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

Author: GAN CHEN (UID: 3035092662)
Supervisor: Dr. Reynold C.K. Cheng
Date: Apr 16, 2017
Abstract

As reported in the latest plans of the Hong Kong government, there is a pressing need for Hong Kong to be an intelligent city. Our group is going to take on this task and make the whole Hong Kong city intelligent. Therefore, the main purpose of this project is to automatically detect any traffic incidents from online sources that happen on the roads for the whole Hong Kong city, and then alert drivers and provide alternative routing solutions to them. To achieve this, we are going to have an android mobile application to manage incidents for the whole Hong Kong city. There are mainly three functionalities in our android mobile application: incident report, incident information and navigation to avoid incidents.

Regarding to incident report functionality, users of our application is able to report any incidents that they have encountered. And then the socket will receive those information and deliver them to all other users. Regarding to incident information functionality, currently the incident information except for the user reported ones is crawled from Hong Kong Transportation Department. Because there crawled files are plain text files therefore we need to extract location data from them in order to show them as real-time incidents on the map. Regarding to navigation to avoid incident functionality, we are going to develop our own navigation algorithm that can avoid incident spots by taking advantage of OpenStreetMap road network data of Hong Kong. The data from OpenStreetMap is trustworthy because some people have used it to make cool applications, and one of them has got 0.1 billion downloads in android play store.
Acknowledgement

Special thanks to my supervisor Dr. Reynold C.K. Cheng, who gave me valuable advice throughout the project. And also my mentor Sun HaiQi, who gave me kind support throughout the project. And also my group mate Alice, although she only worked with me in the first semester, we benefited a lot from helping each other.
# Table of contents

1. Introduction  
   1.1 Background  
   1.2 Problem Statement  
   1.3 Objective  
   1.4 Scope  
   1.5 Workload Distribution  
   1.6 Outline  

2. Methodology  
   2.1 User base  
   2.2 Application functionalities  
   2.2.1 Incident report functionality  
   2.2.2 Incident information functionality  
   2.2.2.1 Location data extraction  
   2.2.2.2 Observation from crawled files  
   2.2.2.3 Location keyword matches  
   2.2.4 Incident information with android  
   2.2.4.1 Incident on the map  
   2.2.4.2 Incident list  
   2.2.3 Navigation functionality  
   2.2.3.1 General navigation  
   2.2.3.2 Incident avoidance algorithm  
   2.2.3.2.1 Road network data  
   2.2.3.2.2 A* algorithm  
   2.2.3.2.3 Different locations of incidents  
   2.2.3.2.3.1 Incidents on vertices  
   2.2.3.2.3.2 Incidents on edges that cause congestion  
   2.2.3.2.3.3 No-left-turn, no-right-turn and oneway road  
   2.2.4 Socket server  

---

1. Introduction  
   1.1 Background  
   1.2 Problem Statement  
   1.3 Objective  
   1.4 Scope  
   1.5 Workload Distribution  
   1.6 Outline  

2. Methodology  
   2.1 User base  
   2.2 Application functionalities  
   2.2.1 Incident report functionality  
   2.2.2 Incident information functionality  
   2.2.2.1 Location data extraction  
   2.2.2.2 Observation from crawled files  
   2.2.2.3 Location keyword matches  
   2.2.4 Incident information with android  
   2.2.4.1 Incident on the map  
   2.2.4.2 Incident list  
   2.2.3 Navigation functionality  
   2.2.3.1 General navigation  
   2.2.3.2 Incident avoidance algorithm  
   2.2.3.2.1 Road network data  
   2.2.3.2.2 A* algorithm  
   2.2.3.2.3 Different locations of incidents  
   2.2.3.2.3.1 Incidents on vertices  
   2.2.3.2.3.2 Incidents on edges that cause congestion  
   2.2.3.2.3.3 No-left-turn, no-right-turn and oneway road  
   2.2.4 Socket server
1. Introduction

1.1 Background

It can be shown from the figure(Figure 1) below that the number of traffic accidents in Hong Kong does not reduce significantly from 2005 to 2015. Instead, it remains around 15000 per year and around 2000 of them are serious or fatal. Therefore it is necessary for us to figure out some ways to improve this situation.

As this is an old-fashioned problem for cities, therefore there are already some automatic incident detection technologies existing to help reduce the number of the traffic incidents. However, most of them exploit data sent from stationary sensors and cameras installed on the roads, which incurs expensive installation and maintenance fees of such sensors. To avoid expensive installation and maintenance fees of sensors, we would like to develop an android mobile application to explore and extract incident information from some providers, such as Hong Kong Transportation Department website. And then provide navigation that can help users to avoid incidents based on those incident information by implementing an adapted A* algorithm.
1.2 Problem Statement
From the background, we can see that Hong Kong has an urgent need to be an intelligent city since the number of traffic accidents per year is relatively large and it has not been improved effectively in the past ten years. Moreover, the existing incident detection technologies have expensive installation and maintenance fees. Therefore, our team is going to take on this task by developing an android mobile application to provide navigation that can avoid incidents.

1.3 Objective
Since there is compelling amount of traffic incidents in Hong Kong per year, our group is going to make the whole Hong Kong city more intelligent by properly managing incidents in the whole Hong Kong city. Therefore, the main objective of this project is to automatically detect from online sources any traffic incidents that happen on the roads for the whole Hong Kong city, and then alert drivers and provide alternative routing solutions to them.

1.4 Scope
This project will develop an intelligent system for the whole Hong Kong city to reduce the number of traffic accidents. Although our system aims to detect all kinds of incidents that will cause accidents on the roads on the long run, for the current stage we will first focus on traffic accidents that happens on the roads. It will have a front-end android mobile application to collect geographic information from our users(especially passengers), and a back-end server to analyze and evaluate those collected geographic information in order to detect traffic accidents. The core work of our system is based on incident information collected by some means. As the originally crawled information is mostly pure words, we need to further process those crawled files and extract useful geographical information from those files.
1.5 Workload distribution

<table>
<thead>
<tr>
<th>Mobile UI (Android)</th>
<th>Alice (CS second major student)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident report (Android)</td>
<td></td>
</tr>
<tr>
<td>Incident Information (Android)</td>
<td>Me (CS final year undergraduate student)</td>
</tr>
<tr>
<td>Navigation (Android)</td>
<td></td>
</tr>
<tr>
<td>Socket Server</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. Table of workload distribution

This table shows clearly the workload distribution of this incident management project. The mobile user interface was designed by my partner Alice, who is a computer science second major student and was working with me during in first semester. The incident report functionality of the android mobile application was also finished by her. The remaining work, including the incident information and navigation functionalities of android were done by me, so was socket server design and implementation.

1.6 Outline

The above is the basic background and scope of our incident management project. In the following, we will describe the methodologies of the project. The methodologies of the project will be discussed in terms of the three main functionalities of the android mobile application: incident information, incident report and navigation that can avoid incidents.

2. Methodology

Since our ultimate goal is to properly manage the traffic incidents in the whole Hong Kong city, our focused population will be Hong Kong drivers. Generally speaking, we are going to write an android mobile application that provides three main functionalities: incident report, incident information and navigation to help avoid incidents.
2.1 User base

Our team is going to write an android application to manage traffic incidents for drivers, therefore it is important to figure out the amount of potential users in the market. After some considerations, we choose traffic passengers in Hong Kong as our target users. When a passenger is taking a means of transportation, he or she can play the role as a driver to provide valuable data for traffic incident detection. When a passenger is on the roads, he or she can play the role as pedestrians to report traffic incidents.

After some research, we found some papers related to the amount of passengers daily in Hong Kong and summarize it into a table (see Figure 3 below). From this figure, we can see that there are many different companies providing bus service in Hong Kong. And their services are distributed around different districts of Hong Kong. From the last row of this table, we can see that the total number of passengers taking buses daily is 4.52 million. It is a pressing number, since the population of Hong Kong is just around 7 million. From all these data, we may conclude that we have a large user base since there are such a large amount of passengers daily and they are distributed around different districts of Hong Kong.

<table>
<thead>
<tr>
<th>Company in service</th>
<th>Kowloon (no. of bus routes)</th>
<th>New Territories (no. of bus routes)</th>
<th>Hong Kong Island (no. of bus routes)</th>
<th>Cross-habor (no. of bus routes)</th>
<th>Size of fleet</th>
<th>Total passengers daily</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kowloon Motor Bus Company Limited (KMB)</td>
<td>316</td>
<td>NA</td>
<td>61</td>
<td>3686</td>
<td></td>
<td>2.66 million</td>
</tr>
<tr>
<td>The New World First Bus Services Limited</td>
<td>8</td>
<td>45</td>
<td>34</td>
<td>706</td>
<td></td>
<td>44800</td>
</tr>
<tr>
<td>Citybus Limited</td>
<td>21</td>
<td></td>
<td></td>
<td>181</td>
<td></td>
<td>77000</td>
</tr>
<tr>
<td>Long Win Bus Company Limited</td>
<td>NA</td>
<td>23</td>
<td>NA</td>
<td>189</td>
<td></td>
<td>98000</td>
</tr>
<tr>
<td>The New Lantao Bus Company Limited operates</td>
<td></td>
<td></td>
<td></td>
<td>124</td>
<td></td>
<td>69000</td>
</tr>
<tr>
<td>Public Light Buses (green and red buses)</td>
<td>NA</td>
<td></td>
<td></td>
<td>4350</td>
<td></td>
<td>1.86 million</td>
</tr>
<tr>
<td>In total</td>
<td></td>
<td></td>
<td></td>
<td>9236</td>
<td></td>
<td>4.52 million</td>
</tr>
</tbody>
</table>

Figure 3. Different companies that provide bus services
2.2 Application functionalities

For our android mobile application to be of real use, our android mobile application should be able to attract enough users. According to the data we have got from the website ‘appbrain’, there are mainly two types of android mobile application that are popular in Hong Kong: navigation-based and information-based application.

As shown in the table (see Figure 4 below):

Regarding to navigation-based application: HKBus+ is a mobile application that provides all kinds of bus information to users and at the same time also provides navigation support based on those bus information. It got over 500 thousand in Hong Kong. HK-mini bus one only focuses the information and navigation on mini-buses in Hong Kong, but it also got over 500 thousands. Regarding to information-based application: GovHK Notifications is an application that just delivers government information users. But it got over 100 thousand downloads. HK public holidays is an even simpler application compared to the previous one. Besides functioning like a calendar, it also highlights the public holidays of Hong Kong. Such a simple application got over 500 thousand downloads.

<table>
<thead>
<tr>
<th>Navigation-based</th>
<th>Information-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKBus+ (over 500k downloads)</td>
<td>GovHK Notifications (over 100k downloads)</td>
</tr>
<tr>
<td>HK-mini bus (over 500k downloads)</td>
<td>HK public holidays (over 500k downloads)</td>
</tr>
</tbody>
</table>

Figure 4. This table shows the popularity of navigation-based and information-based applications by showing the number of downloads they have got at the present.

Based on this fact, it is reasonable for us to provide navigation and information functionalities to attract users. Apart from that, we will provide incident report functionalities to support our incident management purpose.
2.2.1 Incident report functionality

Our application allows the users to report emergency incidents on the roads. In this way, drivers will feel that they are involved in our detection process. In this android mobile application, we provide a convenient user interface for users to report incidents for us. In this paper, the convenient user interface is shown as an example.

As shown in the picture (see Figure 7 below):

The location of the reporting spot will be automatically detected by android location service (either GPS, Wifi or cell tower ID) and shown under the ‘Location’ caption. Some categorical tags are also provided for users to choose from different types of incidents in order to save time if they do not have enough time to type in the details. We also provide input text field for users to type in something specific if they like. Finally, there is a submit button for users to submit the report once all information is typed in.

Figure 5. Incident report user interface
2.2.2 Incident information functionality

Our application users are able to receive most updated information of emergency incidents on the roads. This can be the best reward for our application users. Every time our server detects an emergency incident from Hong Kong Transportation Department website or receives an emergency report from users, all drivers using our application will be informed instantly of these emergency incidents. Emergency incidents are sent to users in two manners. The first one is by listing all historical emergency incidents within our application appropriately. The second one is showing incident information as violet icons on the google map.

2.2.2.1 Location data extraction

Before we talk about how the incident information is shown in the android mobile phone, we need to first understand how the incidents are extracted from the crawled files. Currently the incident information for this project is crawled from Hong Kong Transportation Department website, the crawled file contents are plain text. That is, we can not directly tell the geographic information related to a specific incident information from crawled files. As shown is the picture below, this incident information is related to ‘Wan Chai’. But we cannot know this from plain text files directly, therefore we need to locate keywords indicating the spanning area of a specific incident in the plain text file.

Figure 6. A sample crawled incident information
2.2.2.2 Observation from crawled files

From my observations, the location keywords often appear in the title of an incident information file, therefore it is reasonable to put more notices on evaluating the title. Moreover, the location name appears in those files is in district level. That is, it is either Kowloon, Central, Wai Chai or some other district names of Hong Kong. The last observation is that the location keyword related to a specific crawled file from Hong Kong Transportation Department generally appears more regularly than other possible locations in the same file.

2.2.2.3 Location keyword matches

Given those observations, we plan to match all possible location names with certain substrings in one crawled file. We first save all possible location names in a text file, as shown in the figure below. In order for fast filtering during the runtime, all location names are further classified into three parts, with each belonging to one of the Hong Kong’s three main districts: Hong Kong Island, Kowloon and New Territories. From my observation of these location names, the maximum length of the location name is 5. Therefore, we only need to generate all substrings of up to length 5 from each crawled file and then compare each generated substring with names in the location file to find any matches.

![Figure 7. List of location names](image)

8
To decide on the final related location keyword for a specific, the frequency of the matches for each location name will be recorded and the location name with the largest count will be the final result. If there is a tie among match count, we will consider the specific location information as spanning through all those locations with the same highest match count. Given those matched location names, we will then pass each of them to Google API in this format: ‘[location name],Hong Kong’ for geographic coordinate and then show corresponding incident information on the google map.

2.2.2.4 Incident information with android

With geographic coordinates of all those incident information ready in the database, we will then be able to deliver them to our users in these two forms: incidents on the map and incident list.

2.2.2.4.1 Incident on the map

This method is used to show real-time incident information with geographic coordinates on the google map.

Figure 8. Real-time incident information user interface
As shown in the above figure, all those incident spots are shown in the first picture as violet icons. The title of the corresponding traffic incident can be shown after users click on a specific violet icon. To view the whole content of the corresponding incident information, the users just need to further click on the title. With this design, real-time traffic incidents are delivered to users instantly and users will be aware of this once they open this android application since the google map interface is the first screen that the users will see when they open our android application.

2.2.2.4.2 Incident list

Real-time incident is useful to users with no doubt. However, sometimes people may also concern about past incident information. To fulfill the needs of different user groups, we also provide a function which is called incident list to present past incident information to interested users. As shown in the figure below, district filters (Hong Kong Island, New Territories and Kowloon) are shown under the tag ‘LIST’ for users to choose from different districts. The incident information includes title, time of happening and location of happening of the incident. On the right hand side of the incident information, there is a clickable image to open a new page showing the corresponding location of incident.
2.2.3 Navigation functionality

For our android application to be popular, our application will provide navigation functionality that is proved to be prevalent in Hong Kong as this article has shown before. The navigation functionality is specific to buses. That is, instead of considering all kinds of means of transportation, it only focuses on buses. Because the information of buses is easy to collect and it is also convenient for us to make routing plan based on buses since the traveling route of each bus is fixed. Moreover, the navigation functionality is generated from the road network in our own database due to the purpose of avoiding routing through incident spots. The road network data is already collected from some supported open source community.

2.2.3.1 General navigation

For the usage of navigation functionality, as shown in the picture (see Figure 5 below): Google map is used as the interface for user, the current location is detected by android GPS service and shown on the map when the map is first open as shown in the first picture. We also provide autocompleted location name search function powered by Google API to users as shown in the second picture. Based on the current location and the user destination, our backend server is going to generate the best possible routes that can avoid incidents for users. Once the users select a route generated by our server, the android mobile application will show the route on google map by invoking corresponding API as shown in the third picture.
2.2.3.2 Incident avoidance algorithm

As an incident management application, our main purpose is to provide navigation that can avoid incidents for users based on the incident information crawled from some trustworthy sources like Hong Kong Transportation Department website and the incident information provided by our users through the incident report functionality.

2.2.3.2.1 Road network data

Unfortunately, the Google navigation API does not support navigation that can avoid specific points. Therefore we need to configure our own road network database of Hong Kong and generate navigation plan to avoid incidents by implementing an appropriate incident avoidance algorithm. After some online searches, OpenStreetMap is our decided sources for road network data of Hong Kong. Because some people has used its data to build some offline navigation applications and one of them even has 0.1 billion downloads worldwide, so we think it is trustworthy enough. And OpenStreetMap is also widely used for academic purpose in the past years.
Moreover because the data we got from OpenStreetMap is in xml file format containing raw data of the Hong Kong’s road network, we need to transform it into the format that is easy to be loaded into memory by our program. After some implementations, we have transformed OpenStreetMap raw data into two .txt files of vertices.txt and edges.txt.

Here is the format for vertices.txt file:
[node_id] [latitude] [longitude]

Here is the format for edges.txt file:
[way_id] [a series of node ids(neighboring nodes are adjacent on graph)]

2.2.3.2 A* algorithm

With road network data of Hong Kong ready in the database, we need to decide on our base algorithm for navigation. Dijkstra’s algorithm and A* algorithm are both potential candidates. After some experiments, we found A* algorithm has better performance than Dijkstra’s algorithm in terms of both number of nodes visited and time spent on finding the shortest path.

Having decided to apply A* algorithm in our system, we need to understand how this algorithm works step by step:

Step 1: Initialize shortest path distance(SPD) from source node to any other nodes to be infinity.

Step 2: Initialize shortest path from source node to any other nodes to be null.

Step 3: Initialize all nodes as unvisited.

Step 4: Initialize a priority queue, which sorts the nodes in ascending order of the sum of the SPD from source node to the current node and the euclidean distance from the current node to the destination node.

Step 5: Set the SPD from source node to source node to be 0.

Step 6: Add source node to the priority queue.

Step 7: Pop a node from the priority queue.

Step 8: Mark the popped node as visited.
Step 9: If this node is the destination node, terminate the algorithm.
Step 10: Update the SPD of each neighbors of the popped node, if the neighbor is not visited and the current SPD of it is greater than the sum of the SPD of the current node and the edge distance from the current node to this neighbor.
Step 11: If the priority queue is empty, terminate the algorithm. Otherwise, go to step 7.

2.2.3.2.3 Different locations of incidents
Incidents may lie on vertices or edges of the graph. When they lie on edges, we may have more situations to consider. Such as no left turn, no right turn, one-way road and even congestion problems. For each of these cases, we need to consider them separately.

2.2.3.2.3.1 Incidents on vertices
When incidents lie on vertices, we need to mark all those incident spots as visited and all other nodes as unvisited in step 3 of the A* algorithm. Such kind of incidents will introduce the most performance penalties on finding shortest path in terms of number of nodes visited, shortest path distance and time spent on find the shortest path. Some experimental results will be shown in ‘Experiment’ section to clarify this issue.

2.2.3.2.3.2 Incidents on edges that cause congestion
When the incidents lie on edges and cause congestion problem. That is, the whole edge is not available from either side of the edge. To avoid such edges on navigation, we need to remove them before we execute our A* algorithm to find the shortest path for given source and destination nodes. Such kind of incidents also introduce performance penalties on number of nodes visited, shortest path distance and the time spent on find the shortest path. Some experimental results will be shown in ‘Experiment’s section to illustrate this issue.
2.2.3.2.3.3 No-left-turn, no-right-turn and oneway road

When the incidents lie on edges, they may not always cause the whole edges to be not available. Instead, they may introduce some other problems like no-left-turn, no-right-turn and oneway road problem. To solve this problem, we are going to introduce a new algorithm that is presented by a published paper: A route navigation system with a new revised shortest path routing algorithm and its performance evaluation[1].

In order to illustrate this algorithm, we are going to introduce a no-left-turn example. Even though the example mentioned in this report is about no-left-turn case, we can easily adapt it into no-right-turn and one-way road cases[1].

This is a 7-node undirected graph with node 4 being the no-left-turn node [1]. That is, our navigation path should not travel from node 4 through node 5 [1]. Let node 1 be the source node and node 7 be the destination node [1].

Figure 11. Original 7-node graph as an example
This is the distance matrix of the above undirected 7-node graph, where M means infinity [1]. And because it is an undirected graph, edge length from node 1 to node 2 is the same as the edge length from node 2 to node 1 [1].

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>M</td>
<td>4</td>
<td>M</td>
<td>M</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>M</td>
<td>0</td>
<td>3</td>
<td>M</td>
<td>5</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>M</td>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>5</td>
<td>0</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>M</td>
<td>5</td>
<td>M</td>
<td>5</td>
<td>0</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>M</td>
<td>1</td>
<td>5</td>
<td>M</td>
<td>M</td>
<td>0</td>
<td>M</td>
</tr>
</tbody>
</table>

Then we need to mark node 4 as dummy node (remove it from the graph). And then directly connect every pairs of its adjacent nodes [1]. Like the graph above, node 4 has four adjacent nodes: node 3, node 5, node 2 and node 7. Therefore, we need to directly connect 6 pairs of adjacent nodes: node 3 with node 5, node 3 with node 2, node 3 with node 7, node 5 with node 2, node 5 with node 7 and node 2 with node 7 [1]. After we connect all these pairs, the length of the connected links need to be determined [1].
The figure above shows the distances of 6 pairs of adjacent nodes from either directions passing through the node 4, which is the dummy node in this case [1].

The figure above shows the distances of 6 pairs of adjacent nodes from either direction without passing through node 4, which is the dummy node in this case [1].
As shown in the above figure, the final edge length of every connected adjacent pair is determined by the minimum value of the corresponding edge length of the connected pairs with and without passing through the dummy nodes [1].

This figure shows the revised distance matrix of the undirected 7-node graph. The yellow color in the table highlights the affected values [1].
To summary this algorithm, here are the steps:

Step 1: Mark all dummy nodes as visited [1].

Step 2: Directly connect all adjacent pairs of dummy nodes [1].

Step 3: Assign edge length to each connected adjacent pair as the minimum value of edge length between two nodes under the condition of considering and not considering the dummy nodes [1].

Step 4: Apply the A* algorithm mentioned above to find the path of the given source and destinations [1].

2.2.4 Socket server

Our Android mobile application is not alone, it is accompanied by the socket server. The socket server has mainly three functionalities. First, it is receiving incident reports from users and deliver incident information to users instantly. Second, it is analyzing crawled incident information files and extracting location data from it for the purpose of showing incidents on the map. Third, the adapted A* algorithm is implemented on server side, and the corresponding results will be sent to users as navigation to avoid incidents.

3. Implementation details

In this section, I am going cover implementation details of both Android mobile application and the socket server.

3.1 Android mobile application implementation

The implementations of the Android mobile application of this incident management system is divided into three parts: Activities, Utilities and Fragments. They are going to be mentioned about one by one.
3.1.1 Activities
There are three files under Activities:
- MainActivity
- ShowIncidentInfo
- ShowMapActivity

Regarding to MainActivity activity, it is a class file to create three tabs on the screen: Map, Report and List. The tab Map is used for navigation functionality, the tab Report is used for incident report functionality and the tab List is used to show incident information from both the Hong Kong Transportation Department website and the incidents reported by our users. This activity is also the startup activity when users open up this application. And it is going to be responsible for some initialization works like establishing android’s internal SQLite databases and connecting to socket server for further interactions.

Regarding to ShowIncidentInfo activity, it is responsible for displaying the contents of incident information that is crawled from the Hong Kong Transportation Department website when users click on the title of incident under the tab List or when users click on the incident spot icons under the tab Map. Regarding to ShowMapActivity activity, it is responsible for displaying a google map user interface showing the locations the corresponding incident information when users click on the map icon under the tab List. In order to use Google Map, we need to install Google Play Services by SDK manager manually.

3.1.2 Fragments
There are five files under Fragments:
- TrafficCondition
- EmergencyReports
- EmergencyList
- EmergencyListContents
- AccidentListAdapter
Regarding to TrafficCondition, it is responsible for providing navigation service that can avoid incidents to users and displaying traffic incidents crawled from Hong Kong Transportation Department. With regard to navigation service, the current location is tracked by android’s location service and the destination is input by our users. In order for the users to be able to input the destination conveniently, we apply google places autocompletion service to help users finish inputing destination name. Finally, the navigation request is going to be sent to our socket server for routing solution that can avoid incidents and the routing solution is going to be displayed on the google map. With regard to the display of traffic incidents, it is going to retrieve incident information from server periodically and then display the location with corresponding information on google map as violet icons.

Regarding to EmergencyReports, it is responsible for collecting incident reports from our users by creating some user interface elements like EditText, CheckBox and Button. The information will be collected by corresponding asynchronous listeners and then be packaged as JSON string in order to be sent to the socket server.

Regarding to EmergencyList, it is responsible for creating three sub tabs: Hong Kong Island, New Territories and Kowloon by using FragmentTabHost, which is from the internal library. Regarding to EmergencyListContents, it is responsible for creating a ListView and then pass the AccidentListAdapter instantiated by incident information retrieved from socket server and a JSONArray to store the items of ListView to it. Regarding to AccidentListAdapter, it is responsible for displaying corresponding items when users scroll up and down the ListView, by retrieving the data from JSONArray.

### 3.1.3 Utilities

There are four files under Utilities:

- ClientSocket
- DBHelper
- GPSTracker
- LocationDataCollection
Regarding to ClientSocket, it is responsible for creating a client Socket that is going to communicate with the socket server. A new worker thread is created to hold the DataInputStream on the Socket, which will keep active for the incoming messages like incident information as long as the socket is not broken. And whenever we want to send messages to the socket server, we need to pass it the DataOutputStream on the Socket. Since there should be one and only one client Socket for each user, we are going to apply synchronized singleton pattern to promise there will be only one instance of ClientSocket at a time. And a boolean variable ‘connected’ is used to keep track of the status of the socket such that the client can re-connect to the socket when there are something wrong happens.

Regarding to DBHelper, it is responsible for holding SQLiteDatabase. This class instance should also be a singleton since we do not want to create more than one SQLiteDatabase. Otherwise, it may waste memory and make the programs confusing.

Regarding to GPSTracker, it is responsible for tracking the location of users by using the services provided by two providers: GPS_PROVIDER and NETWORK_PROVIDER. The GPS_PROVIDER has the higher accuracy than the NETWORK_PROVIDER. However, when we are indoor, GPS_PROVIDER does not work but the NETWORK_PROVIDER can still give us a rough location. Therefore, in our implementation, we will prefer the GPS_PROVIDER it is available. Otherwise, the NETWORK_PROVIDER is going to be applied. The variable MIN_DISTANCE_CHANGE_FOR_UPDATES is used to adjust the minimum distance of the location data updates. The variable MIN_TIME_BW_UPDATES is use to adjust the minimum time span when the location data is to be updated. In android, GPS service is managed by LocationManager therefore we are going to access this variable for all the location updates.
Regarding to LocationDataCollection, it is responsible for sending the location data of the clients to server per second in terms of latitude, longitude and the corresponding timestamp. In this class, we are going create a Timer that holds the data collection TimerTask. The Timer is related to only single thread, it is going to run the TimerTask that is assigned to the thread periodically at the given rate. And the location data collected by GPS service is going to be sent to the socket per second by the client Socket.

3.1.4 Database
The database framework used in this application is SQLite since we only need to store incident information that is crawled from the Hong Kong Transportation Department and incidents that is reported by our users. There is only one database in this application with the name ‘Incidents Info’. And this database only has one table that is so called ‘Incidents’ with the following fields:
- title VARCHAR(255) not null
- content VARCHAR(8000) not null
- location VARCHAR(255) not null

3.1.5 Manifest
There are some settings in the manifest file of this application. The minSdkVersion is set to be 15, which means it will not work for an android phone that has the SDK version lower than 15 since the android operating system will prevent this from happening. And we also need to require several permissions in order for the application to work appropriately:
- android.permission.ACCESS_NETWORK_STATE
- android.permission.INTERNET
- android.permission.ACCESS_WIFI_STATE/CHANGE_WIFI_STATE
- com.google.android.providers.gsf.permission.READ_GSERVICES
- android.permission.ACCESS_FINE_LOCATION
- android.permission.READ_PHONE_STATE
3.2 Socket server implementation

All files related to the socket server is contained in a folder called ‘server’, the following are the files of the server folder:
- SocketServerBDT.java
- HandleRequest.java
- Crawler.java
- location.html
- roadnetwork processing, which is a folder

Regarding to SocketServerBDT.java, it is responsible for creating ServerSocket that is listening on a specific port number. Whenever it receives a connection request from the client, it is going to create a Socket for communicating with this client. And each client connection will be allocated one new thread such that we can handle the situations of many clients.

Regarding to HandleRequest.java, it is responsible for handling read and write operations with the clients through corresponding socket. With regard to the read operation, it is managed by another thread since the thread should be blocked waiting for new incoming messages. Different incoming messages is recognized by different strings: “gps”, “incident_report”. For the “gps”, it is responsible for collecting location data sent from each client and save to database if necessary. For the “incident_report”, it is responsible for collecting incident information reported by our users and then deliver this message to all our clients. The incident information will be shown both on the list and a violet icon on the map within the mobile application. With regard to the write operation, it has only one kind of write operation: sending incident information crawled from Hong Kong Transportation Department and the incidents reported by our users to all the users. All the data exchanged on the socket through the read or write operation is in JSON format, which is prevalent among web developers.
Regarding to Crawler.java, it is responsible for collecting incident information from the Hong Kong Transportation Department and then save them into the corresponding plain text files. In order to sort the crawled contents chronologically, an internal class Link that implements Comparable<Link> is created. Each time the method start() is invoked, it is going to crawl the specified most updated incident informations. By design, the time span for crawling new incidents is 15 minutes.

Regarding to location.html, it is a file that contains all location names of Hong Kong that is in district such as Tsuen Wan, Kowloon, Central and so forth. The program is going to compare all substrings of length up to 5 in the title of incident information with the location name in this file.

Regard to the folder roadnetwork processing, it is mainly for implementing shortest path finding algorithm. Here are the files contained in this folder:
- RoadnetworkGeneration.java
- AStar.java
- Node.java
- RType.java
- Pair.java
- Experiments.java

Regarding to RoadnetworkGeneration.java, it is responsible for converting raw road network xml data format of Hong Kong from OpenStreetMap into the files that can be easily read into memory. The idea is simple, we just need to split the xml dat format by some appropriate separators. The final result will be two text files: vertices.txt and edges.txt. Each line of vertices.txt contains node identifier, latitude of the node and the longitude of the node, which are separated by single space. Each line of edges.txt contains way identifier and a series of node identifiers along the way, which are also separated by single space.
Regarding to AStar.java, it is responsible for generating routing solution that can avoid incidents based on the adapted A* algorithm. It first read in the road network data from the files vertices.txt and edges.txt. The information of nodes from vertices.txt is maintained by a HashMap with the node identifier as the key and an object Node as the value. Class Node is intentionally created for storing all necessary of graph, which will be described later. Once the road network is ready in the memory, we can get our routing solution by calling the function getPath(). The details of the internal algorithm has been mentioned in the previous sections. And as we know, the priority queue of the A* algorithm depends on two factors: the shortest path distance from the source node to the current node and the euclidean distance from the current node to the destination. The former one is updated while we are approaching the destination node, the latter one is initialized in the initialization process calling the function getEdgeLengthOnEarth, which is used to calculate the euclidean distance between two geographical coordinates on earth’s surface.

Regarding to Node.java, it is responsible for maintaining the basic information of a vertex on graph. The node identifier is stored in a long integer variable. The latitude and longitude of the node is stored in double variable. And all adjacent nodes is stored in a HashMap with the adjacent node’s Node reference as the key and the corresponding edge length as the value. There are also some other variables of class Node used for different situations, where ‘dist’, ‘ecliDist’, ‘visited’ and ‘prev’ are used for finding the shortest routing solution that can avoid incidents and ‘removed’, ‘added’ and ‘dummy’ are used for implementing an incident avoidance algorithm from a published paper.

Regarding to RType.java, it is a class that is created for storing different properties of returned values from the getPath() function that is used for finding shortest path routing solution that can avoid incidents. The variable ‘path’ is used to store a list of nodes on the final routing solution and the variable ‘count’ is used to store the number of nodes visited during the route-finding process. Note that the number of nodes visited is different from the length of the path. The number of nodes visited counts all the nodes that belongs to the shortest path tree of the source before we find the destination node.
Regarding to Pair.java, this class is mainly created for experimental purpose such that it is certain that the randomly generated pairs of source and destination nodes are distinct from each other, which is promised by the variable ‘left’ must not be greater than the variable ‘right’. And because this class is going to be used with HashTable related data structure like HashMap and HashSet, its hashCode() and equals() functions derive from the class Object are also overridden.

Regarding to Experiments.java, it is responsible for generating experimental results in terms of different situations as necessary. We first call generatePairs() to randomly generate distinct specific number of source and destination pairs, and then call the getPath() function of Astar.java with regard to different situations in order to generate appropriate experimental results.

4. Experiment
This section mainly corresponds to the previously mentioned incident avoidance algorithm. We are going to compare the performance of Dijkstra’s algorithm and A* algorithm. The performance of incident avoidance algorithm with regard to different types of incidents will also be analyzed and evaluated.

4.1 Experimental setup
The following is the system specifications of the runtime environment when I perform the experiments:
- Operating System: MacBook Pro (Retina, 13-inch, Early 2015)
- Processor: 2.7 GHz Intel Core i5
- Memory: 8 GB 1867 MHz DDR3
- Startup Disk: Macintosh HD
- Java Virtual Machine: Java HotSpot(TM) 64-Bit Server VM
- Java Runtime Environment: Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
4.2 Parameters setup
The following is the parameters that is going to be used in the experiments.
- fromLat: The latitude of the source node.
- fromLng: The longitude of the source node.
- toLat: The latitude of the destination node.
- toLng: The longitude of the destination node.
- number_of_nodes: The number of nodes that the shortest routing solution passes through.
- distance: The length of the shortest routing solution.
- time_spent: The time that the system spends on find the shortest routing solution.
- diffNode: Difference on the number of nodes visited, positive value means increasing.
- diffTime: Difference on the time spent on path finding, positive value means increasing.
- diffDist: Difference on the length of the shortest path, positive value means increasing.

4.3 A* algorithm with Dijkstra’s algorithm
In this section, we are going make experiments in order to compare the performance of A* algorithm and Dijkstra’s algorithm. 100 distinct source and destination pairs will be generated and the performance will be evaluated in terms of number of nodes visited(number_of_nodes) and time spent(time_spent) during the route finding process.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.527832:114.1950212</td>
<td>22.4287936:113.9963168</td>
<td>259927</td>
<td>27.207333573080934</td>
<td>362659749</td>
</tr>
<tr>
<td>22.650439:114.1855269</td>
<td>22.4512294:114.1664351</td>
<td>140008</td>
<td>29.89401699446879</td>
<td>413424645</td>
</tr>
<tr>
<td>22.2494584:114.2247041</td>
<td>22.2831135:114.1974182</td>
<td>25490</td>
<td>6.74950381706965</td>
<td>115237539</td>
</tr>
<tr>
<td>22.318382:113.9971676</td>
<td>22.3811875:114.2725952</td>
<td>401583</td>
<td>35.88432815540231</td>
<td>403164856</td>
</tr>
<tr>
<td>22.263946:113.5872166</td>
<td>22.3776154:114.2027606</td>
<td>307697</td>
<td>93.8552739896187</td>
<td>318359318</td>
</tr>
<tr>
<td>22.450659:114.0072142</td>
<td>22.5426175:114.1294741</td>
<td>212191</td>
<td>22.159718314955626</td>
<td>249229130</td>
</tr>
<tr>
<td>22.6135896:114.1851947</td>
<td>22.5497675:113.8829696</td>
<td>269422</td>
<td>36.35679592093917</td>
<td>293293221</td>
</tr>
<tr>
<td>22.3853073:114.1408378</td>
<td>22.3424353:114.1785395</td>
<td>60700</td>
<td>9.141276818881067</td>
<td>146578063</td>
</tr>
<tr>
<td>22.2235836:113.8470762</td>
<td>22.5383266:113.9364222</td>
<td>260283</td>
<td>50.03631184401294</td>
<td>288351578</td>
</tr>
<tr>
<td>22.3668262:114.1116295</td>
<td>22.5445425:113.9686311</td>
<td>514920</td>
<td>35.40220573529881</td>
<td>496853040</td>
</tr>
</tbody>
</table>

Figure 18. Performance of Dijkstra’s algorithm
The figure above(Figure 18) shows the first 10 items of Dijkstra’s algorithm out of 100 experimental results.
The figure above shows the first 10 items of A* algorithm out of 100 experimental results.

This figure shows the comparison results between Dijkstra’s algorithm and A* algorithm, where ‘increased’ means the corresponding item of Dijkstra’s algorithm is greater than that of the A* algorithm and ‘decreased’ has the reverse meaning. On average, the 100 experimental results is ‘increased’ on the number_of_nodes by 211232, ‘decreased’ on the time_spent by 193965931 nano seconds. The shortest path distance remains unchanged for all sample.
From these three tables, we can see that the number of nodes visited when finding the shortest routing solution is pretty much less than that of Dijkstra’s algorithm. And the shortest path distance of these two algorithms are the same, which means they are doing the right thing. For the last column, we can see that even though the number of nodes visited of A* algorithm when finding shortest path is less than that of Dijkstra’s algorithm, its system running time is longer than that of Dijkstra’s algorithm. This may deviate from our expectations. The reason for this is that the time spent on calculating the euclidean distance from a node to the destination in the initialization process is relatively expensive. To fully make use of the merits of A* algorithm, we need to perform some precomputing at the system startup process. That is, pre-calculate the euclidean distance between every pair of nodes in order to speed up running time of the route finding process of A* algorithm.

4.4 Incidents on vertices

In this section, we are going to make 100 experiments on incidents that happen on vertices and then compare its performance with the navigation without any incidents. In this experiment, we are going to focus on number of nodes visited and the length of the shortest routing solution.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1620083:113.547397</td>
<td>22.3530725:114.2495781</td>
<td>66526</td>
<td>87.09885776171866</td>
<td>493451872</td>
</tr>
<tr>
<td>22.4762345:114.0819704</td>
<td>22.426289:114.2364071</td>
<td>8053</td>
<td>31.962676350491864</td>
<td>487786750</td>
</tr>
<tr>
<td>22.2332828:114.1720929</td>
<td>22.3704238:114.1828283</td>
<td>46192</td>
<td>19.12922617987717</td>
<td>473082867</td>
</tr>
<tr>
<td>22.6302166:114.0659405</td>
<td>22.615698:114.0204572</td>
<td>83883</td>
<td>32.70802336338145</td>
<td>491634796</td>
</tr>
<tr>
<td>22.5137831:114.0801363</td>
<td>22.4771251:114.1093509</td>
<td>3734</td>
<td>7.57113545955672</td>
<td>437861158</td>
</tr>
<tr>
<td>22.5956774:113.9684339</td>
<td>22.310051:114.2423096</td>
<td>248556</td>
<td>55.76417614198057</td>
<td>677458153</td>
</tr>
<tr>
<td>22.4204434:113.9975604</td>
<td>22.563062:114.1403819</td>
<td>50077</td>
<td>27.110839670710035</td>
<td>495187834</td>
</tr>
<tr>
<td>22.2547266:114.1624183</td>
<td>22.3097496:114.2376339</td>
<td>23078</td>
<td>13.119299588611604</td>
<td>459973723</td>
</tr>
<tr>
<td>22.3052104:114.2703155</td>
<td>22.5461372:114.2246753</td>
<td>122135</td>
<td>37.76126320877033</td>
<td>534210973</td>
</tr>
<tr>
<td>22.6485515:114.0614801</td>
<td>22.128863:113.5421608</td>
<td>376869</td>
<td>109.63936189122379</td>
<td>727727425</td>
</tr>
</tbody>
</table>

Figure 21: Navigation without incidents

This figure shows the first 10 items of navigation with any incidents out of 100 experimental results.

30
Figure 22. Navigation with incidents on vertices

This figure shows the first 10 items of navigation with incidents on vertices out of 100 experimental results.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1620083:113.547397</td>
<td>22.3530725:114.2495781</td>
<td>66530</td>
<td>87.10363620602674</td>
<td>478117850</td>
</tr>
<tr>
<td>22.4762345:114.0819704</td>
<td>22.426289:114.2364071</td>
<td>81338</td>
<td>32.014744362242126</td>
<td>486956905</td>
</tr>
<tr>
<td>22.2333288:114.1720929</td>
<td>22.3704238:114.1828283</td>
<td>46198</td>
<td>19.1297694793091</td>
<td>467805102</td>
</tr>
<tr>
<td>22.6302166:114.0659405</td>
<td>22.615698:114.0204572</td>
<td>83904</td>
<td>32.712851876584516</td>
<td>485279970</td>
</tr>
<tr>
<td>22.5137831:114.0801363</td>
<td>22.4771251:114.1093509</td>
<td>3701</td>
<td>7.62522446555632</td>
<td>440136264</td>
</tr>
<tr>
<td>22.596774:113.9684339</td>
<td>22.310051:114.2423096</td>
<td>248774</td>
<td>55.76902292280874</td>
<td>690328843</td>
</tr>
<tr>
<td>22.4204434:113.9975604</td>
<td>22.563062:114.1403819</td>
<td>51134</td>
<td>27.272237233747163</td>
<td>476686657</td>
</tr>
<tr>
<td>22.2547266:114.1624183</td>
<td>22.3097496:114.2376339</td>
<td>24807</td>
<td>13.659722589011855</td>
<td>461407820</td>
</tr>
<tr>
<td>22.3052104:114.2703155</td>
<td>22.5461372:114.2246753</td>
<td>122689</td>
<td>37.82135365560145</td>
<td>525926340</td>
</tr>
<tr>
<td>22.6485515:114.0614801</td>
<td>22.128863:113.5421608</td>
<td>377169</td>
<td>109.863607888863</td>
<td>728687493</td>
</tr>
</tbody>
</table>

Figure 23. Comparison results

This figure shows the comparison results of the navigation without any incidents and the navigation with incidents on vertices, where ‘increased’ means the corresponding item of navigation with incidents on vertices is greater than that of the navigation without any incidents and ‘decreased’ has the reverse meaning. On average, the 100 experimental results is ‘increased’ by 7159 in terms of number_of_nodes and ‘increased’ by 0.2602 kilometers in terms of distance.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1620083:113.547397</td>
<td>22.3530725:114.2495781</td>
<td>increased:4</td>
<td>increased:0.0047784443087294</td>
<td>decreased:15334022</td>
</tr>
<tr>
<td>22.4762345:114.0819704</td>
<td>22.426289:114.2364071</td>
<td>increased:485</td>
<td>increased:0.05206801175027209</td>
<td>decreased:829845</td>
</tr>
<tr>
<td>22.2333288:114.1720929</td>
<td>22.3704238:114.1828283</td>
<td>increased:6</td>
<td>increased:5.43299537404726E-4</td>
<td>decreased:5277765</td>
</tr>
<tr>
<td>22.6302166:114.0659405</td>
<td>22.615698:114.0204572</td>
<td>increased:21</td>
<td>increased:0.00482851320365654</td>
<td>decreased:6354826</td>
</tr>
<tr>
<td>22.5137831:114.0801363</td>
<td>22.4771251:114.1093509</td>
<td>decreased:33</td>
<td>increased:0.0540890596965965</td>
<td>increased:2275106</td>
</tr>
<tr>
<td>22.596774:113.9684339</td>
<td>22.310051:114.2423096</td>
<td>increased:218</td>
<td>increased:0.00484685030030629</td>
<td>increased:12870690</td>
</tr>
<tr>
<td>22.4204434:113.9975604</td>
<td>22.563062:114.1403819</td>
<td>increased:1057</td>
<td>increased:0.16139756303712716</td>
<td>decreased:18501177</td>
</tr>
<tr>
<td>22.2547266:114.1624183</td>
<td>22.3097496:114.2376339</td>
<td>increased:1729</td>
<td>increased:0.5402300400251</td>
<td>increased:1434097</td>
</tr>
<tr>
<td>22.3052104:114.2703155</td>
<td>22.5461372:114.2246753</td>
<td>increased:554</td>
<td>increased:0.060094683111853</td>
<td>decreased:8284633</td>
</tr>
<tr>
<td>22.6485515:114.0614801</td>
<td>22.128863:113.5421608</td>
<td>increased:300</td>
<td>increased:0.024245997639212646</td>
<td>increased:960068</td>
</tr>
</tbody>
</table>
From the experimental results, we can see that the number of nodes visited of the shortest routing solution of navigation with incidents on vertices is generally greater than that of the navigation without any incidents. And the shortest path distance of the navigation with incidents on vertices is all greater than that of the navigation without any incidents. We can infer from experimental results that navigation with incidents on vertices will introduce some performance penalties on the number of nodes visited of the shortest routing solution and the length of the shortest path distance. However since the shortest path distance is only ‘increased’ by 0.2602 kilometers on average and our target users are drivers, this penalty should be acceptable.

4.5 Incidents on edges
When the incidents lie on edges, it may cause the whole edge to be not available or just cause some problems like no-left-turn, no-right-turn and one-way road. For the case that the whole edge is not available, that is, congestion problem, and the case that the edge is partially available, that is, no-left-turn, no-right-turn and one-way road. We are going make two groups of experiments respectively.

4.5.1 Congestion
In this section, we are going to make 100 experiments on incidents that happen on edges causing the whole edge to be unavailable and then compare its performance with the navigation without any incidents. In this experiment, we are going to focus on number of nodes visited and the length of the shortest routing solution.
Figure 24. Navigation without incidents

This figure shows the first 10 experimental results of navigation without any incidents out of 100 candidates.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.532928:113.9281529</td>
<td>22.5527205:114.1075509</td>
<td>15300</td>
<td>20.02143665016843</td>
<td>431942065</td>
</tr>
<tr>
<td>22.2745643:113.8945287</td>
<td>22.5466202:114.1499315</td>
<td>133253</td>
<td>53.2807161555741</td>
<td>520719119</td>
</tr>
<tr>
<td>22.5007598:114.1691977</td>
<td>22.1872278:113.5475358</td>
<td>446737</td>
<td>97.3217523681333</td>
<td>789378874</td>
</tr>
<tr>
<td>22.3608075:114.0977979</td>
<td>22.3313332:114.2300271</td>
<td>40925</td>
<td>17.18772970594431</td>
<td>447873625</td>
</tr>
<tr>
<td>22.2772852:114.1778184</td>
<td>22.3996442:114.1913213</td>
<td>50342</td>
<td>16.405206787914047</td>
<td>463180447</td>
</tr>
<tr>
<td>22.2871001:114.1326955</td>
<td>22.302369:114.1814013</td>
<td>15849</td>
<td>7.35472635644026</td>
<td>429200530</td>
</tr>
<tr>
<td>22.3849225:114.2752707</td>
<td>22.2575139:114.2090707</td>
<td>71840</td>
<td>23.09265595600931</td>
<td>466983739</td>
</tr>
<tr>
<td>22.6259768:113.5261313</td>
<td>22.3449329:114.1775861</td>
<td>51449</td>
<td>169.33066578403648</td>
<td>449336425</td>
</tr>
<tr>
<td>22.3759429:114.1074702</td>
<td>22.2107941:113.5600363</td>
<td>179166</td>
<td>75.0471514730945</td>
<td>540835927</td>
</tr>
<tr>
<td>22.5145316:114.0641068</td>
<td>22.4901013:114.1351819</td>
<td>6472</td>
<td>12.002173762921482</td>
<td>422058027</td>
</tr>
</tbody>
</table>

Figure 25. Navigation with congestion

This figure shows the first 10 experimental results of navigation with incidents on edges that cause the whole edge to be not available out of 100 candidates.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.532928:113.9281529</td>
<td>22.5527205:114.1075509</td>
<td>15300</td>
<td>20.048554706406822</td>
<td>426650400</td>
</tr>
<tr>
<td>22.2745643:113.8945287</td>
<td>22.5466202:114.1499315</td>
<td>134604</td>
<td>53.38830656205688</td>
<td>519621400</td>
</tr>
<tr>
<td>22.5007598:114.1691977</td>
<td>22.1872278:113.5475358</td>
<td>449246</td>
<td>97.54476476965516</td>
<td>793613352</td>
</tr>
<tr>
<td>22.3608075:114.0977979</td>
<td>22.3313332:114.2300271</td>
<td>41384</td>
<td>17.214504438497517</td>
<td>452178791</td>
</tr>
<tr>
<td>22.2772852:114.1778184</td>
<td>22.3996442:114.1913213</td>
<td>50305</td>
<td>16.412536809583077</td>
<td>465519281</td>
</tr>
<tr>
<td>22.2871001:114.1326955</td>
<td>22.302369:114.1814013</td>
<td>16080</td>
<td>7.394344318310181</td>
<td>434825400</td>
</tr>
<tr>
<td>22.3849225:114.2752707</td>
<td>22.2575139:114.2090707</td>
<td>71486</td>
<td>23.092928518302163</td>
<td>469743684</td>
</tr>
<tr>
<td>22.6259768:113.5261313</td>
<td>22.3449329:114.1775861</td>
<td>52347</td>
<td>169.41904161430443</td>
<td>446163947</td>
</tr>
<tr>
<td>22.3759429:114.1074702</td>
<td>22.2107941:113.5600363</td>
<td>179369</td>
<td>75.06916127244848</td>
<td>539149528</td>
</tr>
<tr>
<td>22.5145316:114.0641068</td>
<td>22.4901013:114.1351819</td>
<td>6513</td>
<td>12.006348076214845</td>
<td>421678450</td>
</tr>
</tbody>
</table>
This figure shows the comparison results of the navigation without any incidents and the navigation with incidents on edges that cause the whole edge to be not available, where ‘increased’ means the corresponding item of navigation with congestion is greater than that of the navigation without any incidents and ‘decreased’ has the reverse meaning. On average, the 100 experimental results is ‘increased’ by 4220 in terms of number_of_nodes and ‘increased’ by 0.5930 kilometers in terms of distance.

From the experimental results, we can see that the number of nodes visited of the shortest routing solution of navigation with incidents on edges that causes congestion is generally greater than that of the navigation without any incidents. And the shortest path distance of the navigation with incidents on edges that causes congestion is all greater than that of the navigation without any incidents. We can infer from the experimental results that navigation with incidents on edges that causes congestion will introduce some performance penalties on the number of nodes visited of the shortest routing solution and the length of the shortest path distance. However since the shortest path distance is only ‘increased’ by 0.5930 kilometers on average and our target users are drivers, this penalty should be acceptable.
4.5.2 No-left-turn, no-right-turn and one-way road

In this section, we are going to make 100 experiments on incidents that happen on edges causing the whole edge to be partially unavailable, that is, the situations of no-left-turn, no-right-turn and one-way road. And then compare its performance with the navigation without any incidents. In this experiment, we are going to focus on number of nodes visited and the length of the shortest routing solution.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.603165:114.217995</td>
<td>22.5889039:113.937754</td>
<td>33546</td>
<td>37.896038959926095</td>
<td>495825946</td>
</tr>
<tr>
<td>22.3390419:114.2589133</td>
<td>22.2812921:114.2053968</td>
<td>29230</td>
<td>12.740138442697658</td>
<td>851928381</td>
</tr>
<tr>
<td>22.2476059:114.1544281</td>
<td>22.4018824:114.325525</td>
<td>146387</td>
<td>33.23419655168723</td>
<td>579078774</td>
</tr>
<tr>
<td>22.300078:113.9356946</td>
<td>22.5501279:113.9469686</td>
<td>18609</td>
<td>35.20598619855842</td>
<td>467745876</td>
</tr>
<tr>
<td>22.4160466:114.334986</td>
<td>22.1394388:113.5602322</td>
<td>402268</td>
<td>103.81805731611952</td>
<td>814443963</td>
</tr>
<tr>
<td>22.3071864:114.0137125</td>
<td>22.5444641:114.1060838</td>
<td>83796</td>
<td>40.0641153926738</td>
<td>534363487</td>
</tr>
<tr>
<td>22.3129126:113.9379754</td>
<td>22.5656947:114.1777516</td>
<td>140889</td>
<td>53.176789490755496</td>
<td>571712579</td>
</tr>
<tr>
<td>22.312706:114.0416714</td>
<td>22.3458422:113.5721227</td>
<td>309094</td>
<td>88.5742855590604</td>
<td>681357672</td>
</tr>
<tr>
<td>22.3196987:114.2536126</td>
<td>22.6519753:113.9958787</td>
<td>267678</td>
<td>57.31986160375712</td>
<td>702245052</td>
</tr>
<tr>
<td>22.4466229:114.1346014</td>
<td>22.6050594:113.9116824</td>
<td>112860</td>
<td>38.10617996889002</td>
<td>545524672</td>
</tr>
</tbody>
</table>

Figure 27. Navigation without incidents

This figure shows the first 10 experimental results of navigation without any incidents out of 100 candidates.

<table>
<thead>
<tr>
<th>fromLat:fromLng</th>
<th>toLat:toLng</th>
<th>number_of_nodes</th>
<th>distance</th>
<th>time_used(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.603165:114.217995</td>
<td>22.5889039:113.937754</td>
<td>33546</td>
<td>37.896038959926095</td>
<td>620937144</td>
</tr>
<tr>
<td>22.3390419:114.2589133</td>
<td>22.2812921:114.2053968</td>
<td>29230</td>
<td>12.740138442697658</td>
<td>483627031</td>
</tr>
<tr>
<td>22.2476059:114.1544281</td>
<td>22.4018824:114.325525</td>
<td>146387</td>
<td>33.23419655168723</td>
<td>581664375</td>
</tr>
<tr>
<td>22.300078:113.9356946</td>
<td>22.5501279:113.9469686</td>
<td>18609</td>
<td>35.20598619855842</td>
<td>467830593</td>
</tr>
<tr>
<td>22.4160466:114.334986</td>
<td>22.1394388:113.5602322</td>
<td>402268</td>
<td>103.81805731611952</td>
<td>833345153</td>
</tr>
<tr>
<td>22.3071864:114.0137125</td>
<td>22.5444641:114.1060838</td>
<td>83796</td>
<td>40.0641153926738</td>
<td>525228120</td>
</tr>
<tr>
<td>22.3129126:113.9379754</td>
<td>22.5656947:114.1777516</td>
<td>140889</td>
<td>53.176789490755496</td>
<td>566311969</td>
</tr>
<tr>
<td>22.312706:114.0416714</td>
<td>22.3458422:113.5721227</td>
<td>309094</td>
<td>88.5742855590604</td>
<td>680264191</td>
</tr>
<tr>
<td>22.3196987:114.2536126</td>
<td>22.6519753:113.9958787</td>
<td>267678</td>
<td>57.31986160375712</td>
<td>698722619</td>
</tr>
<tr>
<td>22.4466229:114.1346014</td>
<td>22.6050594:113.9116824</td>
<td>112860</td>
<td>38.10617996889002</td>
<td>545424199</td>
</tr>
</tbody>
</table>

Figure 28. Navigation with incidents

This figure shows the first 10 experimental results of navigation with incidents on edges that cause the whole edge to be partially unavailable out of 100 candidates.
This figure shows the comparison results of the navigation without any incidents and the navigation with incidents on edges that cause the whole edge to be partially unavailable, where ‘increased’ means the corresponding item of navigation with no-left-turn, no-right-turn and one-way road is greater than that of the navigation without any incidents and ‘decreased’ has the reverse meaning. On average, the 100 experimental results is ‘increased’ by 6796 in terms of number_of_nodes and ‘increased’ by 0.1468 kilometers in terms of distance.

From the experimental results, we can see that the number of nodes visited of the shortest routing solution of navigation with incidents on edges that causes no-left-turn, no-right-turn and one-way road is generally greater than that of the navigation without any incidents. And the shortest path distance of the navigation with incidents on edges that causes the edge to be partially available is generally greater than that of the navigation without any incidents. We can infer from the experimental results that navigation with incidents on edges that causes congestion will introduce some performance penalties on the number of nodes visited of the shortest routing solution and the length of the shortest path distance.

However, there are compelling amount of items that is only ‘increased’ by 1 in terms of number_of_nodes, and at the same time remains unchanged. This fact means that by applying the adapted A* algorithm proposed by the paper mentioned above to this kind of situation, it introduces very small performance penalty on both the number of nodes visited and the shortest path distance of the shortest routing solution.
5. Difficulties encountered

5.1 Differentiation on different users
As mentioned above, the target user of our mobile application would be bus passengers in Hong Kong because we may have a large user base. And the passengers can play the role as driver when they are on the buses, and they can also play the role as pedestrians when they are on the roads. The difficulty here is that we may not be able to exactly recognize when the passengers are playing the role as a driver and when they are playing the role as a pedestrian. Our current solution is to keep track on the moving speed of our application user per minute. Whenever there is a sudden change on the moving speed, the system wakes up and compare the current moving speed of the user with the average speed of common vehicles. Based on this approach, we may be able to alleviate this difficulty to some degree.

5.2 Deficiencies on incident avoidance algorithm
As mentioned in previous section, an adapted A* algorithm will be used as the algorithm to avoid incidents for navigation. The deficiency of this algorithm is that incident spots may exist on the vertices or edges. When it is existent on the vertices, if we directly mark it as visited in the initialization process then all the adjacent edges will be invalid. In that case, we may lose useful information and end up with suboptimal or even no suitable routing solution. When it is existent on the edges, we are lucky because we just need to remove the specific edges and with most information retained. Therefore, there should be some tradeoffs between optimality and the ability of incident avoidance in the future improvement of this algorithm.
6. Future work

There are still some spaces for improvement in this project:

- Incident report can be in voice form. Although our user interface for incident report functionality is quite convenient for users, the user experience can still be improved if we can provide voice input for users when they want to report any incidents that happen on the roads.

- Integrating incident detection algorithm into this project. The core of this project is to provide navigation that can avoid incidents based on the incident information that is crawled from the Hong Kong Transportation website and the incidents that are reported by our users. However for crawled incident information, we need to design some methods to extract location data from the plain text files. This kind of extraction may not have hundred percent accuracy. Therefore it is better for us to have our own incident detection algorithm so that we can avoid the inaccuracy of location data extraction algorithm.

- Further improvement of incident avoidance algorithm. Incident avoidance algorithm is a big topic, the method introduced in this project is relatively restricted in two aspects. First, it is not dynamic. Whenever we detect an incident, we have to recalculate the whole path in order to avoid it, which may be insufficient in terms of both the number of nodes nodes visited during the route finding process and the time spent on find the best routing solution. Second, the incident is only considered as a spot. In our algorithm, we only consider the incident as spots. However in reality, it may have affecting areas. In this case, we can suggest a routing solution that is passing through the affecting areas of some incidents.
## 7. Project schedule and milestones

<table>
<thead>
<tr>
<th>Date</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 2, 2016</td>
<td>Detailed project plan&lt;br&gt;Project webpage</td>
</tr>
<tr>
<td>Oct-Jan</td>
<td>Mobile user interface design and three mobile application functionalities:&lt;br&gt;- Incident report&lt;br&gt;- Incident information&lt;br&gt;- Navigation to avoid incidents</td>
</tr>
<tr>
<td>Jan 22, 2017</td>
<td>Preliminary implementation&lt;br&gt;- Mobile user interface&lt;br&gt;- Incident report functionality&lt;br&gt;- Incident information functionality:&lt;br&gt;sources of incident information is from Hong Kong Transportation&lt;br&gt;- Navigation to avoid incidents: decided to implement adapted A* algorithm as incident avoidance algorithm for navigation&lt;br&gt;Detailed Interim report</td>
</tr>
<tr>
<td>Feb-Apr</td>
<td>Optimization of mobile application and server systems</td>
</tr>
<tr>
<td>April 16, 2017</td>
<td>Finalized tested implementation&lt;br&gt;- User interface is fully refined&lt;br&gt;- Three main functionalities of android mobile application is fully implemented and tested&lt;br&gt;- Location data extraction algorithm is fully refined and tested&lt;br&gt;- Incident avoidance algorithm is fully refined and tested&lt;br&gt;Final report</td>
</tr>
</tbody>
</table>
8. Conclusion

In conclusion, we currently already have a prototype of android mobile application. we have implemented basic user interface and all three main functionalities of our android mobile application. Regarding to incident information functionality, we first crawled trustworthy incident information from Hong Kong Transportation and then extract useful location data from it by our location data extraction algorithm. Regarding to navigation functionalities, we first downloaded all necessary data of Hong Kong’s road networks from OpenStreetMap and then applied our incident avoidance algorithm to provide navigation that can avoid incidents to users. Regarding to incident report functionality, user input the details of incidents on the android mobile application and then our socket server is responsible for save it and send this incident information to all other users.

For the socket server, it has mainly three functionalities. First, it is receiving incident reports from users and delivering incident information to users as soon as it is received. Second, it is analyzing crawled incident information files from the Hong Kong Transportation Department and extracting location data from it for the purpose of showing incidents on the map. Third, the adapted A* algorithm that can avoid incidents is implemented on server side, and the corresponding results will be sent to users as navigation to avoid incidents.
9. References

WIT Transactions on The Built Environment, Vol 77, © 2005 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)

   https://developers.google.com/places/android-api/autocomplete

   https://developers.google.com/maps/documentation/geocoding/start

[4] Sockets programming in Java

   https://www.javatpoint.com/multithreading-in-java

   https://developers.google.com/maps/documentation/android-api/

[7] OpenStreetMap Nominatim API
   http://wiki.openstreetmap.org/wiki/Nominatim

[8] Jsoup [A web crawler in java]
   https://jsoup.org/