A Large Incident Management System (IMS) For Intelligent City Development

FINAL REPORT
COMP4801 FINAL YEAR PROJECT

FYP18012
CHOW, YIN CHAK (3035271529)
KWONG, MATTHEW WANG SHUN (3035277030)
YUEN, HOI MAN (3035244526)

SUPERVISOR
DR. CHENG, REYNOLD C.K.

WRITTEN BY KWONG, MATTHEW WANG SHUN
Abstract

Varies of studies and algorithms have pushed the boundaries on the efficiency of route planning services, but most traffic network routing algorithms only works on static graph and ignore the possibility of reacting to real time traffic incidents and events. Drivers especially emergency services would benefit heavily if a route planner can provide an actual shortest path based on current traffic conditions.

This project aims at developing a real time traffic management system. Specifically, this project extends A* algorithm by providing a new heuristic function. In order to incorporate the algorithm, a backend web service and a cross-platform mobile application has been developed to complete the system. To demonstrate the idea, a simulator is incorporated into the mobile application to model different scenarios.

A new set of algorithms written in C++ is designed, proposed and implemented for traffic-aware and incident-aware pathfinding. Backend web service implemented using CppCMS [1], which acts as a middleware between the user and the implementation of the algorithms. A cross-platform mobile application developed by React Native [2] which provides routing service and simulation.

This system is able to adjust its strategy accordingly in order to suggest the shortest path for the users. This system can provide most of the fundamental functionalities that common route planners do, while being able to adopt the change of traffic conditions in Hong Kong. This would lower the traveling time while drivers are driving under unusual traffic conditions.
Acknowledgments

We would like to thank our supervisor, Dr. Reynold Cheng, for guiding us throughout in the project.

We would also like to express our gratitude towards the Department of Computer Science, Faculty of Engineering, The University of Hong Kong for giving us the opportunity and technical support for doing this project.
# Table of Contents

Abstract ............................................................................................................................................................... 1
Acknowledgments ............................................................................................................................................... 2
Abbreviations ...................................................................................................................................................... 5
1 Introduction ................................................................................................................................................ 6
   1.1 Background and Motivation ................................................................................................................ 6
   1.2 Objective and Scope ............................................................................................................................ 7
   1.3 Deliverables ......................................................................................................................................... 8
   1.4 Outline of Report ................................................................................................................................. 8
2 Methodology .............................................................................................................................................. 9
   2.1 Technology .......................................................................................................................................... 9
      2.1.1 React Native ..................................................................................................................................... 9
      2.1.2 MobX ........................................................................................................................................... 9
      2.1.3 Mapbox SDK ................................................................................................................................ 9
      2.1.4 Turf .............................................................................................................................................. 9
   2.2 Software Architecture Design ............................................................................................................ 10
      2.2.1 Flux ............................................................................................................................................ 10
      2.2.2 MobX ......................................................................................................................................... 12
      2.2.3 Flux vs. MobX ............................................................................................................................. 13
   2.3 Function Design ................................................................................................................................. 14
      2.3.1 Routing Service .......................................................................................................................... 14
      2.3.2 Reroute Service ......................................................................................................................... 17
      2.3.3 Simulation .................................................................................................................................. 19
3 Results ....................................................................................................................................................... 21
   3.1 Mobile Application ............................................................................................................................. 21
      3.1.1 Home Page .................................................................................................................................... 21
      3.1.2 Location Searching ..................................................................................................................... 24
      3.1.3 Routing ...................................................................................................................................... 25
      3.1.4 Reroute ...................................................................................................................................... 26
      3.1.5 Simulation .................................................................................................................................. 27
   3.2 Incident Simulator .............................................................................................................................. 28
4 Conclusion and Future Work .................................................................................................................... 29
References ......................................................................................................................................................... 30
List of Figures

Figure 1. Flux data flow ........................................................................................................... 10
Figure 2. MobX data flow ...................................................................................................... 12
Figure 3. Routing service system sequence diagram .......................................................... 14
Figure 4. Routing service frontend sequence diagram ......................................................... 15
Figure 5. Reroute service system sequence diagram ............................................................ 18
Figure 6. Simulation system sequence diagram .................................................................... 19
Figure 7. HKSAR boundary (red region) ................................................................................ 20
Figure 8. Home Page ............................................................................................................. 21
Figure 9. Enable location service confirmation dialog ......................................................... 21
Figure 10. Restaurant Marker .............................................................................................. 22
Figure 11. Compass .............................................................................................................. 22
Figure 12. 3D buildings ........................................................................................................ 22
Figure 13. Location search result ......................................................................................... 24
Figure 14. Direction Page ..................................................................................................... 25
Figure 15. Progress bar ....................................................................................................... 25
Figure 16. Suggested path .................................................................................................... 25
Figure 17. Routing result ..................................................................................................... 26
Figure 18. Reroute result ..................................................................................................... 26
Figure 19. Simulation drawer menu ..................................................................................... 27
Figure 20. Types of simulations .......................................................................................... 27
Figure 21. Simulation result ................................................................................................. 27
Figure 22. Incident simulator .............................................................................................. 28
Abbreviations

API  Application programming interface
GPS  Global Positioning System
GUI  Graphical User Interface
IMS  Incident Management System
OSM  OpenStreetMap
UI   User Interface
1 Introduction

1.1 Background and Motivation

Traffic management is one of the most complex and crucial problems in developed cities like Hong Kong. Many online map services featuring route planning service are developed aim to solve this problem. With such a feature, it is now possible to quickly identify the shortest path between the desired starting point and destination. This enables drivers to spend a minimal amount of time and effort in finding the right direction while they concentrate on driving.

One trivial case is to model this problem to a shortest path problem on a graph $G(V, E)$. $G$ represents the road network; $v$ represents the junctions between the road segments, $\forall v \in V$; and $e$, with positive edge weight, represents the road segments, $\forall e \in E$. The road network can be modelled as a graph with positive weights. This problem is popular and well-studied. Throughout the researches, popular algorithms like Contraction Hierarchies [3] have been presented. These algorithms all work similarly. They first pre-process the graph without knowing the origin and destination. Then provide the shortest path with the given origin and destination. These algorithms assume the road network is static, so it only has to be pre-processed once and can be used for answering a large number of O.D. requests on the same road network efficiently.

However, the actual road network is not necessarily static, which there are 2 main scenarios causing the changes.

1. Traffic incidents and events potentially affect the travelling of road segments.

2. A lot of other vehicles are served and routed in the road network, causing substantial traffic in the road network.

Since these algorithms only aware of the pre-processed graph, they are unable to react to these scenarios. Because of that, these algorithms have lost their advantages, as their indexing strategies include initial edge weights (default travelling time) which are changed or lengthened in these case scenarios in their precomputed data and relies on them to speed up the search. That is, expensive re-computation has to be done whenever there is a change in the road network.

Road incidents and large-scale events being the main reasons for severe traffic congestions. This leads to huge waste of valuable time and even economic loss. In fact, citizens in Hong Kong may suffer at least a 200% increase in commuting time in serious traffic jams [4]. Therefore, a travelling route suggestion service that can take incidents into consideration may provide a way to alleviate the problem.

Currently, one of the common workarounds is to use one of the traditional algorithms mentioned and perform re-computation in a regular basis. Although this workaround takes the balance between efficient query and expensive re-computation, it functions by tolerating the use of outdated traffic data. In doing so, suboptimal paths may be returned by the algorithm which increase the actual traveling time. Sufficiently, traveling a longer path and spending slightly more time on it will mostly acceptable in end user’s point of view, but this does not consider cases where the shortest path must be provided to emergency services, where every second means life and death.
On the other hand, amongst all available solutions for this problem, TallyGo [5] (previously known as ClearPath) provides the most similar service as this project is trying to deliver. Users first submit routing requests with origin and destination locations (O.D. requests) to the application server over the Internet, a path with the shortest travelling time will then be returned to the user device and be shown on a digital map. While calculating the path to be returned, TallyGo detects and takes live incidents and road congestions into consideration and suggests alternative paths to the user to ensure reasonable travelling time and prevent worsening the congestion. Despite being a well-established product, there are still shortcomings of TallyGo that prevents it from being directly used in Hong Kong:

1. Large-volume requests are not specifically handled in TallyGo. In such cases, a gigantic amount of O.D. requests with the same origin may be sent out at the same time when many users are trying to leave an event. If all requests are served with the same suggested route, new traffic congestion may be created by the users on same path segments, the efficiency of the application may then be diminished.

2. TallyGo does not support Hong Kong as one of its service regions for routing with incident data because of the lack of accurate incident and traffic data available in Hong Kong.

1.2 Objective and Scope

We hypothesize that there is significant value to get a route planning service to detect and react to real-time traffic incidents autonomously. Due to the wide breadth of the two components and the time constraints, this project focuses on traffic management, more specifically, how to take real-time events into consideration while doing shortest path searches.

Understanding the limitations mentioned above, it also aims to deliver a web-based incident management service (IMS) that provides route suggestion functions focusing on the two main contributing factors of traffic jams – Large-scale events and Road Incidents, and to bridge the gap left by the other services:

1. **Large-scale Events**
   Huge traffic flow and extra congestion may be generated from large-scale events and occasions like the Hong Kong Book Fair and large concerts held by famous singers. Therefore, the service aims to achieve routing which suggests efficient routes that minimize extra congestion created and ensure a reasonable travelling time for the users. This can be achieved by updating the map graph with the extra loads created to the roads by the served O.D. requests added to the edge heuristics.

2. **Road Incidents**
   To avoid duplication of work, while the upstream incident detection mechanism relying on the traffic data provided by Hong Kong taxis will be handled by a peer group under the same supervisor Dr. Reynold Cheng, this project will put focus on using existing Hong Kong map data for the routing of O.D. requests assuming incident reports are available for access. The service tries to suggest a route which has minimal path segments passing through any congested areas affected by road incidents. Here, “road incidents” does not only cover car crashes, large events that generate huge traffic flow to nearby areas are also included.

3. **Routing in Graph with Dynamic Edge Weights**
   This project tries to deliver a set of algorithms that brings sensibility of changes in edge weights in the road network brought by the other requests and the live incidents, and ultimately able to perform routing in a graph with dynamic edge weights.
1.3 Deliverables

There are 3 major deliverables in this project:

1. **Routing algorithm**
   A set of algorithms designed to solve origin to destination routing requests that take real-time events into consideration. Accepts parameters passed from the Backend service and returns the corresponding result back to the Backend in real time.

2. **Backend service**
   A set of functions implemented as web API endpoints to provide routing, rerouting and incident management functionalities to the Frontend service. Acts as the controller to handle request parameters (e.g. origin and destination location in routing) and pass them to corresponding algorithm modules for the results needed to be returned to the Frontend.

3. **Frontend service**
   A mobile application acts as both application prototype and simulator.
   a. Application prototype provides a route planning service. It also includes basic features from ordinary mapping mobile application.
   b. Simulator simulates an environment which involves multiple users in the road network, mimicking the real world. The environment can be used to test and improve the routing algorithm efficiently and quickly.

1.4 Outline of Report

The documentation for this project is divided into three reports. Each of them emphasizes on the methodologies, results and challenges encountered for different aspects, despite all reports share the same background and motivation of the project. Reader are recommended to refer to all three reports to acquire a complete understanding of the project.

This report focuses on the implementation of the frontend service (i.e. the application and the simulator) and proceeds as follows: This report first describes the methodologies including the technologies involved and the workflow of different system functionalities. Then, the final result is discussed, followed by some concluding remarks and possible extensions of this project.
2 Methodology

This project consists of 3 major components: the routing algorithm, the backend service and the frontend service. The subsequent sections talk about the frontend service of this project.

2.1 Technology

There are several key technologies and libraries involved in this mobile application.

2.1.1 React Native

React Native is a cross-platform mobile application framework developed by Facebook, which allows supporting both Android and iOS platforms in a tight schedule. Also, due to its cross-platform nature, only one version of the application has to be maintained, as opposed to multiple versions for multiple OS platforms.

On the other hand, React Native has relatively higher performance than similar frameworks and has gained its popularity rapidly in recent years, which provides strong community support.

2.1.2 MobX

MobX is a state management library for JavaScript, which also supports React Native. The main reason for using a third-party state management library is because the default state management Flux has unsatisfied performance, while MobX provides a great performance improvement (see 2.2.3 Flux vs. MobX).

Comparing to Redux, an implementation Flux pattern which is generally a more popular choice. MobX is more suitable for small to medium size project like this project due to its simplicity and ease of use.

2.1.3 Mapbox SDK

Mapbox is a provider of custom online maps for websites and applications. It provides a development toolkit which allows a customizable map view to be embedded into a mobile application. The decision of choosing Mapbox SDK instead of the more popular Google Maps API is, in fact, a compromise. The terms of service of Google Maps [6] has explicitly forbidden the creation of “a substitute of the Google Maps Core Services”, which includes route planning.

However, despite being less popular, Mapbox SDK is more customizable than Google Maps API in terms of visualization. The only limitation is the scarcity of documentation which causes a few setbacks during the implementation process.

2.1.4 Turf

Turf, or Turf.js is a JavaScript library for spatial analysis. It provides spatial operations and helper functions for creating GeoJSON data object. GeoJSON is a special form of JSON object which is used to represent map objects like paths, polygons, and boundary boxes in Mapbox SDK.
2.2 Software Architecture Design

2.2.1 Flux

React Native inherits the design pattern of React, which make use of Flux, an architectural pattern proposed by Facebook for building client-side application, instead of Model-View-Controller (MVC) [8]. Flux applications have 4 parts, store, dispatcher, view, and action. Controllers do exist in Flux however, it is being considered as controller-views, which their responsibilities are handled by views [7].

In this project, although most of the state management are handled by MobX, there are still scenarios require the use of Flux, for example passing data from parent components to child components and vice versa.

2.2.1.1 Store

Stores manage application state, which can store both domain state and user interface state (e.g. whether user log in successful or not). Unlike model in MVC, store does not provide any data structure to represent certain object, instead it acts as a dictionary which one of its responsibility is to manage model instances. Hence, stores shall not take other stores as dependencies.

One of the key properties of store is the application cannot directly modify the state managed by the store. State changes can only be requested by passing an action to the dispatcher. This ensures the integrity of the state managed by the store and prevent any unwanted state changes.

After receiving actions, store decides on which of them to act. Once the store has made state changes, it emits a change event, notifies the view to update accordingly.

In this project, these properties are maintained and encapsulated by MobX. State changes can be requested through a set of APIs exposed which the syntax is similar to conventional setters.

2.2.1.2 Dispatcher

Dispatcher is a singleton, meaning that there will be one and only one dispatcher instance exists in the entire application. It broadcasts actions/events to all registered stores regardless whether the stores are responsible to handle such events or not. In other words, it is store’s responsibility to decide whether it should react to the received events.
2.2.3 View

View is the user interface component, which is responsible for rendering the user interface and for handling the user interactions. In other words, it handles responsibilities of both MVC’s view and controller. It listens to store changes and re-render accordingly. In order to separate the concern, views can be further divided into presentation and container views.

Presentation views are responsible for rendering the UI and for managing UI logics. They do not connect to dispatcher or stores; Hence they will not react to any state changes. Once user interactions have been performed, presentation views will notify container views for further processing.

Container views are wrappers of presentation views. They are connected to stores and dispatcher and listen for events from stores. Container views dispatch actions in response to user interactions notified by presentation views. On the other hand, once container views have received events emitted by the store, they will get the updated state and pass the new data to presentation views.

2.2.4 Action

Actions are objects that contain information needed to perform certain actions. For example, a LoginAction object may contain username and password for performing account login. Actions are usually immutable; Their properties and values cannot be changed once created. This ensures their integrity throughout the data flow.
2.2.2 MobX

![Figure 2. MobX data flow](source: [9])

Compare to Flux, MobX’s architecture is much simpler, which contains state, action and view.

2.2.2.1 State

State is the data that drives the application, which is the same as the “state” stored in Flux’s stores (see. 2.2.1.1).

In this project, classes name “Store” are created to store the state of the application. They retain most of the properties of Flux’s stores, except the state managed by MobX can be directly modified as opposed to pass actions to the dispatcher.

2.2.2.2 Derivations

Derivations are variables or data derived from the state. They are commonly used in views to provide data for the UI. When the state changes, the corresponding derivations will change accordingly and trigger the views to re-render. Generally, MobX distinguishes two kind of derivations, Computed values and Reactions.

Computed values are derived from the current observable state. For example, if the state stores the catalogue of an online shop, then one possible computed value can be the number of items in the catalogue.

Reactions are side effects that happen automatically if the state changes. Traditionally, reactions are mostly used for re-rendering part of the UI while the state changes. However, since React Native can trigger re-rendering automatically. Reactions are seldomly used in this project.

2.2.2.3 Action

In MobX’s concept, “an action is any piece of code that changes the state” [9] and should not be confused with actions in Flux. As the characteristic of controllers of MVC closely aligns with that of actions, the controllers in this project are named “Actions”.

2.2.3 Flux vs. MobX

For Flux, it provides a better project structure and ensures integrity of state management, because states cannot be modified unwillingly. However, due to its complicated data flow structure, it sacrifices performance. First, passing actions to the dispatcher introduces overhead to the data flow process. Second, Flux’s dispatcher acts as a broadcaster, meaning that it will send the actions to every registered store regardless whether the stores are responsible to handle such actions or not. This creates unnecessary conditional checking and further increases the overhead of handling state changes.

On the other hand, states in MobX can be directly modified, which the integrity of the states are not guaranteed. However, its simplified structure is more understandable and easier to learn and improves the performance of state management.

For large scale projects, maintainability usually outweighs performance. Therefore, Flux, or one of its implementations Redux is preferred. However, the structure of small-scale projects is relatively simple regardless, which they benefit less from the well-defined structure of Flux. In fact, due to considerable among of boilerplates from Flux and Redux, the project structure may be clumsier than not using either of them. Therefore, being a small to medium scale project, this project mainly uses MobX for state management, and uses Flux if necessary, since React Native uses Flux by default, and using Flux for some operations can achieve a better structural design.
2.3 Function Design

There are several main functions in this application. The remaining sections of this part present the general workflow of each function.

2.3.1 Routing Service

Routing service is the most common function users will be using. This single endpoint first finds the shortest path, then it updates the graph according to the path found. As shown in Figure 3, the normal workflow (a.k.a. main success scenario) is as follow:

1. User inputs an origin and a destination
2. Frontend service sends a HTTP request to the backend service with the input origin and destination
3. Backend service find the closest nodes which are no further than 100m from the input origin and destination respectively. Denote the 2 nodes as $O'$ and $D'$.
4. Backend service makes a function call to the implementation of the algorithm with $O'$ and $D'$ as arguments
5. The implementation of the algorithm returns a path
6. The implementation of the algorithm goes through update stage
7. The implementation of the algorithm returns the path to the backend service
8. Backend service returns the path to the frontend service
9. Frontend service stores the path for the reroute service (see 2.3.2 Reroute Service)
10. Frontend service presents the path to the user
Given the input origin $O$ and destination $D$, if routing service is unable to find a node in the graph which is within 100m from $O$ and $D$ respectively, it will consider $D$ is inaccessible from $O$. In reality, this means either $O$ or $D$ is in a remote area which is inaccessible by vehicles.

The following section presents a more detail workflow of routing service in frontend.

![Sequence Diagram](image)

*Figure 4. Routing service frontend sequence diagram*

As shown in Figure 4, the normal workflow is as follow:

1. User presses the direction button in the application
2. The application presents the direction page
3. User enters a keyword to search for a location
4. The application makes a HTTP request to Mapbox Geocoding service with the input keyword
5. Mapbox Geocoding service returns a list of locations matching the keyword
6. User picks one of the locations
7. The application adds the selected location (a stop, waypoint) to a list

*System repeats step 3 - 7 until there are 2 stops in the list (i.e. 1 origin and 1 destination)*
8. The application sends a HTTP request to the backend service with the input list of stops
9. Backend service returns a path (in the form of a list of edges)
10. The application converts the path into a list of vertices
11. The application makes a HTTP request to Mapbox Map Matching Service with the list of vertices
12. Mapbox Geocoding service returns a new list of vertices
13. The application calls lineString helper function from Turf to convert the list of vertices to polyline string
14. The application draws the polyline on the map view and presents it to the user

Notes

Step 3 - 7: The design of the frontend service in fact supports more than 2 stops (i.e. multiple waypoints between the origin and destination). However, due to the time constraints, this feature is eventually not implemented.

Step 4, 5: Mapbox Geocoding service provides a forward geocoding service. It takes a string as keyword and a number of searching criteria and return a list of locations matching the keyword and the criteria.

Step 9: The returned path is in the form of a list of edges instead of a list of vertices is due to technical considerations of reroute service

Step 11, 12: Mapbox Map Matching service snaps fuzzy, inaccurate GPS coordinates to the OpenStreetMap road and path network. This produces cleaner paths that can be displayed on a map.

Step 13: In order to draw a path on the map view of Mapbox SDK, the path must be represented as a polyline string.
2.3.2 Reroute Service

Reroute Service is automatically invoked for every $N$ minute, where $N$ is a constant. The main purpose of this endpoint is to perform new O.D. requests for the user with the current location as the new origins.

This brings the benefit of rerouting user away from incidents that happen after the initial route is returned. In the initial settings, the system has no knowledge of returned paths and therefore can only prevent routing users into incidents when the incident appears before the routing request. This may not be ideal in a real-life use case when an incident occurs in the returned path after the path is returned, or that when many other vehicles suddenly arrive at a road segment earlier than this user. The user will not be notified and will ultimately experience a longer total travelling time in the traffic jam. The reroute functionality is therefore created for user clients to request for routing from the current position regularly and get an updated route calculated with the latest traffic information like the traffic density of the roads and the presence of incidents. Note that the newly calculated path will only be used and returned to the user when it costs less time than the remaining travelling time of the currently used path for maintaining the user experience.

In real-life, vehicle locations can deviate from what estimated in the initial route due to factors like road conditions and realized travelling speed. The routing algorithm depending on the Current Density cache may therefore be misled and returns non-optimal paths. The reroute function can therefore also act as a calibration of the density information stored in the system by resetting and updating the density impact brought by each vehicle with their latest position and the newly predicted locations. The Current Density cache can therefore reflect the realized traffic better and help the routing algorithm to return a path with better travelling time.

Currently, the time interval is set to 5 seconds for testing purposes, while it is believed that setting the time interval to a few minutes will be more suitable for department server deployment. When considering the length of the time interval, there are several factors to consider:

1. **User experience**
   The more frequent Reroute service is invoked; the more accurate the returned path is. Users will be able to follow the right path much easier.

2. **Workload of the server**
   The more frequent Reroute service is invoked; the busier the server is. A busy server may have slow response or even causing the HTTP connection to timeout.

3. **Mobile data usage**
   The more frequent Reroute service is invoked; the more mobile data is used. Bandwidth-intensive applications are always less desirable, since users may have to pay more for their cell phone bill, giving a poor impression tot the users. Even though this application requires internet connection, it should avoid draining user’s mobile data.

Therefore, in order to balance the these factors, it is recommended to set the time interval to 1 – 5 minutes.
As shown in Figure 5, the normal workflow is as follows:

1. Frontend service gets user’s current GPS location
2. Frontend service retrieves the stored path (see 2.3.1 Routing Service)
3. Frontend service sends a HTTP request to the backend service with the GPS location as the origin, the destination previously input in Routing service and the retrieved path (the “old path”)
4. Backend service makes a function call to the implementation of the algorithm with the input origin and destination as arguments
5. The implementation of the algorithm returns a new path
6. Backend service compares the estimated traveling time of the old path and that of the new path
7. Return the old path to frontend service if the estimated traveling time of the old path is no longer than that of the new path for N minutes
8. Return the new path to frontend service if otherwise
9. Frontend service presents the path to the user
10. The implementation of the algorithm goes through update stage

This service is transparent to users. Frontend will automatically trigger this service for every N minute, until users have reached the destination or have closed the application.
2.3.3 Simulation

In addition to the end user functions, this application also serves as a graphical simulator. Although the various components of the system can be implemented as a working application without the simulator, in order to have a robust and structured system, having the ability to test various scenarios and display the result graphically important.

![Sequence Diagram]

As shown in Figure 6, the workflow of the simulation is as follow:

1. User chooses the type of simulation to conduct
2. The application generates a list of origins and destinations based on the selected type
3. For each pair of origin and destination
   a. Treat it as ordinary O.D. request and run Routing Service

There are 5 types of simulation, which each of them indicates how the origins and destinations are generated.

The first 4 types of simulations are **pre-set simulations**, which the coordinates are selected from a fixed set of 100 coordinates.

1. Single Origin to Single Destination Simulation
2. Single Origin to Multiple Destinations Simulation
3. Multiple Origins to Single Destination Simulation
4. Multiple Origins to Multiple Destinations Simulation

These simulations are intended to provide a better visualization of the result. For example, type 1 simulation can demonstrate how the algorithm react to the degree of congestion in the traffic. Instead of directing all requests to the same path, the algorithm tends to redirect latter requests to new path(s) because the original path is congested.
These 4 simulations are enough covered all possible cases. However, it is important to make sure the algorithm not only works on the coordinates pre-sets, but also arbitrary locations. Therefore, an extra type of simulation is included.

The last type of simulation is *random simulation*, the coordinates are generated by pseudo-random number generator bounded within the HKSAR map boundary (see Figure 7). The reason for using a bound polygon instead of a simple bound box is because the surface area of water is significant when comparing to that of land. Using a rectangle boundary will generate a significant number of coordinates pointing towards the water, causing too many invalid cases being generated.

On the other hand, although the current boundary is imperfect, which some of the water regions like lakes are not excluded and some land regions like Chek Lap Kok are not included, the number of successful cases is enough, and the boundary serves its purpose for this project. While the failure cases (i.e. cases where there is no possible path traveling from the given origin to the destination) can be served as part of the simulation. In that case, there is no need to generate failure cases for testing intentionally.
3 Results

3.1 Mobile Application

The following section illustrates the main functionalities provided by the application and step by step instructions are introduced.

3.1.1 Home Page

Being the first scene displayed when executing the application, the layout of this scene is important as this is the first impression users made upon using this application. It resembles common design of similar applications. This is done on purpose such that users do not have to re-adopt the new UI. This page contains a Map View implemented using Mapbox SDK, a search bar at the top of the screen, and two buttons, the Current Location Button and the Direction Button, at the bottom right corner of the screen.
3.1.1.1 Map View

By default, Hong Kong is shown at the centre of the map. This is because this project, especially the algorithm, is focusing on the situations and scenarios appeared in Hong Kong. Besides presenting a map to the users, the Map View also provides other features.

**Marker**

Markers are added to indicate special locations like schools, hospitals, and restaurants. As the result, users will be able to find these locations easier. Also, when GPS localization service on user’s device is enabled, an indicator will be displayed on the map. This indicator is directional aware, this means not only this indicator is able to indicate the current location of the user, but also indicating what direction are the users facing.

**Compass**

By default, the map is displayed such that North is on the top of the screen. When the map is rotated, a compass will be displayed on the upper right corner of the screen. This compass only indicates the directions corresponding to the orientation of the map, and do not functions as an actual compass.

When users press the compass, the rotation angle of the map will be reset such that North is on the top of the screen.

**3D Buildings**

3D Buildings are added for the easiness of identifying locations instead of merely visual effects. In addition to take reference from street names and house numbers, users can also take reference from the shape of the buildings. This is especially useful in crowded areas like Central and Mong Kok. In these areas, the accuracies of the GPS localisation service are usually undesirable due to the poor signal reception, while the street names and house numbers are hard to observe. In such case, it is easier to recognise the locations based on their general shapes.

However, due to the limited support of Mopbox SDK, only 3D buildings are supported, other structures like overpasses are still rendered in 2D.
3.1.1.2 Current Location Button

The functionalities of the Current Location Button (the white button in Figure 8) can be divided into 2 cases.

**Case 1: GPS localization service is disabled**

When pressing the button, the application will request for turning on the GPS localization service by displaying a confirmation dialog (see Figure 9). Users can choose to accept or reject the request.

Enabling GPS localization service improves user experience while using this application. First, an indicator will be placed on the map to indicate the current location and the direction of the user. This can help the users to quickly identify their current location, especially when users are doing routing request and looking for direction. Second, the location searching service provided by this application is in favour to the locations that are closer to user’s current location. In other words, locations that are closer to users tend to appear at the top of the search result.

**Case 2: GPS localization service is enabled**

When pressing the button, the application will rearrange the map such that user’s current location will appear at the centre of the map. This is a common feature provided by many similar applications as an improvement of quality of life. As the result, users can quickly identify their current location without performing too many pan gestures.

3.1.1.3 Direction Button

The Direction Button (the light blue button in Figure 8) is solely for navigation purpose. It navigates user to the Direction Page, where users can perform O.D. request using the provided GUI.
3.1.2 Location Searching

Similar to most of the map services, this application features a location searching function. The searching function is supported by Mapbox SDK forward geocoding service and some of the data like the images and the short descriptions appeared in the search result are provided by Wikidata.

In the Home Page, a search bar is displayed at the top of the screen. By entering keyword(s) in the search bar, the application will present a list of locations matches the entered keyword.

For the visualization of the search result, instead of conventional list of texts design, a list of cards (ViewPager cards) featuring a photo and a brief description (if available) for each location is displayed. This is because it is not uncommon to see multiple locations share the same name. For example, apart from the well-known Queen’s Road in Hong Kong, there is another Queen’s Road located in India as well as a “Queens Road” in London. With the addition of the photos and descriptions, it is easier for the users to distinguish the locations.

Moreover, the search result is in favour to the locations that are closer to user’s current location. That means if a user in Hong Kong is searching for “Queen’s road”, “Queen’s Road, Hong Kong” would be prioritized before “Queen’s Road, India”. This further improve the user experience of location searching.

Figure 13. Location search result
3.1.3 Routing

Users may look for direction by performing an O.D. request in the Direction Page of this application. The Direction Page contains two main components, the Stops List, and the Direction Footer.

Stops List, or Waypoints List, presents the selected origin and destination of current O.D. request (see Figure 14 yellow region). The UI design reserves rooms for extension where additional waypoints can be added to current O.D. request (i.e. Travelling Salesman Problem). Since this application is expected to be mainly used in smart phones which have small screens. When multiple stops and destinations are selected, the list will occupy significant amount of space, blocking the map. Therefore, this list is displayed in a separated page instead of at the top of the screen together with the map. This extension however is eventually not implemented in this project due to the time constraint. The UI currently prevents users from selecting more than 2 waypoints, and the backend service only accepts exact 2 waypoints (i.e. one origin and one destination).

By pressing one of the waypoint placeholders on the list, user will be navigated to the Location Search Page, where users can select the desired origin and destination. Then, the process is the same as performing location searching from the Home Page (see 3.1.2 Location Searching). After entering the keywords in the search bar, the application will present a list of matching locations and users may select the desire location from the search result. If users wish to modify their choice of origin or destination, they may press the corresponding waypoint again to modify it.

After selecting both origin and destination, the application will send an HTTP request to the backend service. A progress bar will be displayed to indicates the progress of handling the O.D. request. This progress bar is in fact a “fake” progress bar, which does not animate according to the actual progress. This is a common UI implementation workaround for processes that are hard or cannot obtain the progress of it. The main purpose of it is to let the users know the application is still running instead of crashed. After the backend service has finished handling the request, it will either return a suggested path or an empty path to indicate the O.D. request is invalid. Finally, the path (if any) reported by the backend service will be displayed on the map.

If users wish to make a new O.D. request or review their waypoints selection, they can toggle the Stops List and the map by pressing the buttons on the left and right side of the Direction Footer (see Figure 14 green region) respectively. Also, the Direction Footer is originally intended to display the next major waypoint users are heading to along the path. Yet, due to the time constraint, this feature is eventually not implemented, and a placeholder text is used instead.
3.1.4 Reroute

After an O.D. request has been made, the application will send Reroute requests on a regular basis automatically. If a new path is returned by the Reroute Service, the original path will be removed, and the new path will be added to the map. For example, in Figure 17, an O.D. request has been made from Sha Tin to Central. While Figure 18 shows a new path returned from reroute service where the user is currently in Choi Hung.
3.1.5 Simulation

In the Home Page, a drawer menu button can be found on the left side of the search bar (see Figure 8). When the button is pressed, a drawer menu will be presented. This drawer menu, when taking reference from similar applications, is intended to include advanced features or settings of the application. For this project, only the simulation feature has been included.

When users press the Picker (see Figure 19 red region), a dialog will be prompted with a list of simulations available for user to select.

For the pre-set simulations (i.e. the first four list items in Figure 20), once the users have selected the desired type of simulation, users may press the “simulate” button to run it. Once the simulation has completed, the result will be presented on the map.

If random simulation is selected, a slider will be presented to the users where users may select the number of O.D. requests generated in current simulation. Currently, the maximum number of requests generated is 100. This limit is introduced to prevent overloading the department server.
3.2 Incident Simulator

Incident simulator is a simple tool developed using JavaFX, the successor of Java Swing. It simulates the appearance and resolution of traffic incidents and contains 2 services to handle each of the task respectively. They send incident appearance reports and incident resolution reports to the backend service for every random amount of time to simulate the detection and clearance of traffic incidents. Currently, the interval between sending each incident report has been set to randomly generate between 0 – 3 seconds. This interval has been greatly shortened in order to trigger incidents handling frequently in backend service for testing purposes and is obviously unrealistic when comparing to actual traffic scenarios.

On the other hand, a simple GUI has been developed which contains two buttons. By clicking the buttons, users can trigger incident appearance and resolution respectively without waiting for the time to pass.

![Incident Simulator](image)

*Figure 22. Incident simulator*
4 Conclusion and Future Work

We have justified why real time traffic management is important and how taking traffic incidents and conditions into consideration while performing O.D. request can be useful to avoid traffic congestions. As discussed, most of the algorithms from previous studies pre-process the graph based on initial edge weights in order to improve query performance, and in doing so, restrict their capability in reacting to real time traffic events due to expensive re-computation. Even though, there is a workaround which may return a generally acceptable suboptimal path, an optimal path needs to be returned during emergencies, namely for emergency services.

Motivated with this intention, this report firstly outlines the advantages of building a cross-platform mobile application for this project, and how that was facilitated by React Native and varies technologies. Next, it introduces Flux, and MobX, and how it was leveraged as the structure of the components. Next, it presents the workflow of varies functionalities provided by this project. Finally, the result and the GUI of the application are introduced.

Although the initial aim of the project is to implement a cross-platform mobile route planning application, it is not achieved completely. Minimal testing and fine tuning have been done on iOS devices even though React Native is technically a cross-platform mobile application framework. Nevertheless, much effort was put into closely incorporating the less popular, lack of documentation Mapbox SDK in to the application, and structuring the project as discussed in this report. Thus, given the essential functionalities and carefully engineered project structure have been established, it is believed that extensions can be easily done on this project.
References


