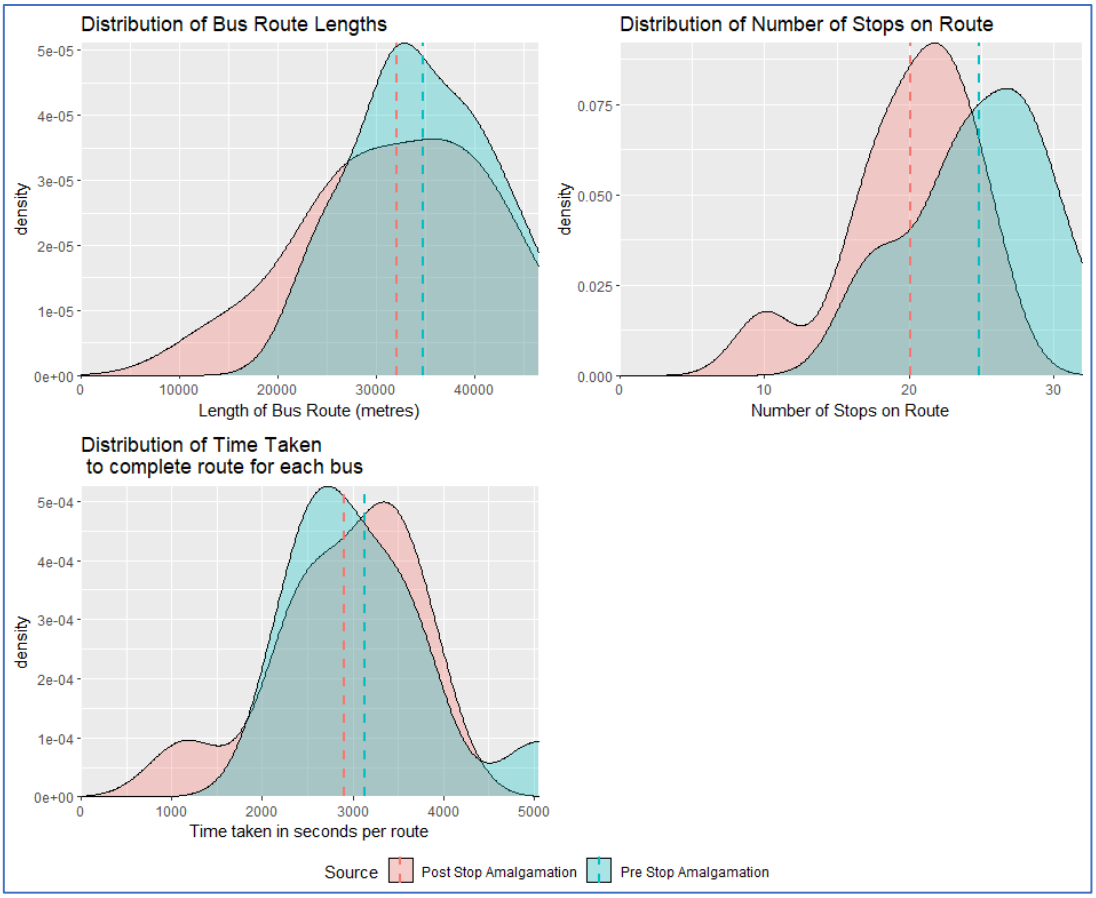

FIGURE 4.4.3 – Comparison of bus routes in base solution (pre stop amalgamation) and amalgamated solution (post stop amalgamation) across several metrics: route length, number of stops and time taken.

In Figure 4.4.3, the bus routes in the amalgamated solution and the base solution are compared. The top left graph is a distribution of the length of the 11 produced bus routes. The amalgamated solution produced an average bus route length of 32.077 kilometres while the base solution's routes were 34.696 kilometres. This reduction of 2.619 kilometres per route, while small, scales into a reduction of 28.79 kilometres in total distance.

The top left graph shows the number of stops on each route. The amalgamated solution's routes stop at an average of 20 stops each compared to 24.8 stops in the base solution. This reduction in stops means that student travel times are faster, routes are shorter and travel time is quicker. This can be seen in the final graph in the bottom left, where the distribution of time taken per route is displayed. The amalgamated routes are slightly quicker on average, taking 2898 seconds (48 minutes 18 seconds), as opposed to 3134 seconds (52 minutes 14 seconds) in the base solution. These marginal improvements lead to a 43-minute reduction in total bus travel time in the amalgamated solution. From these graphs, amalgamated routes are shown to be shorter, quicker, and less complex than the results of the base solution.

Travel Time and Walking Distances

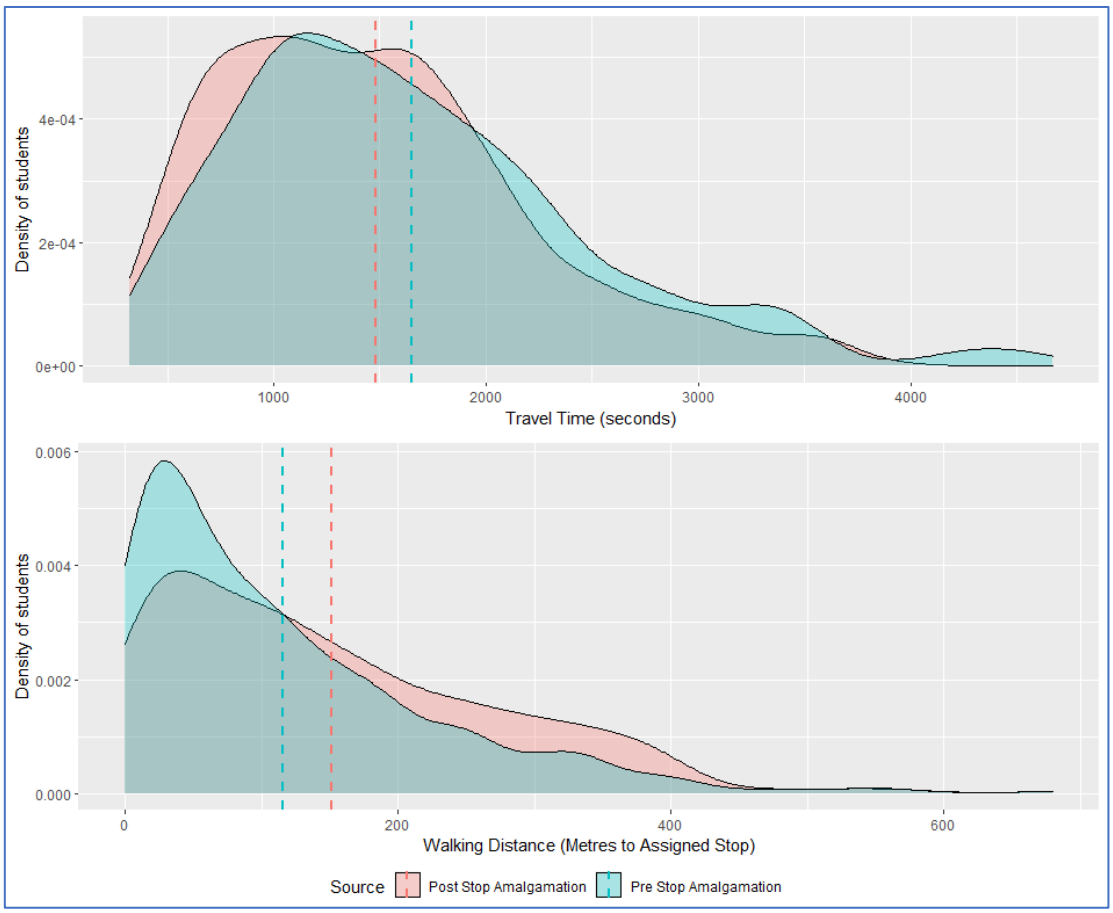

FIGURE 4.4.4 – Comparison of travel time (top) and walking distance (bottom) for the students in the base solution and amalgamated solution.

The graphs in Figure 4.4.4 examine the changes between the two solutions for the students. The top graph shows the distribution of travel times. Travel time per student in this graph is calculated by adding the students walking time to the time that the bus takes to get from that student's stop to the school while stopping at the rest of the stops on the route. The average student travel time in the amalgamated solution is 1,481 seconds (24 minutes and 41 seconds) and is 1,650 seconds (27 minutes and 30 seconds) in the base solution. This improvement, while small when examined per student, leads to total student travel time being reduced by 30 hours, 25 minutes, and 11 seconds.

This improvement in travel time comes at a cost, namely an increase in average walking distance, as can be seen in the bottom graph of Figure 4.4.4. Students in the pre-amalgamated solution walk on average 126.33 metres to their stop, contrasted with a post-amalgamated average walking distance of 161.30 metres. This is an average increase of 34.97 metres per student. The total student walking distance is increased by 30.97 kilometres in the amalgamated solution. Both solutions have the same number of students violating the walking distance constraint.

5. CONCLUSIONS AND RECOMMENDATIONS

This chapter discusses the discoveries and areas of further exploration for this project. Several recommendations are made about future enhancements and investigations.

5.1 Conclusions

The implementation of a distance-constrained clustering system for stop creation was successfully delivered to the client in the form of a customisable data pipeline. The pipeline can produce bus stops for inputted students that obey the maximum walking distance set by the school district and are placed in suitable, sensible locations that follow configurable parameters and avoid minor roads and cul-de-sacs, in a process outlined in Section 3.4 and 3.5. The bus stops created by that process can then be improved by implementing the route-efficiency-improving techniques described in Section 4.3. The improved bus stops are used to produce close-to-optimal bus routes using OR-Tools. The final routes from the pipeline can be outputted in a suitable csv format, that can be immediately plugged into Good Travel Software's visualisation tool.

The efficiency-improving techniques developed in Section 4.3 have been shown to improve the quality of produced bus routes in student travel time and total bus distance travelled on a sample dataset in Halifax, Canada in Section 4.4. This improvement comes with a trade-off in increased student walking distance; however, these students are still all walking within the permissible walking distance constraint.

5.2 Recommendations

Generalise Bus Stop Location Query

The Overpass API query uses community sourced OpenStreetMap data. While the OpenStreetMap community strives to ensure consistent tagging across the world, there may be instances where localised tagging conventions occur. As the query designed for this project was made solely with the sample dataset in Halifax, Canada, it may be overly specialised to the idiosyncrasies of the Halifax OpenStreetMap community. It would be prudent to generalise this process by testing it on other, varied municipalities.

OpenStreetMap Proposals

In this project, the presence of a turning circle tag was used as an indicator of a cul-de-sac (explained in Section 4.3). However, the OpenStreetMap community notes that on occasion turning circles can be placed at the end of residential cul-de-sacs where further development is planned (OpenStreetMap, Tag:highway=turning_circle, 2021). This may lead to situations where turning circles do not denote the end of residential cul-de-sacs. It may be in the interest of Good Travel Software to submit a proposal for the adoption of a 'cul-de-sac' tag on OpenStreetMap data to improve this feature.

As OpenStreetMap is always accepting new proposals, it is necessary to monitor tagging convention to make sure the Overpass API query remains relevant in the future. For example, there are several proposals concerning the sidewalk tagging convention that may become relevant soon (OpenStreetMap, 2021).

TRINITY COLLEGE DUBLIN

Management Science and Information Systems Studies

Project Report

GOOD TRAVEL SOFTWARE

Clustering in School Bus Route Optimisation

April 2021

APPENDICES

Prepared by: Diarmuid Coffey ***Supervisor: Simon Wilson***

A. ORIGINAL PROJECT OUTLINE

Client:	Good Travel Software
Project:	Clustering in School Bus Route Optimisation
Location:	Digital Court, Rainsford Street, Dublin 8
Client Contact:	Peter Soutter
Dept. Contact:	Simon Wilson (swilson@tcd.ie)

Client Background

Good Travel Software was formed in 2010. It is based in the Digital Hub in Dublin. It develops software solutions for the short term car rental market. It sells a web-based reservation, payment and management system for this business and is also developing a suite of data analytics solutions that help to solve some of the logistical problems that arise in running these schemes.

Project Background

Good Travel Software works with municipalities, and other organisations that manage large public transport system, on optimal routing for shuttle buses. A good example is a school district in North America that has to collect students from their homes each morning and get them to their school by the start of classes, then return them home at the end of the day, and wants to do this in the most time-efficient way with the smallest number of buses. GTS has developed a suite of routing tools that are given the addresses of where to pick up and drop off passengers, and a fleet of buses, and propose routes that satisfy these demands in as good a way as possible.

Client Requirement

One important issue in school bus routing is the location of bus stops where more than 1 student can be picked up and dropped off. Doing this, rather than visiting each student's house, can make the bussing substantially quicker and more efficient. The task is to identify good bus stop locations that (a) are usually not on very minor roads or cul-de-sacs (b) may have to satisfy some constraints on how far a student can walk to a stop (c) have a pavement that the student can use. This project is to look at how we might use clustering algorithms, as well as information from mapping and directions APIs, to produce a solution to the location of the bus stops

What is involved for the student?

- Coding to interface with data coming from the routing software as well as implement clustering and interaction with APIs. This will make use of Python.
- Proposing how to use the information in the APIs to identify whether a location is in a cul-de-sac or a very minor street.

B. INTERIM PROJECT REPORT

Project: Clustering in School Bus Route Optimisation

Client: Good Travel Software

Student: Diarmuid Coffey

Supervisor: Simon Wilson

Review of Background and Work to Date

Good Travel Software was formed in 2010. It develops software solutions for the short-term car rental market and works with municipalities, and other organisations that manage large public transport system, on optimal routing for shuttle buses. A good example is a school district in North America that has to collect students from their homes each morning, get them to their school by the start of classes, and return them home at the end of the day. GTS has developed a suite of routing tools that are given the addresses of where to pick up and drop off passengers, and a fleet of buses, and propose routes that satisfy these demands in as optimal a way as possible. GTS are looking for support in developing techniques to identify good bus stop locations where more than 1 student can be picked up and dropped up to improve efficiency.

To date, I have:

- Met with my supervisor who is uniquely informed on the project background,
- Performed preliminary clustering investigations on provided datasets,
- Identified and begun familiarising myself with a number of relevant APIs and software that may prove useful throughout the investigation, namely Graphhopper, Google OR-Tools and OpenStreetMaps.

Terms of Reference

Based on the initial project brief and subsequent investigation and discussions with the client, the project's terms of reference have been agreed as follows:

1. To research and investigate relevant clustering algorithms and assess their performance for the placement of bus stops on sample data.
2. To perform distance constrained clustering to limit walking distances for certain categories of students.
3. To perform vehicle routing through the selected bus stops and develop techniques to simplify the route and ensure that the bus stops are not placed on very minor roads or cul-de-sacs.
4. To investigate techniques to identify whether a selected location has a pavement.

Further Work

- Research any clustering algorithms that may suit the data and evaluate their performance on the placement of bus stops. (December/January)
- Research and perform distance constrained clustering (January)
- Solve the resulting vehicle routing problem and develop techniques to simplify any resulting inefficiencies and avoid U-turns and cul-de-sacs. (January/February)

- Investigate pavement recognition and comprehensively test algorithms and techniques. (February)
- Write up report. (February/March)

Conclusions

Upon discussion with my supervisor, pavement recognition will only be pursued should time permit

C. USER DOCUMENTATION

C.1 Required Files

The constituent files for the pipeline are provided in a zip folder entitled *dicoffey.zip*. It contains three python scripts:

- *clustering.py* – a file containing all the clustering methods
- *routing.py* – contains all the routing methods, and the calls to the GraphHopper API
- *busStopCheck.py* – contains all the methods used to retrieve an Overpass API query and process it to check if a location is suitable for a bus stop.

C.2 Configuration Settings

Clustering

```
"""
Created on Wed Mar 10 12:40:56 2021

@author: Diarmuid
This file features the methods needed to perform the distance constrained clustering, assign students to their
stops and improve the solution
"""

from sklearn.cluster import KMeans
from kneed import KneeLocator
import geopy.distance
import numpy as np
import busStopCheck
import time

import routing as route

MAX_DISTANCE = 400
MAX_SEARCH_DISTANCE = 200
```

FIGURE C.2.1 – the configuration variables in clustering.py

The clustering.py file has been configurated to the specifications provided by the Halifax school district. The maximum permissible walking distance is 400 metres. To adjust this, the MAX_DISTANCE variable should be adjusted to the new maximum walking distance.

The MAX_SEARCH_DISTANCE variable controls how far the Overpass API query can look for ways around the query co-ordinates. This can also be adjusted.

Routing

```
"""
Created on Tue Mar 9 11:12:32 2021

@author: Diarmuid
This file contains the methods needed to obtain distance matrices from GraphHopper and to perform the OR-Tools
Routing
"""
import pandas as pd
import numpy as np
import time
import requests
import json
from ortools.constraint_solver import pywrapcp, routing_enums_pb2
import datetime

DEPOT_CO_ORDINATES = [44.72048, -63.69856]
GRAPHHOPPER_MATRIX_URL = "https://graphhopper.com/api/1/matrix?key=4e2c94b1-14ff-4eb5-8122-cacf2e34043d&ch.disable=true"
VEHICLE_CAPACITY = 70


def graphhopper_matrix_depot(bus_stops):
```

FIGURE C.2.2 – The configuration variables in routing.py

In routing.py, the depot co-ordinates should be set to the co-ordinates of the destination school. GRAPHHOPPER_MATRIX_URL is used to send queries to the GraphHopper Distance Matrix API. The string of characters between 'key=' and '&ch.disable=true' is the API key. This particular API key refers to a free GraphHopper account. Should the user seek to upgrade to a premium tier, the new API key provided for this account should be inserted into this space. The VEHICLE_CAPACITY variable refers to the number of students that can be carried by each school bus. School buses are conventionally 70 seaters, but this variable can be changed should a school district require it.

Bus Stop Check

```
"""
Created on Tue Feb  9 01:24:50 2021

@author: Diarmuid
This file contains the checking methods used to query the Overpass API and process the response to assess
whether a location is suitable for a bus stop
"""

import pandas as pd
import numpy as np
import json  # to import json
import requests  # to import requests
import time
import geopy.distance
from scipy.spatial import KDTree

# Distance from coordinate to road allowed
DISTANCEALLOWED = 50
OVERPASS_URL = "http://overpass-api.de/api/interpreter"
```

FIGURE C.2.3 – The configuration variables in busStopCheck.py

The final set of configuration variables can be seen in Figure C.2.3. the DISTANCEALLOWED variable denotes the initial search distance for the Overpass API to use when evaluating the suitability of bus stop locations. The Overpass API URL is contained in OVERPASS_URL. As

the Overpass service is entirely free, this URL should not need to be changed as Good Travel Software adjusts its system to service new clients.

Implementation

The 3 files that make up this project (clustering.py, routing.py and busStopCheck.py) have been designed to provide new techniques to the Good Travel Software system. The methods have been designed to operate as standalone functions, with clear inputs and outs, to expedite the implementation process. The files should be included within the Good Travel Software system. Good Travel Software can then make use of whichever method they need in their scripts by calling the function.

C.3 System Diagrams

The entire Pipeline (Part one)

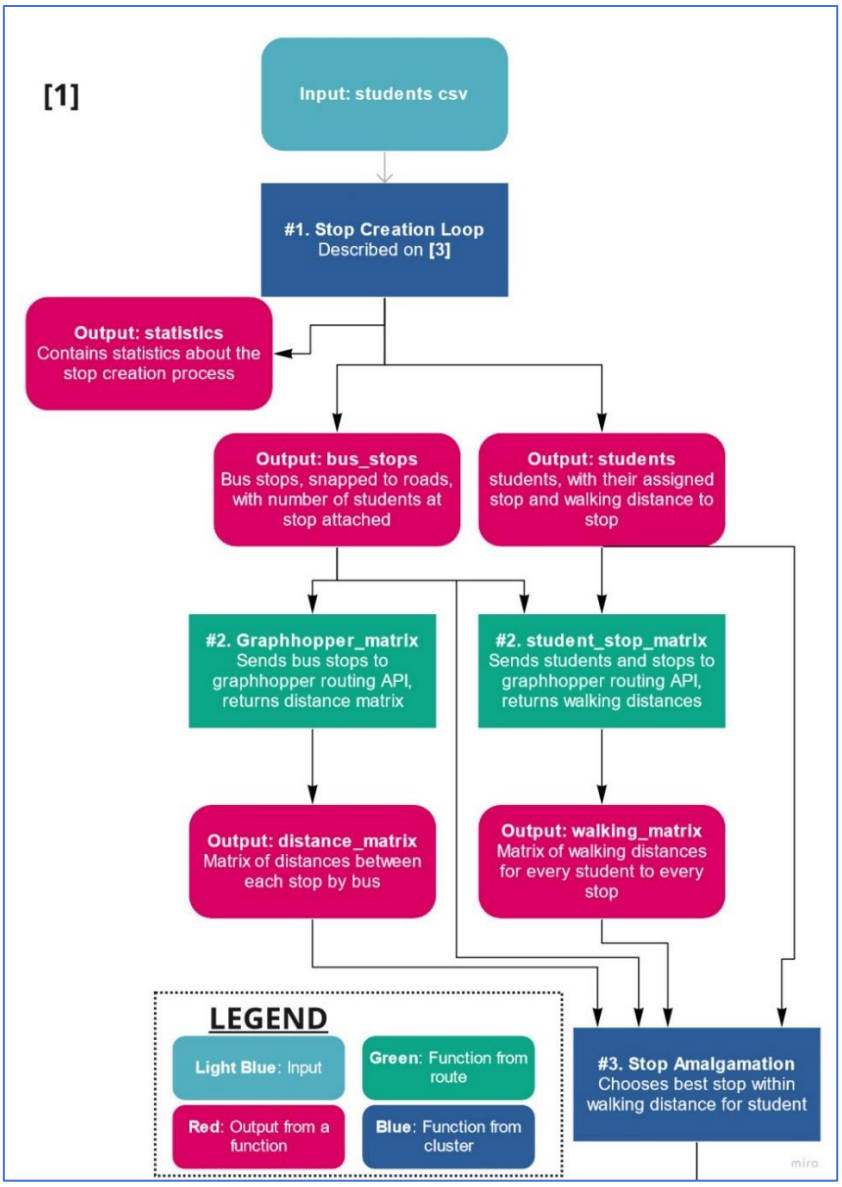

FIGURE C.3.1 – Overall System Diagram, part one (continued on next page)

The Entire Pipeline (Part Two)

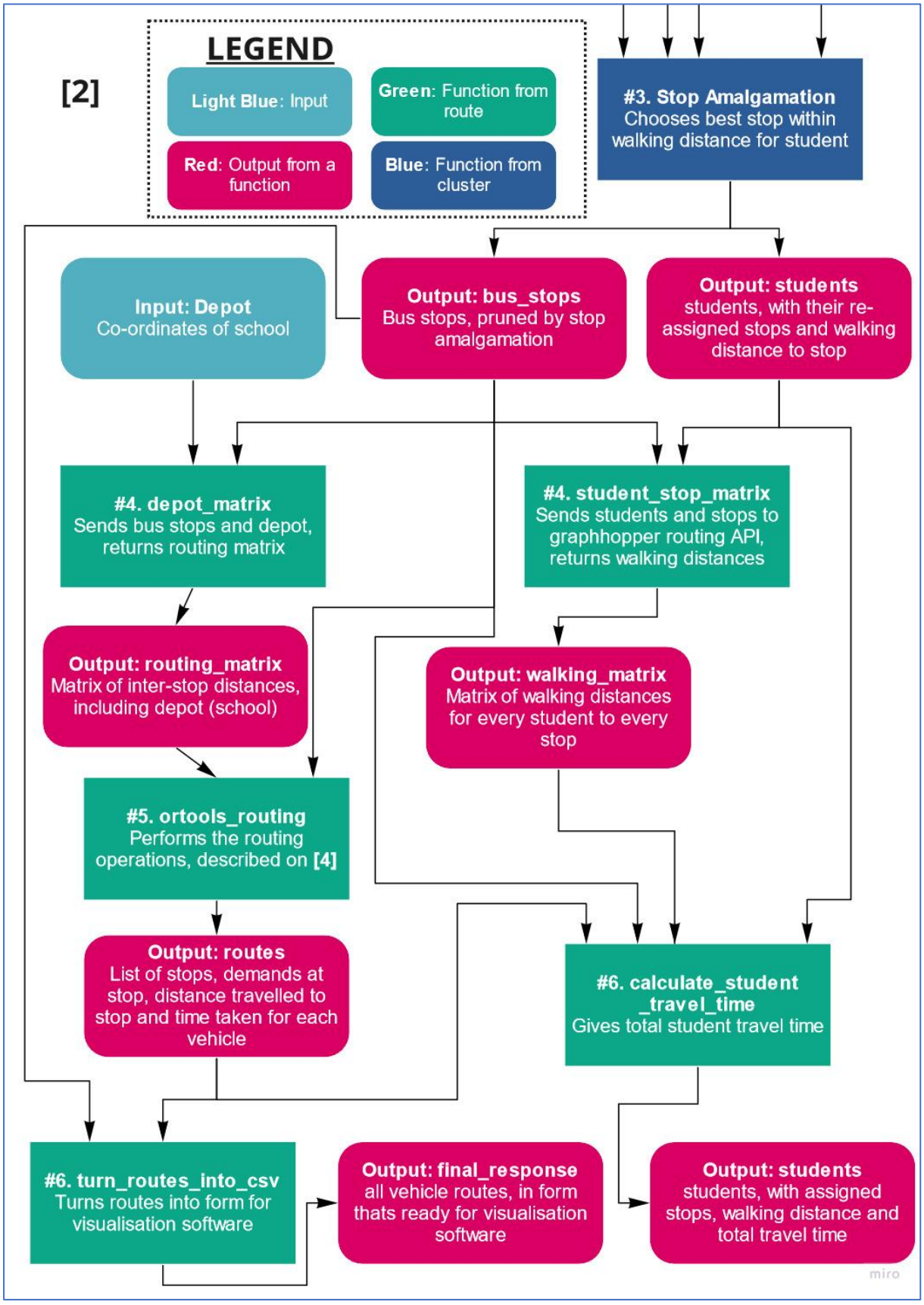

FIGURE C.3.2 – Overall System Diagram, part two

The Stop Creation Loop

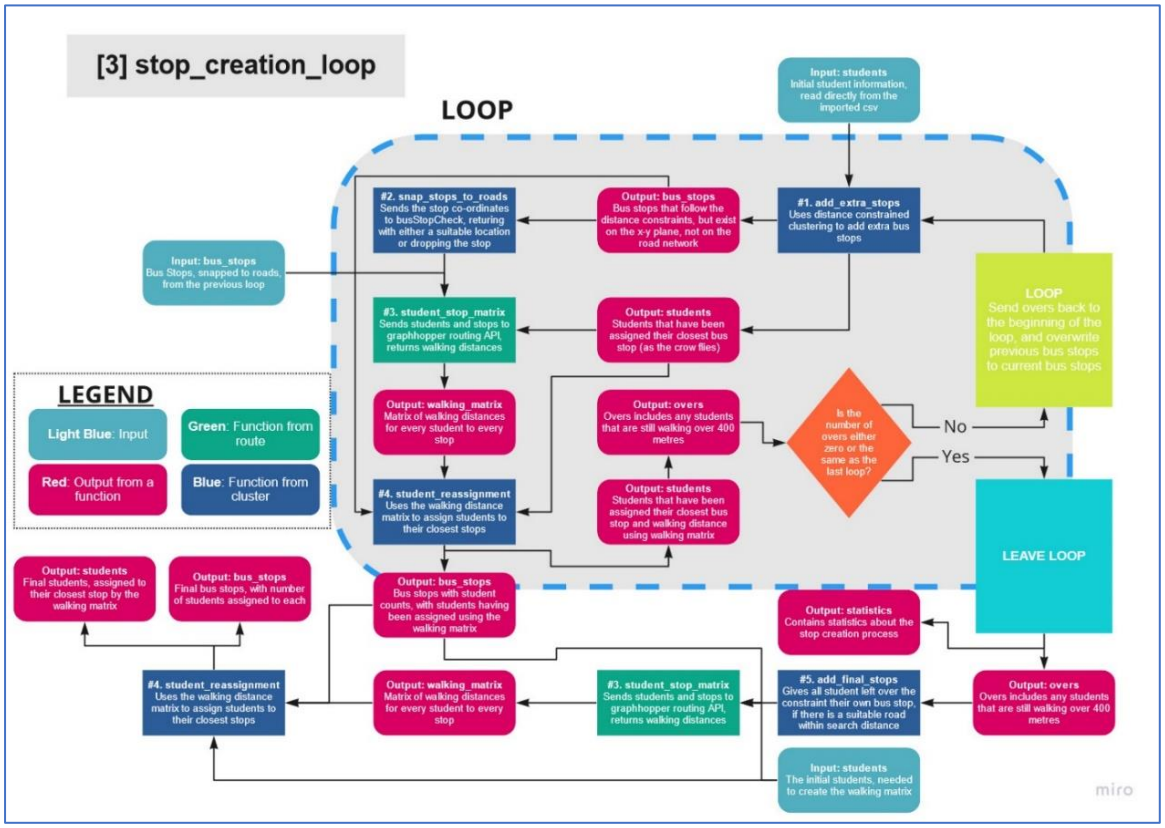

FIGURE C.3.3 – Stop Creation Loop

The OR-Tools Routing Process

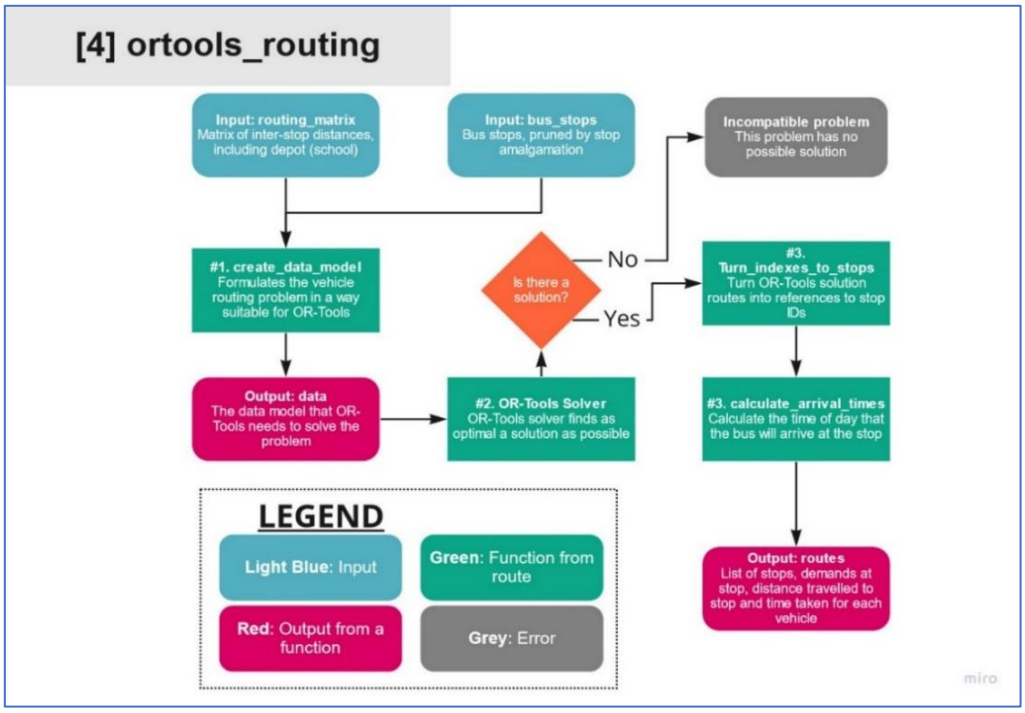

FIGURE C.3.4 – The OR-Tools Routing process

D. INPUTS AND OUTPUTS

D.1 Descriptions of the Data Frames

The final 'routes' output

vehicle_id	sequence	latitude	longitude	Vehicle_cumul_dist	cumul_demands	arrival_time

- **vehicle_id**: A unique identifier for each vehicle.
- **sequence**: Represents the order in which the stops are visited.
- **latitude**: Represents the latitude of the bus stop being visited.
- **longitude**: Represents the longitude of the bus stop being visited.
- **vehicle_cumul_dist**: The cumulative distance travelled by the bus before reaching this bus stop.
- **cumul_demands**: The amount of students on the bus after leaving this bus stop.
- **arrival_time**: The time that the bus will arrive at this stop.

The final students data frame

Student Number	Lat	Lon	Assigned Stop	Walking Distance	Travel Time (s)

- **Student Number**: A unique identifier for each student, supplied in the input.
- **Lat**: The latitude of the student's home address.
- **Lon**: The longitude of the student's home address.
- **Assigned Stop**: The unique ID of the bus stop that the student has been assigned to.
- **Walking Distance**: The distance, in metres, that the student must walk to get to their assigned bus stop.
- **Travel Time (s)**: The total time in seconds that the student must travel, timed from the point they leave their house to walk to their bus stop to the time their bus arrives at school.
-

The final bus stops data frame

Stop ID	Lat	Lon	Distance Moved	Road Type	Students at Stop	Average Walking Distance

- **Stop ID:** A unique identifier for each bus stop, starting from 101.
- **Lat:** The latitude of the bus stop.
- **Lo**n: The longitude of the bus stop.
- **Distance Moved:** The distance between the final bus stop location and the hypothetical location produced by the clustering process.
- **Road Type:** The type of road that the stop is situated on: residential, tertiary, secondary and primary.
- **Students at Stop:** The number of students that will be picked up at this bus stop.

Average Walking Distance: The average distance that the students being picked up at this stop must walk.

D.2 Datasets

The top 15 entries in each dataset have been provided for reference, with the rest being available in the zipped folder accompanying this project.

Sample Input Dataset – 6670, Halifax, Canada

Student Number	Lat	Lon	Grade
297121	44.70069	-63.7615	11
297071	44.70069	-63.7615	12
291748	44.69832	-63.7543	10
289805	44.70241	-63.7573	11
289066	44.69881	-63.7554	10
286047	44.69981	-63.7504	10
282231	44.69749	-63.759	11
272593	44.70492	-63.7537	12
271263	44.70425	-63.7555	12
270628	44.70551	-63.7543	11
317751	44.70009	-63.751	11
317090	44.69826	-63.7554	12
316170	44.69978	-63.7609	12
314037	44.70657	-63.7605	12
313002	44.70106	-63.7553	12
312331	44.7006	-63.7602	10

Output Data frame – students - pre-amalgamation

Student Number	Lat	Lon	Assigned Stop	Walking Distance (m)	Travel Time (s)
297121	44.70069	-63.7615	261	9	860
297071	44.70069	-63.7615	261	9	860
291748	44.69832	-63.7543	310	100	1037
289805	44.70241	-63.7573	210	351	1635
289066	44.69881	-63.7554	310	155	1076
286047	44.69981	-63.7504	209	19	1023
282231	44.69749	-63.759	203	17	959
272593	44.70492	-63.7537	210	60	1426
271263	44.70425	-63.7555	210	76	1437
270628	44.70551	-63.7543	210	94	1450
317751	44.70009	-63.751	209	40	1038
317090	44.69826	-63.7554	310	81	1024
316170	44.69978	-63.7609	261	105	930
314037	44.70657	-63.7605	236	10	774
313002	44.70106	-63.7553	209	391	1291

Output Data Frame – students – post-amalgamation

Student Number	Lat	Lon	Assigned Stop	Walking Distance (m)	Travel Time (s)
297121	44.70069	-63.7615	261	9	733
297071	44.70069	-63.7615	261	9	733
291748	44.69832	-63.7543	310	100	755
289805	44.70241	-63.7573	210	351	2199
289066	44.69881	-63.7554	310	155	794
286047	44.69981	-63.7504	209	19	653
282231	44.69749	-63.759	261	399	1014
272593	44.70492	-63.7537	210	60	1990
271263	44.70425	-63.7555	210	76	2001
270628	44.70551	-63.7543	210	94	2014
317751	44.70009	-63.751	209	40	668
317090	44.69826	-63.7554	310	81	742
316170	44.69978	-63.7609	261	105	803
314037	44.70657	-63.7605	236	10	1871
313002	44.70106	-63.7553	209	391	921

Output Data Frame – bus stops – pre-amalgamation

Stop ID	Lat	Lon	Distance Moved	Road Type	Students At Stop	Average Walking Distance
101	44.74096	-63.7303	41.66717	residential	4	244.25
102	44.72667	-63.7552	123.962	tertiary	6	72.5
103	44.72427	-63.7418	53.43129	residential	9	231.7778
104	44.70686	-63.8256	5.740899	residential	3	89
105	44.727	-63.7345	25.14267	residential	2	104
106	44.71193	-63.8273	51.29477	residential	2	45
107	44.70849	-63.7582	79.70366	residential	5	44.4
108	44.74586	-63.8032	30.73746	residential	7	87.85714
109	44.77227	-63.775	34.23394	tertiary	1	13
110	44.74928	-63.8512	15.59807	residential	1	1
111	44.72281	-63.6705	39.57015	primary	1	239
112	44.72028	-63.8143	107.8626	residential	1	101
113	44.75594	-63.7941	119.3431	tertiary	3	152
114	44.76465	-63.7949	35.13441	residential	4	84
115	44.7265	-63.8195	25.54181	residential	3	211

Output Data Frame – bus stops – post-amalgamation

Stop ID	Lat	Lon	Distance Moved	Road Type	Students At Stop	Average Walking Distance
101	44.74096	-63.7303	41.66717	residential	4	244.25
103	44.72427	-63.7418	53.43129	residential	15	204.2
104	44.70686	-63.8256	5.740899	residential	3	89
105	44.727	-63.7345	25.14267	residential	2	104
106	44.71193	-63.8273	51.29477	residential	2	45
107	44.70849	-63.7582	79.70366	residential	6	100.5
108	44.74586	-63.8032	30.73746	residential	7	87.85714
109	44.77227	-63.775	34.23394	tertiary	2	241.5
110	44.74928	-63.8512	15.59807	residential	1	1
111	44.72281	-63.6705	39.57015	primary	1	239
112	44.72028	-63.8143	107.8626	residential	1	101
113	44.75594	-63.7941	119.3431	tertiary	3	152
114	44.76465	-63.7949	35.13441	residential	4	84
115	44.7265	-63.8195	25.54181	residential	3	211
116	44.72924	-63.7353	81.40579	residential	3	26.33333

Output Data Frame – routes – pre-amalgamation

vehicle_id	sequence	latitude	longitude	vehicle_cumul_dist	cumul_demands	arrival_time
180-A-9	1	44.7009	-63.7095	4053	2	08:48:45
180-A-9	2	44.70376	-63.7012	5309	9	08:50:57
180-A-9	3	44.70137	-63.6979	5968	14	08:52:22
180-A-9	4	44.70086	-63.6982	6030	22	08:52:30
180-A-9	5	44.69963	-63.6978	6178	23	08:52:49
180-A-9	6	44.72924	-63.7353	11289	26	08:58:43
180-A-9	7	44.73126	-63.7312	11765	28	08:59:44
180-A-9	8	44.73236	-63.7336	12593	30	09:01:30
180-A-9	9	44.73746	-63.7302	13313	37	09:03:03
180-A-9	10	44.73912	-63.7326	13809	38	09:04:07
180-A-9	11	44.73655	-63.7261	14610	39	09:05:50
180-A-9	12	44.74096	-63.7303	15339	43	09:07:24
180-A-9	13	44.74749	-63.725	16224	45	09:08:34
180-A-9	14	44.74863	-63.7218	16678	47	09:09:22
180-A-9	15	44.727	-63.7345	19658	49	09:14:51

Output Data Frame – routes – post-amalgamation

vehicle_id	sequence	latitude	longitude	vehicle_cumul_dist	cumul_demands	arrival_time
180-A-9	1	44.727	-63.7345	4088	2	08:30:42
180-A-9	2	44.73126	-63.7312	4628	4	08:31:51
180-A-9	3	44.73746	-63.7302	5344	11	08:33:23
180-A-9	4	44.73912	-63.7326	5840	12	08:34:27
180-A-9	5	44.73655	-63.7261	6641	13	08:36:10
180-A-9	6	44.74749	-63.725	8217	15	08:38:49
180-A-9	7	44.74863	-63.7218	8671	17	08:39:37
180-A-9	8	44.75266	-63.7383	13008	19	08:45:52
180-A-9	9	44.74856	-63.742	13567	24	08:46:38
180-A-9	10	44.73499	-63.7747	17123	25	08:50:23
180-A-9	11	44.72767	-63.8152	21002	26	08:53:58
180-A-9	12	44.72149	-63.8251	22163	31	08:55:16
180-A-9	13	44.72281	-63.8313	22796	33	08:56:37
180-A-9	14	44.72522	-63.8309	23151	39	08:57:23
180-A-9	15	44.72345	-63.8388	24349	43	08:59:32

E. TEST DOCUMENTATION

This section outlines the testing procedure for the development of this system. All testing was done using the sample dataset of students in Halifax, Canada. It would be prudent to test this pipeline on another dataset.

Unit Testing

The process of unit testing involves evaluating individual components of the system to check if they are functioning properly on their own. As certain steps of the pipeline often built on the outputs of the previous steps, it was not always possible to perform unit testing. Unit testing was performed on the initial bus stop creation process and the location checking process. This was performed in conjunction with Google Map's custom map feature and Google Map's Street View. These features were used to make sure bus stops were placed in sensible locations, and that students were assigned to their closest bus stop. For any features that weren't suited to unit testing, integration testing was performed instead.

Integration Testing

The majority of the methods written for this project were tested using integration testing. The design of the pipeline meant that methods could be tested and then re-tested following the introduction of the next section of the pipeline. As the initial bus stop creation process had been unit tested, the following sections (mainly the stop amalgamation method and the OR-Tools routing process) were incrementally added and tested, again using Google Maps custom maps and Street View, to make sure the results were as expected.

System Testing

System testing was performed once integration testing was concluded. System testing involved testing the whole pipeline to make sure the results were correct, by calling each method in series.

Acceptance Testing

Acceptance testing was performed by transferring the complete pipeline to another machine to ensure consistent results.

F. EXTRA PLOTS

F.1 Justification for dropping Stop 239 in the Walking Distance Visualisations

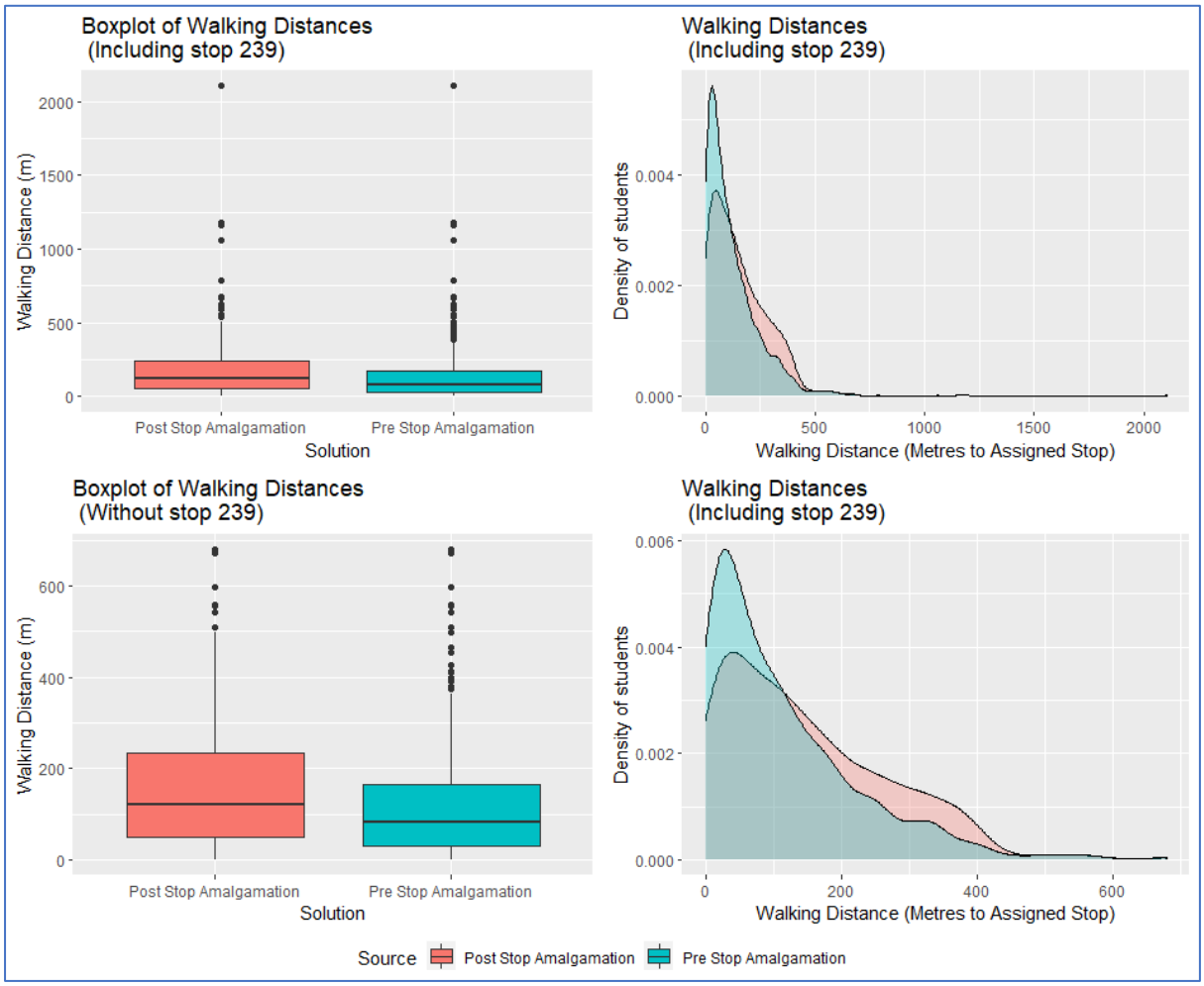

FIGURE G.1.1 – Boxplots and walking distributions before and after dropping stop 239 from the visualisations

G. BIBLIOGRAPHY

API, O. (2021). *The Overpass API's User Manual*. Retrieved from Overpass API: https://dev.overpass-api.de/overpass-doc/en/index.html

Biju. (2008). Agile Software Development. *E-Learning and Digital Media*, 97-102.

Dantzig, R. (1959). The Truck Dispatching Problem. *Management Science*.

Doodlers, D. (2019, February). *Google OR Tools - A Guide*. Retrieved from Medium: https://medium.com/google-or-tools/google-or-tools-a-guide-39f439a5cd0f

Google. (2021). *Distance Matrix API Usage and Billing*. Retrieved from Google Developers: developers.google.com/maps/documentation/distance-matrix/usage-and-billing

GraphHopper. (2021). *Pricing*. Retrieved from GraphHopper: graphhopper.com/pricing/

Laporte. (2009). Fifty Years of Vehicle Routing. *Transportation Science*.

OpenStreetMap. (2021). *Category:Proposed Features Under Way*. Retrieved from OpenStreetMap Wiki.

OpenStreetMap. (2021). *Tag:highway=turning_circle*. Retrieved from OpenStreetMap Wiki.