QATAR UNIVERSTIY

COLLEGE OF ENGINEERING

SAFETY AWARE VEHICLE ROUTING ALGORITHM, A WEIGHTED SUM APPROACH

BY

RADWAN NIZAM

A Thesis Submitted to
the Faculty of the College of
Engineering
in Partial Fulfillment
of the Requirements
for the Degree of
Masters of Science in Computing

June 2017

COMMITTEE PAGE

The members of the Committee approve the Thesis of Radwan Nizam defended on 06/01/2017

Prof.Abbes Amira

Thesis Supervisor

Dr.Khaled Shaban

Committee Member

Dr.Khaled Khan

Committee Member

Dr. Fatih Mutlu

Committee Chair

Approved:

Khalifa Al Khalifa, Dean of College of Engineering

ABSTRACT

NIZAM, RADWAN, FAISAL, Masters
June:2017, Master of Science in Computing
Title: Safety Aware Vehicle Routing Algorithm, a Weighted Sum Approach
Supervisor of Thesis: Prof. Abbes, Amira

Driving is an essential part of work life for many people. Although driving can be enjoyable and pleasant, it can also be stressful and dangerous. Many people around the world are killed or seriously injured while driving. According to the World Health Organization (WHO), about 1.25 million people die each year as a result of road traffic crashes. Road traffic injuries are also the leading cause of death among young people. To prevent traffic injuries, governments must address road safety issues, an endeavor that requires involvement from multiple sectors (transport, police, health, education). Effective intervention should include designing safer infrastructure and incorporating road safety features into land-use and transport planning.

The aim of this research is to design an algorithm to help drivers find the safest path between two locations. Such an algorithm can be used to find the safest path for a school bus travelling between bus stops, a heavy truck carrying inflammable materials, poison gas, or explosive cargo, or any driver who wants to avoid roads with higher numbers of accidents. In these applications, a path is safe if the danger factor on either side of the path is no more than a given upper bound. Since travel time is another important consideration for all drivers, the suggested algorithm utilizes traffic data to consider travel time when searching for the safest route. The key achievements of the work presented in this thesis are summarized as follows. Defining the Safest and Quickest Path Problem (SQPP), in which the goal is to find a

short and low-risk path between two locations in a road network at a given point of time. Current methods for representing road networks, travel times and safety level were investigated. Two approaches to defining road safety level were identified, and some methods in each approach were presented. An intensive review of traffic routing algorithms was conducted to identify the most well-known algorithms. An empirical study was also conducted to evaluate the performance of some routing algorithms, using metrics such as scalability and computation time. This research approaches the SQPP problem as a bi-objective Shortest Path Problem (SPP), for which the proposed Safety Aware Algorithm (SAA) aims to output one quickest and safest route. The experiments using this algorithm demonstrate its efficacy and practical applicability.

ACKNOWLEDGMENTS

I would like to convey my heartfelt gratitude and sincere appreciation to all people who have helped and inspired me throughout the completion of this research.

First and foremost, I am grateful to my supervisor Prof. Abbes Amira, for the continued support during this research, for his patience, motivation and enthusiasm. His guidance helped me in all the time of research and writing of this thesis, and he still is my reference in self-confidence, perseverance, hard work, and organization.

In addition, I am indebted to Dr. Fethi Filali and Dr. Elyes Ben Hamida from Qatar Mobility Innovation Center (QMIC) for their help and support. My thanks are also due to Dr. Mohamed Bin Mokhtar Kharbeche from Qatar Transportation and Traffic Safety Studies Center to the data he provided for this work.

Finally, I would like to thank my family who have supported me throughout my whole study, and who tolerated my extremely busy schedule which sometimes prevented me from spending time with them even at the weekends. My deepest gratitude goes especially to my wife; completing my thesis would not have been possible without her unfaltering and continuous support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

∞	Travel Time Weighting Factor
DA	Dijkstra Algorithm
iRAP	International Road Assessment Program
NRSS	National Road Safety Strategy
OSM	OpenStreetMap
QP**A**	Quickest Path Algorithm
QPHA	Quickest Path Algorithm – Heuristic
QPP	Quickest Path Problem
QTTSC	Traffic Safety Center
RSA	Road Safety Assessment
SAA	Safety Aware Algorithm
SFPP	Safest Path Problem
SP	Safest Path
SPP	Shortest Path Problem
SQPP	Safest and Quickest Path Problem
SUMO	Simulation of Urban Mobility – Traffic Simulator
TDSPP	Time-Dependent Shortest Path Problem
TraCI	Traffic Control Interface
UNGA	United Nations General Assembly
WHO	World Health Organization

WHO	World Health Organization
β	Safety Level Weighting Factor

1 INTRODUCTION

Car ownership is increasing annually in many countries [1]. Although cars increase convenience for people, they also contribute to increasing traffic congestion and road accidents, which result in public demand for road safety solutions worldwide. The transportation sector in Doha, for example, is rapidly growing, and the country's research priorities include mitigating road accidents. These research initiatives led to a significant drop in the number of road traffic deaths between 2010 and 2011 [2], as shown in Figure 1.

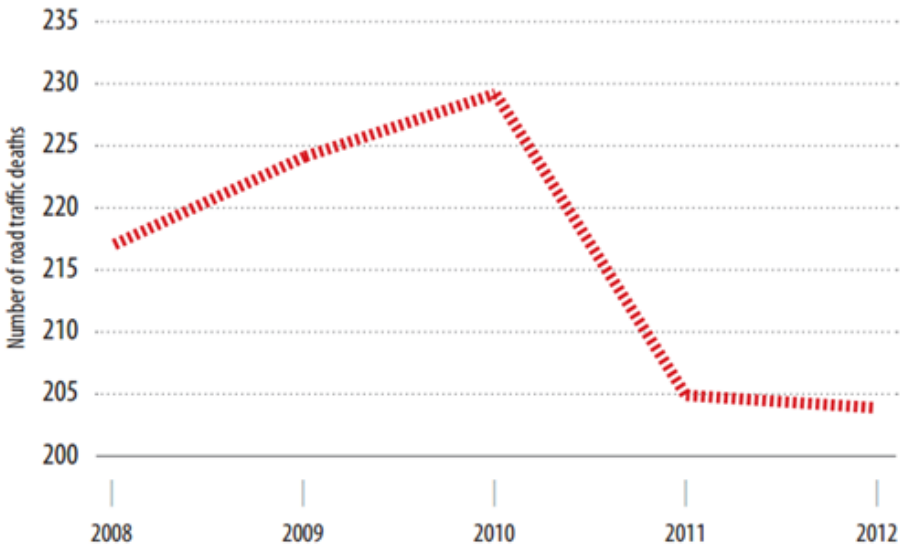


Figure 1 – Trends in Reported Road Traffic Deaths in Qatar [2]

According to The Global Status Report on Road Safety 2015 [2], the number of road traffic crashes has steadily declined in recent years, despite the growing number of registered vehicles in Qatar. Nonetheless, over 200 people are still killed every year due to traffic accidents, of which 34% are drivers, 38% passengers, 28% pedestrians and 90% men. To address the problem of road accidents, Qatar has

adopted a number of key road safety practices, such as a demerit/penalty point system, national speed limits and a seat belt law.

The safe routing of vehicles has become a broad area of research. Nowadays, there are many applications for routing based on travel time and safety. Individuals, industries and businesses that are reliant on driving (such as logistics and distribution and taxi drivers) could all benefit from a routing system designed to find the shortest, safest route. Optimal vehicle routing is generally provided through applications that apply variables such as distance, historic traffic data and real-time traffic data to calculate the efficient route between two locations in a road network. Currently, there are many free vehicle routing services, such as Google Maps and OpenStreetMaps (OSM), that provide optimal vehicle routing solutions but do not consider safety factors.

This work defines the problem SQPP, a bi-objective shortest path problem in which the goal is to minimize both travel time and risk of the route. To achieve this, the road network is imported from OSM and converted to a digital network using the traffic simulator Simulation of Urban Mobility (SUMO). Both the travel time and the safety level for each road are stored in the digital network. This work proposed the algorithm SAA to find the quickest and safest route between two locations in the network.

1.1 Roads Safety

To calculate the relative risk of travelling on a route, the risk must be quantified and aggregated across every segment in the route. In this research, the risk of travelling on a specific road is referred to as road safety level, which can be estimated using reactive and proactive approaches.

To calculate the relative risk of travelling on a route, the risk must be quantified and aggregated across every segment in the route. The risk of travelling on a specific road is referred to as road safety level, which can be estimated using reactive and proactive approaches. A reactive approach to road safety involves the identification of locations experiencing safety problems, while a proactive approach emphasizes the prevention of safety problems before they manifest themselves in patterns of crash occurrences [3].

A reactive approach to road safety is based on the analysis of existing crash data. For example, data mining techniques can be used to identify certain road accident features associated with high frequency of accidents on a road [4]. A reactive approach identifies safety problems and proposes a set of improvements to optimize the safety level of the road. The reactive approach has some limitations: (1) it requires the identification of high crash locations before suggesting improvement plans, and (2) since improvement plans are implemented on a road already built and open to the public, the cost of the improvements is very high. The reactive approach to road safety is a core component of the safety management system, as it is a powerful tool for addressing existing safety problems.

A proactive approach focuses on the evolving Science of Safety. For example, regression algorithms can be used to predict the safety performance of the roads [5]. These algorithms explain crash performance based on physical and operational variables such as intersection form and traffic volume. A proactive approach can be applied in the road design process or used to suggest improvement plans on existing roads to diminish the risk of crashes prior to their reconstruction. Advantages of a proactive approach include that it: (1) prevents car accidents as it does not rely on existing crash data, and (2) avoids extra costs for implementing an improvement plan on a road currently open to the public.

This research follows a proactive approach, in which the road safety level is determined through RSAs [6]. RSA is a formal assessment of the safety performance of an existing or planned road segment or intersection. An RSA can be performed during any or all stages of a project; it can be a tool for public agencies to improve road safety levels. This study uses the safety model of iRAP [7]. iRAP is a registered charity dedicated to preventing the more than 3,500 road deaths that occur every day worldwide. The activities of this charity include inspecting high-risk roads, developing Star Ratings assessments based on road inspection data and providing a simple and objective measure of safety levels inherent in the roads for vehicle occupants, motorcyclists, bicyclists and pedestrians. Five-star roads are the safest, while one-star roads are the least safe. In Qatar, the Public Works Authority 'Ashghal' signed a memorandum of understanding (MoU) with iRAP [8] to adopt and implement iRAP's road safety standards to evaluate roads. This effort aims to raise the safety levels on the roads to a three star minimum; hence, the result of the assessment can be applied to this work in the future.

1.2 Quickest Route

The SPP problem that involves positive and static edge weights is one of the most studied problems in graph theory. In the context of vehicular transportation, since the length of the roads is positive, the proposed algorithms can be used to generate the shortest path between two locations in the network, and by using the travel time on the roads as a weighting function, it is possible to generate the quickest route between locations. In reality, however, road networks tend to have different flow speeds at various times due to reasons such as accidents, rush hours and weather. Hence, the generated route at a given time may not be optimal upon considering predictable future changes. Two approaches are used to tackle this issue: (1) as the traffic situation may change from time to time, the quickest path is re-computed when the vehicle reaches an intersection. This recalculation should verify whether the current route is still the optimal one [9]. (2) Another approach is called the Time-Dependent Shortest Path Problem (TDSPP), in which flow speed on the road changes with time in a predictable fashion. The problem initially dates back to 1966, when it was first proposed in discrete time by Cooke and Halsey [10]. TDSPP assumes that the travel time along each arc is a function of the departure time along the arc and that all such functions are known in advance over all values of time.

1.3 Motivation

Road crashes have enormous health and economic impact in Qatar [11]. Road crashes also significantly impact local communities and society as a whole through infrastructure repair costs, lost output, traffic delays and the demand on police and medical resources. The National Road Safety Strategy (NRSS) 2013-2022 was built to reduce the human suffering inflicted by road traffic crashes, with an ambitious long-term vision of developing "a safe road transport system that protects all road users from death and serious injury."

Globally, in response to the large impact of road crashes around the world, the United Nations General Assembly (UNGA) proclaimed the period 2011–2020 as the United Nations Global Decade of Action for Road Safety. The Decade of Action was launched on 11 May 2011 with the goal of stabilizing and then reducing the forecasted level of global road fatalities by 2020 through increasing activities conducted at national, regional and global levels. In response to this initiative, Qatar's Road Safety Strategy has developed a safety system in accordance with the principles of the framework defined by the Decade of Action. The system helps identify eight key areas of concern for priority action in Qatar. One key area is road design, which can contribute significantly to the number and severity of crashes. For example, many roads in Qatar are divided using a median, but there are often no crash barriers to prevent vehicles from crossing over the median and striking opposing traffic. Intersections in Qatar also vary in suitability for the roads where they are located.

The large incidence of road accidents in Qatar has prompted research centers in Qatar to contribute to the national strategy, providing approaches to significantly reduce road accidents and building Intelligent Transportation Systems. The Qatar

Transportation and Traffic Safety Center (QTTSC) studies patterns of accidents, factors that contribute to road accidents and drivers' attributes and makes recommendations for approaches to improved road safety. Qatar Mobility Innovations Center (QMIC) uses its platform Masarak to provide real-time and historic traffic information, incident detection and traffic prediction, utilizing real-time traffic information to provide intelligent trip planning and other functions.

Prioritization of transportation safety in Qatar and the ongoing work of research centers have produced accurate data such as traffic data, travel time estimation, road safety levels and number of accidents on the roads. This body of traffic and safety data has prompted the creation of solutions for generating safe and quick routes for car travel in Qatar. Such solutions, which help people avoid risky roads and thereby reduce the number of road accidents, can be used in various scenarios. The solution can be used to build a routing service, a core function of any Intelligent Transportation System, and an on-board routing system, in which traffic data and safety data are collected using connected-vehicles technology that allows vehicles to exchange information about traffic jams and road accidents. Social media platforms, such as Twitter, are other sources of information about traffic jams and road accidents. Integrating these platforms with the proposed systems would lead to more optimal solutions.

1.4 Problem Statement

The problem here is to find the quickest and safest route between two locations in a road network, at a specified departure time and an accepted safety level. In this work, the problem is called the Safest and Quickest Path Problem (SQPP). The SQPP is decomposed into three problems that are, (1) finding the quickest route only when any safety level is accepted. (2) finding the safest route, when targeting the highest safety level. (3) finding the route that combine both the safety and the travel time.

It is clear that SQPP is similar to the SPP, in both problems; the objective is to find the optimal solution based on a specified condition. SPP was addressed by many algorithms, Dijkstra Algorithm (DA) [12] and A* [13] were proposed by many researches as good solutions to find the shortest route in a real road network [14] [15], and by using the travel time on the road as a weighting function, the algorithms generate the quickest route.

As mentioned earlier, travel time in transportation networks could vary over time, hence, the required time to travel on a road, depends on the entry time of that road. This problem is known as TDSPP, which already heavily studied by many researches [16] [10] [17] [18]. To tackle TDSPP a road network is represented as a time-based graph, the junctions of the roads represent the nodes of the graph and the road segments represent the edges between the nodes, and the travel time between two nodes at a specified departure time, is given using a time-based function, also an adapted version of Dijkstra algorithm is used to find the dynamic shortest path, this approach is implemented and evaluated by many works in the literature like [18] and

[19] . Figure 2 shows an area in Doha and Figure 3 shows the graph used to represent that area, the black circles and lines represents nodes and edges.

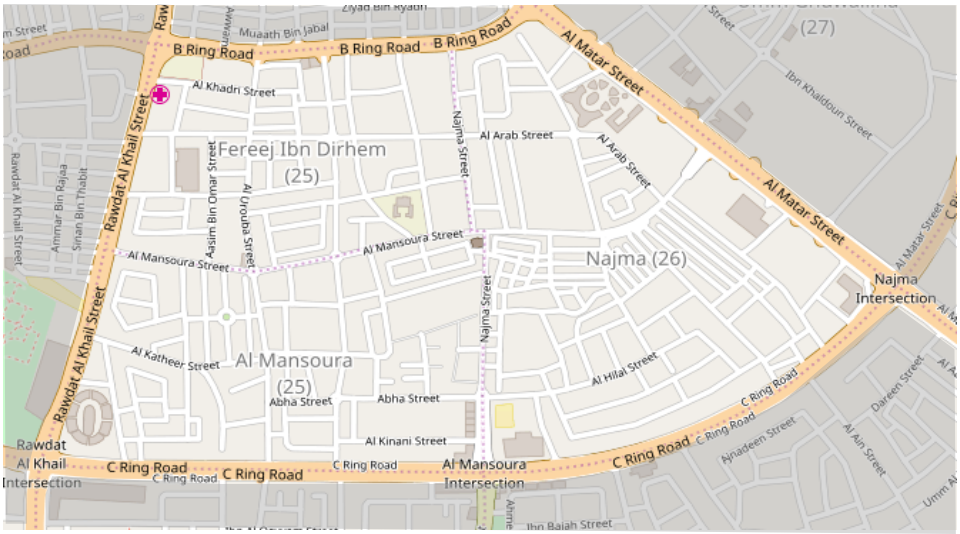


Figure 2 - Area in Doha

Figure 3 – Area in Doha Represented Using a Graph

As mentioned, in a time-based graph, the travel time between two nodes at a specified departure time, is given using a time-based function, by adapting this

function to return the travel time or the safety level, the sub-problems (1) and (2) can be tackled. The third sub-problem, is a bi-criteria optimization problem, where the goal is to minimize both the travel time as well as the risk of the route, to tackle this problem, the solution comes up with an equation to combine the two objectives in a single one.

1.4.1 Time Dependent Graph Example

Figure 4 shows a road network modelled as a time-dependent graph $G_t(V, E, W)$, figure (a) shows its graph (V, E) with four nodes and five edges. The edge-delay functions for the edges $(\mathrm{v1}, \mathrm{v2}), (\mathrm{v2}, \mathrm{v3}), (\mathrm{v1}, \mathrm{v3}), (\mathrm{v2}, \mathrm{v4})$ and$(\mathrm{v3}, \mathrm{v4})$, are shown in figures (b), (c), (d), (e) and (f) respectively. Table 1 shows the possible solutions to travel from $v1$ to $v4$ at $t1 = 10$ and $2 = 50$. According to the travel times, P2 is the best solution to reach $v4$ when a driver left $v1$ at $t1$ and $t2$.

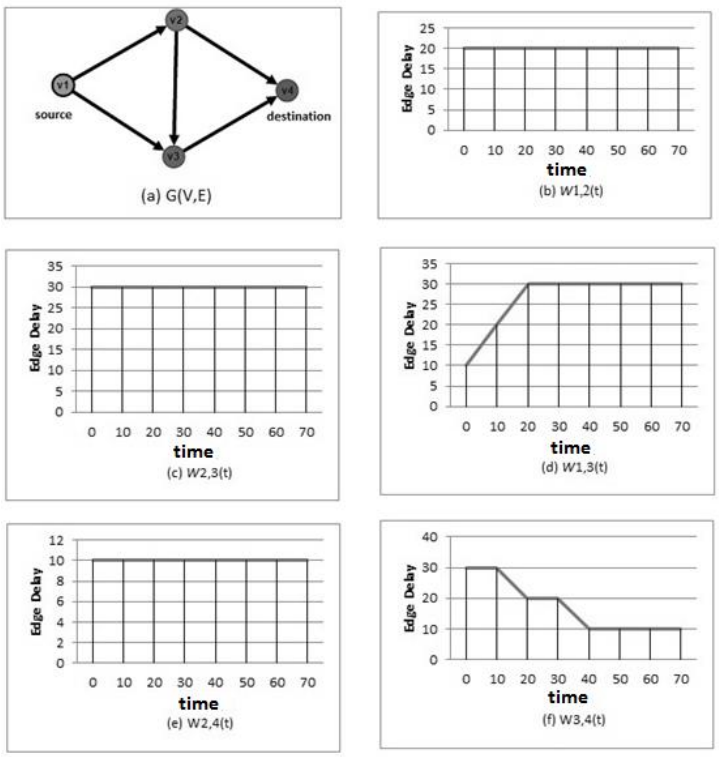


Figure 4 - Time-Based Graph $\boldsymbol{G_t(V, E, W)}$

Table 1- Time Based Graph Example – Routes

Time	Solution	Travel Time
$t1 = 10$	P1 = {v1, v2}, {v2, v3}, { v3, v4}	80
	P2 = {v1, v2}, {v2, v4}	30
	P3 = {v1, v3}, {v3, v4}	50
$t2 = 50$	P1 = {v1, v2}, {v2, v3}, { v3, v4}	100
	P2 = {v1, v2}, {v2, v4}	30
	P3 = {v1, v3}, {v3, v4}	40

1.5 Research Objectives

The overall research aim in this dissertation is to develop an algorithm to provide the quickest and safest route between two points at a given time. Hence the main objectives of this thesis can be summarized as follows:

- Carry out a thorough literature review about different existing routing algorithms;
- Evaluate and analyze the available methods used for road networks representation;
- Perform an empirical study for A* and Dijkstra based routing algorithms using different performance metrics such as scalability, computation time and complexity;
- Design and implementation of a modified version of A* and Dijkstra algorithms using time-dependent graph to find the quickest route;
- Design and implementation of a hybrid technique for safety aware routing using both the length and risk of the path; and
- Develop a framework for validating the proposed safety aware routing algorithms using SUMO platform;

1.6 Research Contributions

The major contribution of this work is SAA algorithm which can find the optimal path in the following cases:

- The quickest route with no consideration to safety level.
- The safest route with no consideration to travel time.
- The safest and quickest route where the minimum safety level is provided as an input parameter.

The proposed algorithm can be used to build safety aware routing services in Qatar. Such services can be used by the public to find safe routes for normal cars, or heavy truck carrying poison gas, inflammable materials, or petrol, and traveling in a city.

For representing road networks, two methods were identified for this purpose. The first method uses directed graph, while the second one uses time-based graph, which makes the second one a good option to get the accurate travel time between two nodes at a specified time. In terms of risk assessment and quantifying risks, the used methods and techniques are divided into two approaches, reactive and proactive. As the proposed algorithm deals with the safety level on the road, the outcome in any of these methods and techniques can be used as an input for the proposed algorithm. In the conducted experiments, as there is no available roads' safety level for Qatar road networks, random values were generated and assigned to the roads. In the area of shortest path algorithms, an intensive review of the related work for finding the shortest path and quickest path was conducted; the objective was to identify the known algorithms in this area, taking into consideration two factors that are, the computation time, and the possibility to consider different factors when finding the

streets in the whole route. Also, two routing algorithms, to find the quickest path, were implemented using different approaches. Furthermore, to solve the SQPP problem, the work reviewed different strategies for solving the problem, and suggests an approach that gives the optimal solution by combining the two objectives, the safety and the travel time. For traffic simulation and route verification, a parameterized integration between the proposed algorithm and the simulation environment was implemented. The integration allows to conduct a benchmark for some routing algorithms, and to verify the output of the proposed algorithm using different parameters.

2 RELATED WORK

In this study, the shortest path problem in a time dependent network and the road safety estimation are two major concerns. The ultimate aim of this chapter is to provide a summary of related work in three areas:

1. Shortest path problem using static graphs
2. Shortest path problem using time-dependent graphs.
3. Safety aware algorithms and approaches to quantitative road safety level.
4. Quickest path problem using V2V communications
5. Strategies for solving the Bi-criteria Shortest Path Problem

2.1 Shortest Path Problem Using Static Graphs

SPP problem have significant practical implications in different areas like computer science and vehicle routing. Classical SPP problems with fixed arc lengths have been studied intensively, resulting in the development of a number of efficient algorithms [17]. Among the developed algorithms, Dijkstra Algorithm (DA) is one of the most used algorithms in the literature because of the good performance of this algorithm [14] [15] [20]. In the following, an overview of some works that evaluated the performance of DA, also some DA variations are included in this section.

In [14]. The performance of 15 routing algorithms was evaluated using real road networks. Three shortest path algorithms that run quickest on real road networks were identified. These algorithms are 1) the graph growth algorithm implemented with two queues, 2) the DA implemented with approximate buckets, and 3) the DA algorithm implemented with double buckets. The study recommends DA to obtain a

one-to-one shortest path because it can be terminated as soon as the shortest path distance to the destination node is obtained.

The work in [15] has evaluated the performance of DA and A* algorithm. A* algorithm is a popular extension of DA that reduces the number of visited nodes when finding a shortest path, if additional information is available that provides a lower bound on the 'distance' to the target. Also, the work evaluated two variant algorithms, that are Dynamic DA, and Dynamic A* algorithm that find the fastest routes. A* use the Euclidean distance between the source node and the target node as a lower bound to travel between the two locations, the distance is calculated when the algorithm selects a new node in the graph. The lower bound in Dynamic A* is the actual time required to reach the destination, the lower bounds are calculated periodically by static all-to-all DA and the results are stored in memory. The work evaluated the performance of the four algorithms in different network size (center has 4025 nodes, suburban has 2597 nodes and remote has 1810 nodes) and different trip length (2km, 4km, 6km, 8km and 10km). As shown in Figure 5, Figure 6 and Figure 7 the computation time for all the algorithms is proportional to the scenario scalability level as well as the trip length. In general, it is clear that the A* and Dynamic A* outperforms DA and its variant due to the heuristic approach of A*. Also, Dynamic A* performs better than A* because the statas the latter needs to calculate the lower bound during its execution while Dynamic A* just loads the lower bound it needs into the memory. As a result, DA is recommended to find the shortest route in the remote areas due to its low complexity and good performance in terms of computation time, DA is recommended in the central and suburban areas for short trips (i.e. $\leq$ 4km), and A* is recommended in the central and suburban areas for long trips (i.e. $\geq$ 6km).

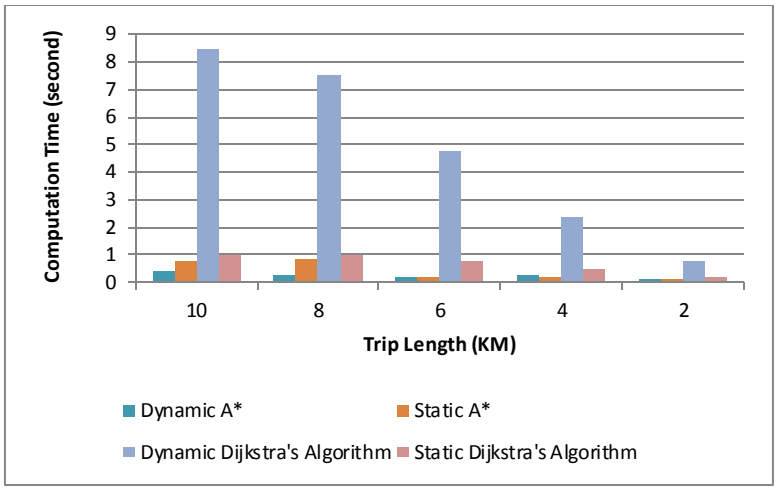


Figure 5 – Computation Time in Center Area in [15]

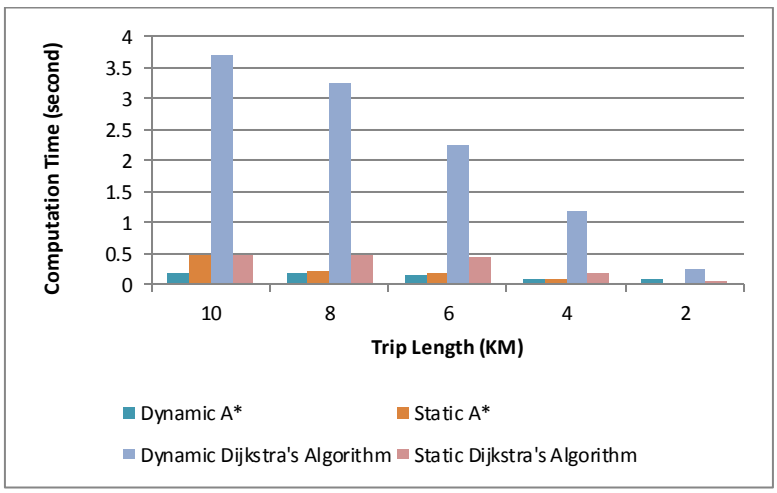


Figure 6 – Computation Time in Suburban Area [15]

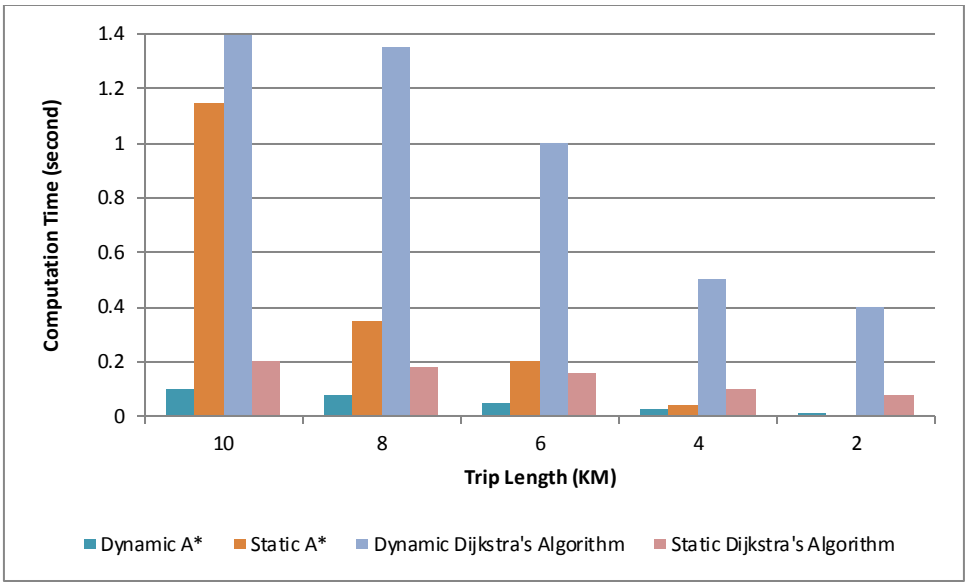


Figure 7 – Computation Time in Remote Area [15]

The performance of DA is a problem of interest for many researchers, F. Benjamen [20] mentioned that the performance of the shortest path algorithm, that follow the labeling method, depends on the strategy used to select the next node to be scanned, and the used data structure to maintain the set of unlabeled nodes. For the first aspect, the commonly used strategies are FIFO, LIFO and Best-First-Search. For the next aspect, the data structures include buckets, linked lists and priority queues like Fibonacci heaps.

In [21] DA was used to provide the optimal route, the weight function in this work considers road length, traffic congestion and road quality to find the optimal path. Each of these factors has a nonnegative weight coefficient between zero and one which allow giving different priorities for those factors.

In the area of cargo route planning, in [22] the work suggests an algorithm to find the shortest path for oversized cargo transportation, the suggested algorithm considers the cost of directions at road intersection, those directions are turning left,

going straight, turning right, and turning back. The algorithm uses a dual graph to describe the steering relationship and restrictions between adjacent roads; in dual graph, the original edge of the road map (which is the road on which trucks drive before making a turn) becomes a node, and the turning direction in the original road map is represented by an edge. As shown in Figure 8, edge de in the auxiliary network based on road corners stores not only the weight value of the turn from road ab to road bc, but also the weight value of the turn from road bc to road ab. They are both functional values related to the value of the turn angle. Meanwhile, the auxiliary arc in the auxiliary network based on road corners also stores the real distance covered by the trucks.

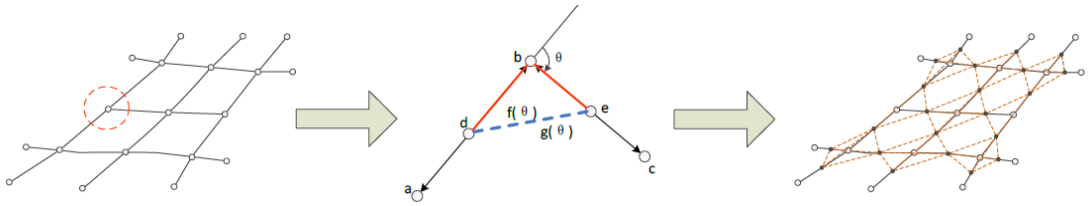


Figure 8 – Road Network as Dual Graph [22]

The nodes in the dual graph contain information about the corresponded edges in the road map such as the cost of passing that edge. Also the edge in the dual graph contains information about the cost of going from the original edge of the destination edge or vice versa. DA is used to find the shortest path between two nodes in the dual graph.

In [23] in order to provide an optimal route during the trip, the work propose a dynamic traffic factor based on time function, which is implemented to the Dijkstra algorithm calculation, Figure 9 shows an example of network for time base dynamic

weight DA . To implement the dynamic traffic factor based on time, a traffic profile for each road is maintained, the profile describes how much time is needed to pass the road at a specified time. The traffic profile can be made different between different conditions like workdays, weekend or holiday. The limitation in this algorithm is that the algorithm calculates the travel time on all edges in a single point of time, it does not consider the required time to travel between the edges of the selected route.

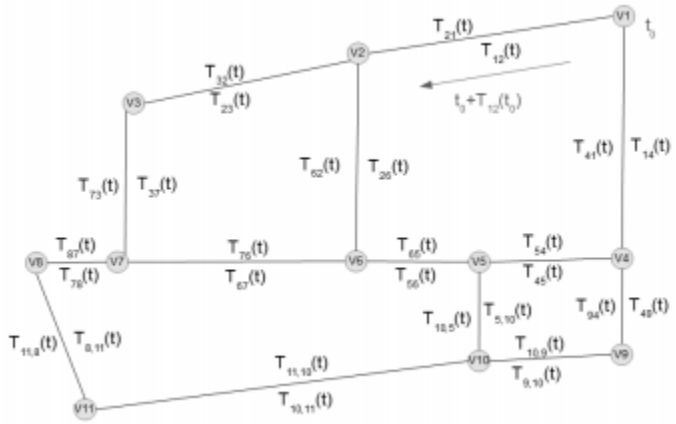


Figure 9 –Example of Network for Timebase Dynamic Weight Dijkstra Algorithm [23]

2.2 TDSPP Problem

The TDSPP problem initially dates back to 1966, when it was first proposed by Cooke and Halsey [10]. Cooke and Halsey proposed a modified form of Bellman's iteration scheme [24] for finding the shortest route between any two vertices. Dreyfus [17] shows that this problem is quite similar to the well-known static shortest path problem and it can be solved by a trivially-modified variant of any label-setting static shortest path algorithm. In the literature, we found that the well-known routing algorithm DA has been used for developing a number of efficient algorithms like [18].

The work in [18] deals with the problem of computing the earliest arriving time $EA_{s*}(t)$, in which the objective is to find the shortest path leaving some source node s at a particular departure time t. The result shows that $EA_{s*}(t)$ can be solved by a trivially-modified variant of any label-setting shortest path algorithm. In this research, a modified version of DA is used, where the weight of an arc is represented by required time to travel on that arc. The asymptotic running time of the modified algorithm exhibit the same performance in practice as their static counterparts, this result was achieved by taking into consideration that the running time of DA depends on the implementation of the priority queue *S*; according to the research, the strongest known running time, $O(m + n \log n)$, is achieved when *S* is a Fibonacci heap. Algorithm 1 shows the modified version of DA.

Algorithm 1 - Label-Setting (Dijkstra's) Algorithm

1. ***for all*** $i \in N \backslash \{s\}: EA_{si}(t) \leftarrow \infty$
2. $EA_{ss}(t) \leftarrow t$
3. $S \leftarrow N$
4.
5. ***While*** $S \neq \emptyset$
6. ***Select*** $i \in S$ *minmizing* $EA_{si}(t)$
7. $S \leftarrow S \backslash \{i\}$
8. ***for all*** j *such that* $(i, j) \in A$
9. $EA_{si}(t) \leftarrow \min(EA_{si}(t), a_{ij}(EA_{si}(t)))$

2.2.1 Finding Best Departure Time to Reduce Travel Time

A variant problem of the TDSPP problem is finding the best departure time for minimizing the total travel time from a place to another over a road network, where the driver has the flexibility to start the trip during a time interval, and to wait for some time in some nodes across the route. This problem has been studied widely and intensively over years. An application of this problem can be a products distribution company that delivers products using trucks. A truck may travel to a place with less travel time, if it delays the trip for some time, e.g. one or two hours, the company can utilize the truck to do other jobs during this time. The main challenges to find the optimal route in this problem, is that edge delay changes as the starting time changes [25]. Different approaches were used to solve the best departure time problem, for example, in [16] a discrete–time approach is suggested, where the starting time interval is discretized into *k* time points, and for each time point an equivalent static graph is created. The new graph has same edges, nodes and delay on every edge at that time point. By finding the shortest path in the *k* graph, the path with the minimum travel time will be the path with the best departure time. This approach has two drawbacks. First, having low value of *k* will result in an inaccurate result, as the

optimal solution can be in an unconsidered time point. Second, having big value for k will result in more computation to find the optimal solution.

2.3 Safety Aware Algorithms and Approaches

In general, safety aware algorithms use a risk model for the road network, which is essentially an assignment for a risk factor to each edge. As mentioned earlier, the risk factor is calculated using a reactive or proactive approach. In the reactive approach, the crash history can be used to predict the likelihood of future deaths and serious injuries [26], also crime data can be used to provide safe urban navigation [27]. However, in the proactive approach, the characteristics of the road and traffic can be examined in order to identify the parts of road network which have the greatest potential risk [28]. In the following, an overview of some works that proposed approaches and algorithms for providing safe routing services.

In [27] an algorithm was proposed in building novel application that utilizes crime data to provide safe urban navigation system. The algorithm aims to generate a small set of paths that provide tradeoffs between distance and safety. The work represents the road network using an undirected graph $G= (V, E)$ where the set of nodes V, represents intersections, and the set of edges E represents streets between the intersections. Each road segment $e \in E$ is associated with two (unrelated) types of weights: its length, denoted by $l(e)$, and its risk, denoted by $r(e)$. $l(e)$ gives the distance between the two intersections connected by the street e, and $r(e)$ gives the probability that a crime will be committed on that segment. Starting from the fact the problem here is a bi-criteria optimization problem, the algorithm searches for the no dominated safe solution P_r^* (safest) and the non-dominated length solution P_l^*

(quickest), in another step, the algorithm searches for all solutions in the region specified by the 2-dimensional representation of the two solutions as shown in Figure 10.

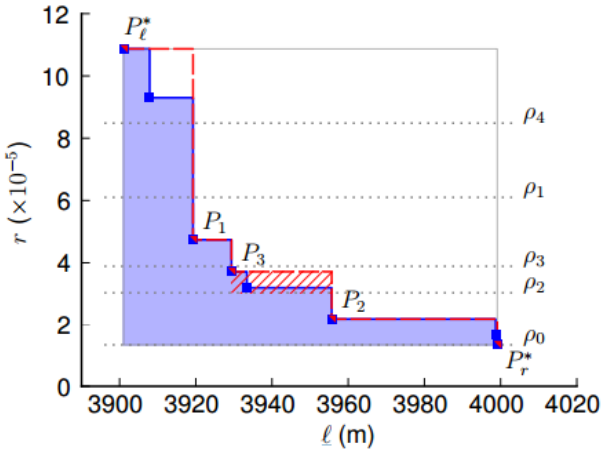


Figure 10 – Solution Space in [27]

The work in [28] is a part of the project SAFEWAY2SCHOOL [29], which suggests a new method to solve the School Bus Routing Problem (SBRP) with consideration for the safety of the children while on board. The method advises to define an assessment criteria which can be used to define a safety score for each link in the transportation network. For example, a systematic examination can be made of the accident statistics to find out which parts of the network have a record of accidents. Another approach is to examine the characteristics of the road and traffic in order to identify the parts of road network which have the greatest potential risk; as an example of features which can be responsible for safety risks in rural and urban road environments are:

Rural areas:

- Sharp bends
- Lack of protective barriers on the road shoulder

- A narrow two way roads without a central partition

Urban areas:

- Lack of good visibility for making turns (due to buildings or on street parking)
- Fast roundabouts
- Complex intersections

In the suggested method, each risk factor has a weight from 0 to 5 (5 indicates a strong influence and 0 no influences), and of this risk factor each road section is assigned with a score from 1 to 5. The method then can provide a risk category for each road, the proposed categories are:

- Green means the road is considered relatively safe.
- Yellow, means the road is considered borderline.
- Red means the road involves a significant safety risk.

The research in [30] compares two road risk assessment methodologies for building the risk model in a vehicle routing service. The first method is a predictive risk assessment from the Economic Evaluation Manual (EEM), which includes procedures to calculate the safety benefits and costs of transport projects (New Zealand Transport Agency, 2013). This risk assessment relies on regression algorithms that explain crash performance based on physical and operational variables, such as intersection form and traffic volumes. The second method tested is the Urban KiwiRAP methodology, which relies on crash history to predict the likelihood of future deaths and serious injuries if current crash trends continue. As a result, through testing it was evident that Urban KiwiRAP was the preferable choice

for determining the risk of roads. It was noticed that the EEM risk metric tended to route onto low volume local roads because fewer crashes would be expected to occur in these environments; however, these are often impractical for routing purposes. The EEM models avoided high volume roads (such as motorways) even when there were comparatively fewer crashes, which is not unexpected as the models return average values for safety performance based on the sample of data from which they were created. The personal risk metric from Urban KiwiRAP contrasted the results from the EEM. It analyzed many roads with a high AADT as low risk due to them having a relatively low number of crashes for the amount of road use. Some lower volume roads with a low number of crashes were classed as higher risk as the ratio of crashes to vehicle volume was greater.

In [31] an algorithm was proposed to find the shortest safe path between two nodes, and also it suggests the best trip starting time to reduce the overall travel time between the two nodes. The algorithm works in two steps. First, it follows a continuous-time approach to identify the arrival times for each node in the graph. For each starting point of time during the provided start time period, there will be an arrival time for an edge. Second, it selects the arrival times to the target point, sort the time in an ascending order to get the best arrival time to the target node. Then, it determines the predecessor of a node on the route utilizing the backward manner from the target node to the starting node.

2.4 Quickest path problem using V2V communicatoins

In [32] a VANET-based A* route planning algorithm was introduced to dynamically calculate the route that meets the shortest travelling time or the lowest fuel consumption criteria, the algorithm uses Google Maps to access the real-time traffic data of road segments and the data collected from the neighboring vehicles using VANET networks. This algorithm uses an improved version of A* algorithm. The work in [33] designed an algorithm that can be used in navigation systems to provide real time traffic information in Tabriz, Tabariz is Iran's fourth largest city with a population of about 1.400.000, the work proposed a scheme for acquiring real-time traffic information based on an amalgamation of VANET networks and both inter-vehicle communication (IVC) and vehicle road-side device communication (VRC) and conventional systems, the work used the A* algorithm with an appropriate cost function.

Table 2 presents all the routing algorithms discussed in this section

Table 2 - Summary of Some Routing Algorithms

Work	Graph type	Factors in weight function	Consider Safety	Safety level approach	Algorithm	Applications
[16]	dynamic	travel time	-	-	Dijkstra and Label Correcting Algorithm	route planning & computing early arrival time
[21]	static	road length & traffic congestion & road quality	considered	-	Dijkstra	route planning
[22]	static	steering angles & travel time along the road	-	-	Dijkstra	cargo route planning
[23]	static	travel time	-	-	Dijkstra	route planning
[27]	static	travel time & safety level	considered	Crime statistics	Dijkstra	urban-navigation
[28]	static	road distance & safety level of the road	considered	Proactive	-	safe door-to-door school $ bus transportation services
[30]	-	-	considered	iRap classification in	Dijkstra	-

				Urban KiwiRAP		
[31]	dynamic	travel time & safety level on the road	considered	-	-	route planning
[32]	static	travel time or fuel consumption	-	-	A*	route planning
[33]	static	travel time	-	-	A*	route planning

2.5 Strategies for solving the Bi-criteria Shortest Path Problem

The shortest path problem in road network is one of the classical single-objective optimization problems that have gained attention from researchers worldwide. However, in some real application there is a set of constraints that should be considered when finding the optimal route, which means the necessity of taking more than one objective into account, resulting in biobjective shortest path (BSP) problems that have been addressed by many multi-objective optimization methods and algorithms. Instances of optimization problem with single objective are resolved in polynomial complexity bound, while (BSP) is classified as NP-complete [35]. This section reviews some methods to resolve the biobjective shortest path problem, i.e. the scalarization technique, label correcting technique, two phases method and a genetic algorithm based technique.

2.5.1 Scalarization Technique

Also called the weighted-sum technique. This technique solves the multi-objective problem by combining the objectives into a single-objective scalar function, and then it minimizes the new objective [36]. This technique uses the concept of Pareto optimal to define the set of optimal solutions.

2.5.1.1 Bi-objective Shortest Path and Pareto-optimal Solutions

A single-objective optimization problem is formulated as follows:

$$\min f(x)$$

$$x \epsilon S,$$

Where f is a scalar function and S is the set of objectives that can be defined as:

$$S = \{x \in R^m : h(x) = 0, g(x) \geq 0\}$$

Multi-objective optimization is formulated as follows:

$$\min[\, f1(x), f2(x), \ldots, fn(x)]$$

$$x \epsilon S,$$

Where n>1and S is the set of objectives. In this context, the objective space is defined as the space in which the objective vector belongs. Also, the attained set is defined as the image of the feasible set under F.

$$C = \{y \in R^n : y = f(x), x \in S\}$$

The notion of Pareto optimality in multi-objective optimization is defined as the following:

A solution with a vector of constraints $x^* \in S$ in a multi-objective problem is Pareto optimal, if all other vectors of constraints $x \in S$, in the other solutions, have a higher value for at least one of the objective functions f_i, this leads to define the weak Pareto optimum and strict Pareto optimum as the following:

- A weak efficient solution or a weak Pareto optimum if there is no $x \in S$ such that $fi(x) < fi(x^*)$ for all $i \in \{1, 2, \ldots, n\}$
- A strict efficient solution or a strict Pareto optimum if there is no $x \in S$ such that $fi(x) \leq fi(x^*)$ for all $i \in \{1, 2, \ldots, n\}$ with at least one strict inequality.

Now Pareto front is defined as the set of all efficient solutions. The shape of the Pareto front identify the trade-off between the objective functions. Figure 11 is an example of a Pareto front which is defined by the points between $(f2(\bar{x}), f1(\bar{x}))$ and $(f2(\bar{\bar{x}}), f1(\bar{\bar{x}}))$, these points are called the non-dominated points.

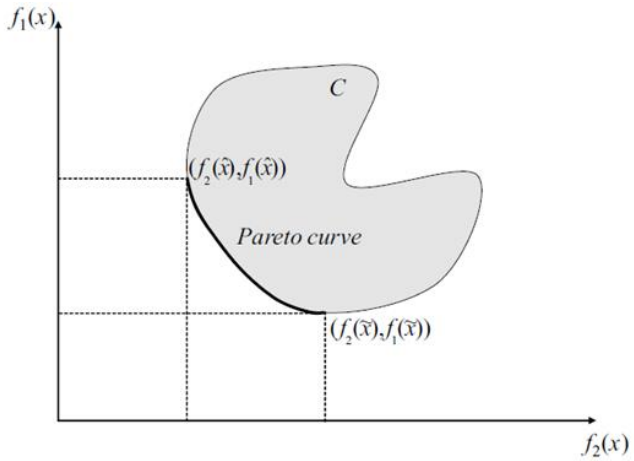


Figure 11 – Example of a Pareto curve [36]

Also shows two weak Pareto optimum that are p1 and p5 and three strict Pareto optimum that are p2, p3 and p4

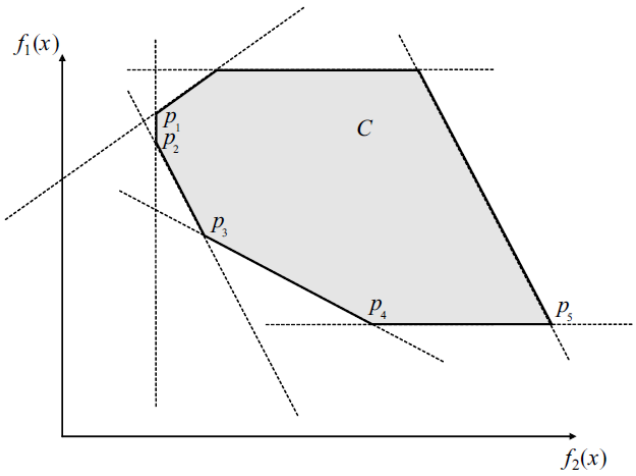


Figure 12 - Example of weak and strict Pareto optimum [36]

2.5.1.2 Solving biobjective shortest path using scalarization technique

This scalarization technique solves the multi-objective problem by combining the objectives into a single-objective scalar function, and then it minimizes the new objective function that is:

$$\min \sum_{i=1}^{n} y_i \,.\, f_i(x)$$

$$\sum_{i=1}^{n} y_i = 1$$

$$\mathrm{y}_i > 0, i = 1, \dots, n$$

$$x \epsilon S,$$

Where n is the number of constraints, S is the set of constraints, f is a scalar function and y is a weighting factor. The new optimization problem is denoted by: $P_s(y)$.

For any multi-objective problem there is more than one optimal solution. According to [36] that solution of the $P_s(y)$ problem is an efficient solution for the original problem, i.e. it belongs to the Pareto front. This technique generates both the strict Pareto optimum and the weak Pareto optimum solutions according the following conditions:

$$\min \sum_{i=1}^{n} y_i \,.\, f_i(x)$$

$$\sum_{i=1}^{n} y_i = 1$$

$$\mathrm{y}_i \geq 0, i = 1, \dots, n$$

$$x \epsilon S,$$

A strict Pareto optimum solution is generated when all the weights in vector *y* are greater than zero, while a weak Pareto optimum solution is generated when at least one of the weights in the vector *y* is equal to zero. The negative side of this technique is that it is not clear how to change the weights in order to generate an optimal solution. Which means it is not easy to develop an algorithm that can finds the optimal weight factors to reach a point in the Pareto front. This leads to the result that in order to find other optimal solutions it requires that more optimizations with different weight values should be performed. However, this approach can produce a considerable computational load.

A shortcoming of this technique is that a uniform spread of weight parameters may not produce a uniform spread of points in the Pareto front. It is observed that all solution is grouped in some parts of the Pareto front and other parts of the curve have not been produced. As an overall evaluation for this technique, if a single-objective scalar function with O(1) time complexity is used with a polynomial algorithm, like any of the shortest path problem algorithms, then this method can be used to find only one optimal solution in the Pareto font in polynomial time.

2.5.2 Biobjective label correcting

This method is an extension of the single-objective version. The difference between the two methods is that there are several labels at each node; each label corresponds to one path. Skriver and Andersen [37] found that the best approach to solve BSP problems is the label correcting with node-selection.

2.5.2.1 Formal definitions

A directed network G = (N, A) consists of a set of nodes N and a set of arcs A c N × N, with a length function len: A → $\mathbb{R}^2$. The length vector for an arc a contains the constraints $\mathrm{len}_1(\mathrm{a})$ and $\mathrm{len}_2(\mathrm{a})$, respectively. In this network, a path P which contains a list of distinct nodes is a sequence of arcs $[\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_\mathrm{k}]$ with $\mathrm{a}_\mathrm{i} = (\mathrm{w}_\mathrm{i}, \mathrm{w}_{\mathrm{i}+1}), \mathrm{i} = 1, \ldots \mathrm{K}$. The length vector of the path is denoted by the equation

$$d(p) = (d1(p), d2(p))$$

$$d_j(P) = \sum_{i=1}^{i} len_j(a_i), \qquad j = 1,2$$

In this network, let's $\mathcal{P}_j$ be a set of paths designated from the source *s to* the destination j. A vector of constraints $x = (x_1, x_2)$ at an arc is said to dominate the vector $y = (y_1, y_2)$, if $x \leq y$ *i.e.* $x_1 \leq y_1$ and $x_2 \leq y_2$.

Let's X be a set of vectors where not element in this set is dominated by another (distinct) element in the same set. In this case, the set X is defined as undominated set. This definition is used to define the undominated path as the following:

A path $P_1 \in \mathcal{P}_j$ dominates path $P_2 \in \mathcal{P}_j$ if vector for P_1 dominates the vector for P_2 and the length of the two vectors are different.

The set $\mathcal{E}_j$ of efficient (s, j) paths contains all paths $P \in \mathcal{P}_j$not dominated by another path in $\mathcal{P}_j$, also the corresponding set of efficient path length vectors is denoted by $\mathcal{F}_j$. Skriver and Andersen [37] consider that the BSP problem here is finding the set $\mathcal{E}_j$ for all $\in N$. Their approach is to find the sets $\mathcal{F}_j$ which represents the distinct path length vectors of the paths in $\mathcal{E}_j$. A backtracking scheme can be used to find the paths corresponding to a path vector in $\mathcal{F}_j$.

2.5.2.2 Biobjective label correcting algorithm

Each node i has a set of labels. A label l at node i is extended by all edges (i, j) with start node i. The extended label l + cij is added to the label set at node j if it is not dominated. The labels dominated by the new label at node j are deleted. Also, a non-dominated extended label l + cij at j has to be reconsidered in a later iteration. The pseudo code of Skriver and Andersen algorithm is shown in Algorithm 1 and described below.

Initially, the list of nodes that should be expanded, i.e. modNodes, is initialized with the source node *s*. The labels node *s* is initialized with 0: Labels(s) = {(0, 0)}. When running the algorithm, the labels at a particular node *i* are extended along the edge *(i, j)*, if the extended label is dominated by already presented labels at the end node *j*, then the edge *(i,j)* is removed from the list of adjacent of node *i* . If the label set of node *j* is changed, then the node is marked to be expanded again. The algorithm terminates when modNodes is empty. Merging is the most expensive operation in the algorithm; the operation is executed when traversing an outgoing arc from a node with multiple labels. All labels should be extended along that arc and verified for dominance. In this algorithm, the number of non-dominated labels grows exponentially in the number of nodes. That is, all the efficient paths have a distinct non-dominated value, which means that the node-labeling algorithms have exponential complexity.

Algorithm 2 - Biobjective Label Correcting

1. ***input***: network *(N,A)*, cost function $c = (c^1, c^2)$, source node *s*
2. modNodes = {s}: list of nodes with modified labels that have not yet been reconsidered, treated in FIFO order
3. Labels(s) = {(0, 0)} and Labels(i)= ∅, i ∈ N\{s}: Labels(i) is the list of labels at a particular node *i*
4. **while** *modNodes* is nonempty **do**
5. remove first node *i* from *modNodes* /∗ FIFO ∗/
6. **for all** *(i, j)* with $j \in \{k \in N | (i, k) \in A\}$ **do**
7. *merge(Labels(i) + cij , Labels(j))* /∗ extend all labels at *i* by *cij* and merge with labels at *j*, eliminating all dominated labels ∗/
8. **if** the label set of *j* has changed and $j \notin$ *modNodes* **then**
9. append *j* to modNodes /∗ FIFO ∗/
10. **end if**
11. **end for**
12. **end while**
13. **output:** efficient route length from the node *s* to all other nodes

The bicriterion path problem is inherently difficult [38], where in a worst case scenario the process of calculating the efficient paths grows exponentially with the network size, in contrast to the single objective shortest path problem for which many polynomial algorithms exist [39].

2.5.3 Two phase method

The method is suggested in [40] to solve the bicriteria optimization problem. This method finds the optimal solutions in two phases. The first phase will generate the extreme optimal solutions based on a weighted sum problem technique, an example of the output of this phase is shown in Figure 13 where lex(1, 2)-best and lex(2, 1)-best are extreme optimal solut8ions. The first phase is started after generating one or two optimal solutions using an initial phase.

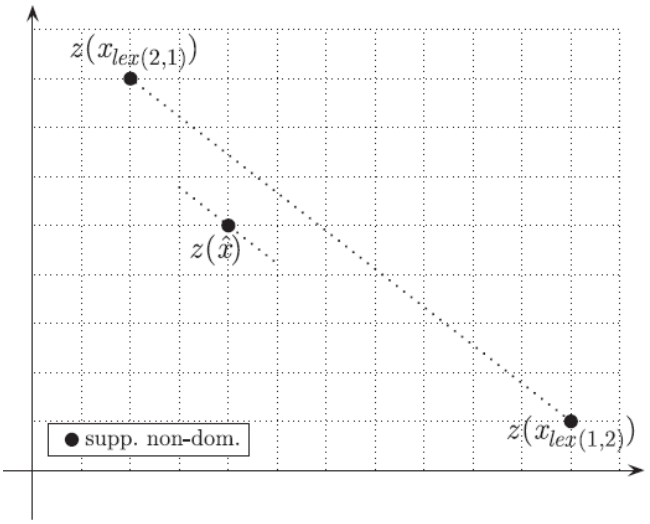


Figure 13 – Ditchotomic method, first iteration [41]

The second phase generates the remaining efficient solutions using an enumerative approach. In general, the search space in the second phase is highly restricted due to the bounds obtained in first phase. As shown in Figure 14, the search space in this phase is restricted to triangles defined by two consecutive extreme optimal points. The first phase is started after generating one or two optimal solutions using an initial phase.

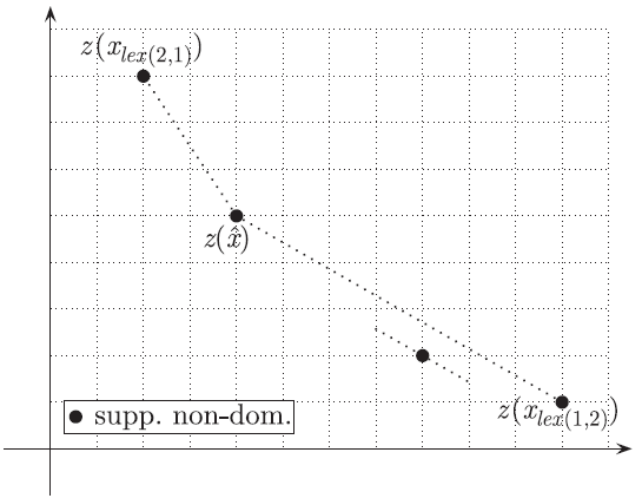


Figure 14 – Dichotomic method, second iteration [41]

2.5.4 Genetic-algorithm based approach

A bicriteria genetic algorithm to solve bicriteria shortest path problems is proposed in [42].The work proposed that each edge in a direct graph carries two attributes denoted by (c_{ij}^1, c_{ij}^2) . The objective is to find the optimal path between a source node $s \in N$ and a destination node $t \in N$. An example of this representation is provided in Figure 15.

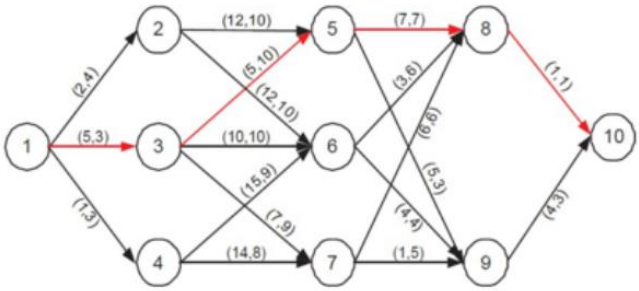


Figure 15 – Road Network [42]

2.5.4.1 Genetic representation

A chromosome is represented using a fixed-length string, and each locus of the chromosome represents a node in the graph. The value of each gene is 0 or 1, (1 if the corresponding node exists in the path, 0 if not).The gene of the first locus is always reserved for the source node and its value is one and the last locus is always reserved to the destination node and its value is one. Figure 16 shows a chromosome representation in this method.

Locus	1	2	3	4	5	6	7	8	9	10
chromosome	1	0	1	0	1	0	0	1	0	1

Figure 16 – Chromosome Representation [42]

2.5.4.2 Initial Population

As shown in Figure 17, initial population is divided into a number of sub-populations; one population for each objective. A sub-population is selected base on the value of the objective Ck. In those sub-populations, the first individual represents the extreme optimal solution which is generated using Dijkstra algorithm, while reminder of the population is generated randomly.

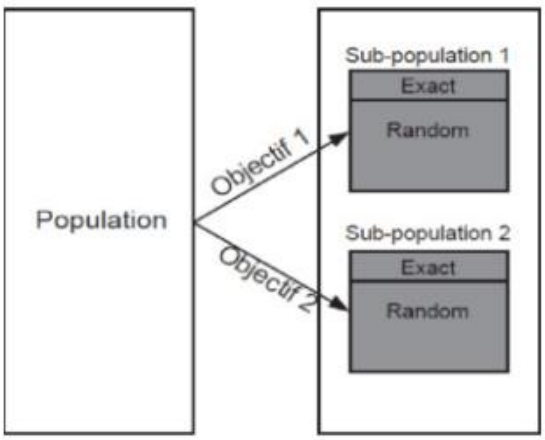


Figure 17 - Initial population composition [42]

2.5.4.3 Crossover

The crossover concerns each sub-population separately, and the offspring produced by this phase represents complete routes. Any two chromosomes, chosen for crossover, should have at least one common (node) except for source and destination nodes, and the generated offspring should have less cost than their parents.

Figure 18 is an example of crossover of 2 parents. When optimizaing C^2 , the parital route from source node 1 to the intermediate node 5 in the crossover is relative to the parent 1 that have the best $C^2 = 13 < 14$. The partial route from the intermediate node 5 to the destination node 10 is relative to the parent 2 that have the best $C^2 = 6 < 8$. As a rule, every offspring improves an objective C^k. For the creation of the offspring a partial path which improves the objective C^k is taken.

	Locus	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
parent1	chromosome	1	0	1	0	1	0	0	1	0	1
											(18,21)
	Locus	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
parent 2	chromosome	1	1	0	0	1	0	0	0	1	1
											(23,20)
	Locus	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
offsprings	chromosome	1	0	1	0	1	0	0	1	0	1
											(19,19)

Figure 18 – Example of crossover

2.5.4.4 Mutation

This phase generates a new random solution from each offspring. Then next step is to cross the solutions on the objectives optimized by the offspring. The objective here is to generate new genes of good quality in order to find the solutions in the Pareto optimal front. As a result, one of the two parents is replaced by the offspring.

2.5.5 Conclusion

This section provides an overview of some methods and techniques to solve the biobjective shortest path problem. All reviwed approaches to solve this problem classified as NP-complete [35] except the Scalarization Technique which is classified as polynomial complexity technique. As seen in this section, the Scalarization Technique can find only one nondominated solutions, while other techniques find many solutions.

3 FORMAL DEFINITION

The main problem tackled by this work is the SQPP problem. The work deals also with the QPP problem and the Safest Path Problem (SFPP), as the research looks to the SQPP problem as a variant of the two problems. This section provides formal definition for the optimal path (e.g. quickest, safest, easiest, etc...), then formally defines the SQPP problem.

3.1 Optimal Path Definition

The problem this work dealing with, is finding the optimal route between two locations in a road network; the optimal route can be defined in a number of ways: it can be the shortest route, the quickest route, the easiest route, or the route that would emit the least amount of air pollution. To formalize this problem, the road network is represented as a directed graph $G= (V, E)$ where V is a set of nodes and E is a set of edges between the nodes. A path P on G is defined as a connected sequence of edges. A path $P_{u,v}$ denotes a path that has a connected sequence of edges, the first edge starts from node u, and the last edge ends at node v. A weight function is used to define the cost of moving between two edges, it donated by $W_{u,v}$ .The weight of path $P_{v0,vk} =$

$(v_0, v_1, \ldots, v_k)$ is the sum of the weights of its constitute edges as shown in the following equation:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \quad (1)$$

The weight of an optimal solution to travel from u to v, denoted by $\delta(u, v)$, can be defined as the solution with the minimum weight, more specifically :

$$\delta(u, v) = min\{ w(p) \} \quad (2)$$

The weight function can be the length of the road, the required time to pass the road, the fuel consumption on the road, or any other metric.

3.2 Quick Path Problem

One way to approach the QPP problem is by applying a version of the weight function that returns the travel time on the edge (u, v), donated by: $tt_{u,v}$; the travel time can be calculated using the equation (3).

$$tt_{u,v} = \frac{l_{u,v}}{m_{u,v}} \quad (3)$$

Where $l_{u,v}$ is the length of the edge and $m_{u,v}$ is the maximum allowed speed on the edge. And the travel time of path $P_{v0,vk} = (v_0, v_1, \ldots, v_k)$ is the sum of the travel times of its constitue edges :

$$tt(p) = \sum_{i=1}^{k} tt(v_{i-1}, v_i) \quad (4)$$

By applying the definition of the optimal solution, the quickest path to pass the edge is given using equation (5)

$$P_{u,v}^{q} = \min\{\, tt(p) \,\} \quad (5)$$

In the real word, travel time varies over time (e.g. day and night, weekend, season), and the equation (3) will not give the accurate travel time. To overcome the limitation in calculating the real travel time, this work models road networks using time-dependent graphs. Following is the formal definition for time-dependent graph:

Definition 1: A time-dependent graph is given by a directed graph $G = (V, E, tt)$, that has an edge-delay function $tt_{u,v}(t)$ associated with each edge (u, v). The function gives the required traveling time between $u\ and\ v$ if one departs from u at time t.

Definition 2: An edge (u,v) has First-In First-Out (FIFO) property, if and only if $tt_{u,v}(t0) \leq tt(t0 + t\delta)$ for $t\delta \geq 0$, or $t1 + tt_{u,v}(t1) \leq t2 + tt_{u,v}(t2)$ for $t1 \leq t2$.

In other words, FIFO property of the edge (u, v) implies that if departing earlier from u, one arrives earlier at v. A Time-dependent network graph is a FIFO graph, if and only if all edges have FIFO property. TDSP problem in FIFO networks is polynomially solvable [43], while it is NP-hard in non-FIFO networks [44]. This concludes that finding the shortest and safest path in FIFO networks is polynomially solvable.

It is worthwhile to point out that a time-dependent graph constructed from a transportation network is a FIFO network [25] if the waiting on nodes is not allowed.

3.3 Safest Path Problem

SFPP is a variant of SPP problem, it can be solved using a weight function that considers the level of risk of the edge (u, v), the function is denoted by $r_{u,v}$, and the risk level of path $P_{v0,vk} = (\mathrm{v}_0, \mathrm{v}_1, \dots, \mathrm{v}_\mathrm{k})$ is the maximum level of risk through its constitue edges, that is :

$$r(p) = \max_{i \to 0,k} (r(v_{i-1}, v_i)) \tag{6}$$

By applying the definition of the optimal solution, the safest path to pass the edge is given using equation (7):

$$P^s_{u,v} = \min\{ r(p) \} \tag{7}$$

The work has extended the definition of time-dependent graph to introduce the risk-factor function $r_{u,v}$ which is the risk factor associated with the edge (u, v). The main difference between the travel time function and the risk function is that the former is subject to time and edge, while the other is subject to edge only. The used time-dependent graph is denoted by $G = (V, E, tt, r)$

3.4 The Safest and Quickest Path Problem

Each road segment is associated with two types of weights that are totally different, i.e. the travel time on the edge, and the risk factor on that edge. The travel-time of an edge corresponds to the required time to travel between the two intersections connected by that edge, while the risk factor on an edge is a measure of the safety level on the edge. At a high level, SQPP takes as input the time-dependent graph $G = (V, E, tt, r)$ together with a pair of source-destination nodes s, t. And the

objective is to provide the user with the safest and quickest path between denoted as $P_{s,t}^{sq}$.

The difficulty with this problem is that, the quickest path is not necessarily the one with the smallest risk and vice versa. Therefore, the problem is a bi-criteria optimization problem, where the objective is to minimize both the travel time as well as the risk. As mentioned before, there are multiple ways of formalizing, and subsequently, solving such a bi-criteria optimization problem. To solve the bi-criteria problem, this work used a hybrid technique that combines the two objectives, the travel-time and the safety level, into a single objective and ask for the path $P_{s,t}^{sq}$ minimizing the weighted combination as shown in equation (8) :

$$w(u,v) = \infty \times (tt_{u,v}(\mathrm{t})/\max_{E}(tt)) + \beta \times (r_{u,v}/5)$$
$$\infty + \beta = 1 \tag{8}$$

Where ∞ is the weighting factor for the travel-time, and β is the weighting factor for the road's safety level, with the constraint that $\infty + \beta = 1$. The two objectives are normalized by dividing the travel time by the maximum travel time between any two edges in E, and dividing the risk factor by the maximum risk factor in the road assessment method.

The weighting approach allows to give priority for one factor over the other, for example, when generating the shortest path between two locations, if $\beta = 0.7$ then the algorithm may give longer route, comparing to the one generated using a lower value for β, however, the former will have a better safety level.

It is clear that the final weight is impacted by the parameters ∞ and β, having an appropriate tuned values for those coefficients is crucial to find the optimal

solution for the problem, the following sections suggest the appropriate values for each objective :

Finding the quickest path: In this case, SFPP problem turned into quick path problem, and can be solved by assigning β with 0, and giving priority to the travel time by assigning ∞ with 1.

Finding the safest path: In this case, the SFPP problem turned into safe path problem, and can be solved by assigning ∞ with 0, and giving priority to the risk factor by assigning β with 1.

Finding the quickest and safest path: In this case, an input parameter should be used to specify the accepted risk level in the route (this parameter is referred to as ACCEPTED_RISK_FACTOR), the value assigned to each weighting factor will vary according to the priority of each objective. For example, Table 3 contains the list of paths to travel between two nodes. P1, P2 and P3 have different travel times and level of risk. If the quickest and safest path is requested, and ACCEPTED_RISK_FACTOR is assigned with 3, then the optimal route will be P2.

Table 3 - Routes with Different Risk Factors

path	travel time (s)	maximum risk in the path (1 to 5)
P1	100	2
P2	150	3
P3	200	1

4 RESEARCH METHODOLOGY

The objective of this project was to develop an algorithm that solve the SQPP problem by generating the quickest and safest route within an accepted period of time (online routing services such as Google and OSM provide the optimal route within 1-2 seconds, so the algorithm should not exceed this time threshold). The first step of this two-step project was to develop an algorithm to find the quickest path, and the second step was to modify the algorithm to consider safety when generating the optimal route.

In the first step, the work extended the Dijkstra and A* algorithms by designing QPA and QPHA algorithms to find the quickest routes between two places. QPA follows the Dijkstra approach in finding the optimal route; however, it uses travel time on the road as a weighting function. QPHA follows the heuristic approach defined in A* algorithm to find the optimal route and also uses travel time on the road as a weighting function. The heuristic function in QPHA estimates the travel time to pass the airline distance between a junction and the target node. The performance of the new algorithms was evaluated, and the winner algorithm was used to implement the SAA algorithm.

In the second step, the SAA algorithm was implemented, using a hybrid technique to combine the travel time and safety level to represent the weight of an edge. After that, an experiment was conducted to verify the ability of the new algorithm to output the optimal route.

4.1 Algorithms Performance Evaluation Metrics

The performance of QPA and QPHA for different trip lengths and road network sizes was evaluated and compared. In the first step, the algorithms were used to generate the quickest routes between multiple origin-destination pairs in different network sizes. In the next step, the simulation environment was used to verify the generated solutions, and simulation output results were used to evaluate the performance of each algorithm. The evaluation metrics included the computation time and the travel time. Computation time is an important metric, especially when routes are being recalculated while the car is moving to make sure that the current route is still the optimum one. An algorithm that needs a long time to calculate the best route may not be useful, as the car will have reached a new starting point by the time the results are generated. Travel time is the estimated time to reach the destination; the best route is calculated by considering the distance of the route and the real-time flow of traffic on that route.

4.2 Safety Level Definition

This work assumes that the safety levels of the roads are estimated using iRAP Star Ratings methods, where five-star roads are the safest while one-star roads are the least safe. The safety level on a road is reflected by the risk factor edge attribute. The value of this attribute ranges from one to five, with the safest edges having the minimum risk factor and unsafe edges having higher risk factors.

5 ALGORITHMS

This section explains in details Dijkstra and A* algorithms, then it explains the algorithms proposed by this work.

5.1 Dijkstra Algorithm

The classical Dijkstra algorithm, solves the problem of finding the shortest path between two nodes in a weighted directed graph in which the weights of all edges are nonnegative, hence, in the weighted graph $G = (V, E)$ and for each edge $(u, v) \in E$ the weight should be $w(u, v) \geq 0$.

Dijkstra algorithm divides graph's nodes into two distinct sets, labeled S and unlabeled Q, initially all nodes are in Q, e.g. they must be still evaluated. A node is moved to S if a shortest path from the source to this node has been found. Initially the distance of each node to the source is set to ∞ as an indication that there is no direct path between the two nodes. The original algorithm runs until Q becomes empty, however, it can be terminated once the target node has been found. In each iteration it selects the node with the lowest distance from the source out the unlabeled nodes, it reads the edges and find the destination nodes of those edges, for the unlabeled destination points, the algorithm checks whether the known distance from the source to the destination point can be reduced if the selected edge is used, if this can be done then the distance is updated and the node is added to Q.

For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a shortest-path

estimate. We initialize the shortest-path estimates and predecessors by the following $\Theta(V)$-time procedure.

The time complexity of this algorithm depends on the implementation of the queue *Q*, when implemented using an unordered array, then the time complexity will be $((V^2)$, when a Fibonacci heap is used to implement the queue then the complexity will be ((V lg V + E).

Algorithm 3 - Dijkstra

```
1.  for each vertex v∈ V[G] do
2.    d[v] ← ∞
3.    π[v] ← NIL
4.  d[s] ←0
5.  S ← Ø
6.  Q ← V[G]
7.  while Q ≠ Ø do
8.    u ← EXTRACT-MIN(Q)
9.    S ← S∪{u}
10.   for each vertex v∈Adj[u] do
11.     if d[v] > d[u] + w(u, v) then
12.       d[v] ← d[u] + w(u, v)
13.       π [v] ← u
```

5.2 A* Algorithm

It is an extension of Dijkstra algorithm, A* achieves better performance (with respect to time) by using heuristics. A* follow the best-first search technique used to find the shortest path, it evaluates the cost to reach a destination node n using the function $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost from the start node to the node *n*, and $h(n)$ is a heuristic function which estimates the cost to get from the node n to the goal [45]. A typical heuristic evaluation function is the air-line distance from the start node to the goal node.

Like Dijkstra algorithm, A* divides graph's nodes into two distinct sets. Labeled *S* and unlabeled *Q*, initially the start node *s* is in *Q*. A node is moved to S if a shortest path from the source to this node has been found. The algorithm runs until *Q* becomes empty or the target node has been found. In each iteration it selects the nearest node to the destination, it reads the edges and find the destination nodes of those edges, for the unlabeled destination points, the algorithm checks whether the known distance from the source to the destination point can be reduced if the selected edge is used, if this can be done then the distance is updated, the distance from this destination point to the target point v is estimated and the node is added to *Q*. If this destination point was not checked before, then the distance from the source node to this destination node is calculated, the distance from this destination point to the target point v is estimated and the node is added to *Q*.

Like Dijkstra, an attribute $d[v]$ is maintained, which is an upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a shortest-path estimate. Another attribute is maintained which is $h[v]$, it is the estimated distance from the node v to the target point, the distance is calculated using the heuristic function $h()$.

A* always finds an optimal solution to a problem if the heuristic function is admissible, i.e. never overestimates the actual cost of solution.

In the context of graph search, the complexity of A* is $O(b^d)$, where b is the average number of successors (children) per node, and d is the depth of the generated path.

Algorithm 4 - A*

```
1.  S ← Ø
2.  Q ← s
3.  h ← Ø
4.  d[s] ← 0
5.
6.  while Q ≠ Ø do
7.    u ← EXTRACT-MIN(Q)
8.    If (u is the target)
9.      stop;
10.   S ← S∪{u}
11.   for each vertex v∈Adj[u] do
12.    if v ∈ Q then
13.      if d[v] > d[u] + w(u, v) then
14.        d[v] ← d[u] + w(u, v)
15.        h[v] ← h(v)
16.        π [v] ← u
17.    else
18.      d[v] ← d[u] + w(u, v)
19.      h[v] ← h(v)
20.      π [v] ← u
21.   Q ← v
```

5.2.1 Critical Analysis for the Heuristic Function

In A* algorithm the heuristic function controls the behavior of the function, the following cases explains the behavior of the algorithm for different values of the heuristic function [46] :

- If $h(n)$ is 0, then A* turns into Dijkstra, as only $g(n)$ has the role, which is guaranteed to find the shortest path.
- If $h(n)$ is lower than (or equal) to the cost of traveling from the node n to the goal, then A* is guaranteed to find the shortest path. The lower $h(n)$ is, the more node A* expands, and consequently making it slower.

- If $h(n)$ is exactly equal to the cost of traveling from n to the goal (which can't be happened), then A* will only follow the best path and will never expand other nodes, hence, it will be very fast.
- If $h(n)$ is greater than the real cost of traveling from n to the goal, then A* is not guaranteed to find the shortest path, but it can run faster.

5.3 Quickest Path Algorithm

This algorithm is a variant of Dijkstra algorithm; this algorithm provides the quickest path from a source junction s to a taget junction, the algorithm stops when it finds the target junction, while standard Dijkstra finds the shortest path between the source edge and all other edges in the graph. The weight function $w(u, v)$ in Dijkstra calculates the distance between the edges $, v$, while in the new algorithm, the weight of an edge is given using the function $tt_{u,v}(t)$ which gives the required time to travel from u to v at time t. More specific, if u is arrived at time $t1$, then the v is arrived at time $t1 + tt_{u,v}(t1)$. As shown in the pseudo code of this algorithm, the array $t[]$ is used to store the travel time between the source edge s and other edges in the graph. Also, the function *EXTRACT-MIN* gives the edge with the smallest travel time from the source edge, while, in Dijkstra, the function returns the nearest edge from the source. The unlabeled set Q is implemented as Fibonacci Heap which helps to achieve a complexity of $O(1)$ in the function *EXTRACT-MIN*.

The time complexity of QPA is ((V lg V + E) as long as the time complexity of $tt_{u,v}(t)$ is in O(1) .

Algorithm 5 – QPA

1. **for** *each vertex* $v \in V[G]$ **do**
2. $t[v] \leftarrow \infty$
3. $\pi[v] \leftarrow NIL$
4. $t[s] \leftarrow 0$
5. $S \leftarrow \emptyset$
6. $Q \leftarrow V[G]$
7. **while** $Q \neq \emptyset$ **do**
8. $u \leftarrow EXTRACT\text{-}MIN(Q)$
9. **If** *(u is the target)*
10. *stop;*
11. $S \leftarrow S \cup \{u\}$
12. **for** *each vertex* $v \in Adj[u]$ **do**
13. **if** $t[v] > t[u] + tt_{u,v}(t[u])$ **then**
14. $t[v] \leftarrow t[u] + tt_{u,v}(t[u])$
15. $\pi[v] \leftarrow u$

5.4 Quickest Path Algorithm – Heuristic

This algorithm is a variant of A* algorithm; this algorithm provides the quickest path from a source junction s to a taget junction. The weight function $w(u,v)$ in A* calculates the distance between the edges u, v , while in the new algorithm, the weight of an edge is given using the function $tt_{u,v}(t)$ which gives the required time to travel from u to v at time t. More specific, if u is arrived at time $t1$, then the v is arrived at time $t1 + tt_{u,v}(t1)$. As shown in the pseudo code of this algorithm, the array $t[]$ is used to store the travel time between the source edge s and other edges in the graph, while the array $h[]$ is used to store the value of the heuristic function $h(v)$. Also, the function *EXTRACT-MIN* gives the edge with the smallest travel time to the target edge, while, in A*, the function returns the nearest edge from

the target. The unlabeled set Q is implemented as Fibonacci Heap which helps to achieve a complexity of $O(1)$ in the function *EXTRACT-MIN*. As mentioned in 5.2, the heuristic $h(v)$ function controls the behavior of the algorithm, in QPHA the function estimates the required time to pass the Manhattan distance between the current edge and the target edge, also it take into consideration the required time to reach the current edge, equation (9) show the definition of this function, where $t[u]$ is the required time to reach the edge u, $md(u,v,t[u])$ calculates the Manhattan distance between u and the target node v at time $t[u]$ in meters, and $S(t[u])$ is the speed estimation along the Manhattan distance at the time $t[u]$ in meter per second (m/s). Also, Figure 19 shows how the Manhattan distance is calculated for an edge when the optimal path between two locations in Doha, that are Landmark Mall and Al Markiya Street, is generated by the algorithm.

$$h(n) = t[u] \; + \; (md(u,v,t[u]) \; / \; S(t[u]) \tag{9}$$

Figure 19 – Manhattan Distance

The function $tt_{u,v}(t)$ should returns a precalculated value in a O(1) time complexity which can be achieved by storing the travel time of each edge in a hash table.

The time complexity of QPHA is $O(b^d)$, as long as the time complexity of both $tt_{u,v}$ (t) and $h(v)$ is in O(1), where b is the average number of successors (children) per edge, and d is the depth of the generated path.

Algorithm 6 - QPHA

```
1.  S ← Ø
2.  Q ← s
3.  h ← Ø
4.  s[s] ← 0
5.
6.  while Q ≠ Ø do
7.    u ← EXTRACT-MIN(Q)
16.  If (u is the target)
8.     stop;
9.    S ← S∪{u}
10.  for each vertex v∈Adj[u] do
11.    if v ∈ Q then
12.      if t[v] > t[u] + tt_{u,v} (t[u])then
13.        t[v] ← t[u] + tt_{u,v} (t[u])
14.        h[v] ← h(v)
15.        π [v] ← u
16.     else
17.       Q ← u
18.
19. function h(n){
20.  return t[u] + (md(u, v, t[u]) / S(t[u])
21. }
```

5.5 Safety Aware Algorithm

The SAA algorithm is variant of QPHA, it uses different method in calculating the weight of an edge. QPHA uses the travel time on the edge, given in the equation (10), as a weight function, while SAA combines both the travel time on the edge and the risk factor of the edge in one value, as shown in equation (11)

$$w(u,v) = tt_{u,v}\ (t[u]) \tag{10}$$

$$w(u,v) = \infty \times (tt_{u,v}(\mathrm{tu}) / \max_{E}(tt)) + \beta \times (r_{u,v}/5) \tag{11}$$

The function $\max_{\mathrm{E}}(\mathrm{tt})$ gives the maximum travel time between two edges in the roadnetwork, this function should returns a precalculated value, and not calculate the maximum travel time in the network as part of this algorithm, hence, the time complexity of this function is O(1). Also, the time complexity of equation (11) is O(1), as a result, SAA has the same time complexity of QPHA.

Algorithm 7 - SAA

```
1.  S ← Ø
2.  Q ← s
3.  h ← Ø
4.  s[s] ← 0
5.
6.  while Q ≠ Ø do
7.    u ← EXTRACT-MIN(Q)
17.  If (u is the target)
8.     stop;
9.   S ← S∪{u}
10.  for each vertex v∈Adj[u] do
11.    if v ∈ Q then
12.      if t[v] > t[u] + tt_{u,v} (t[u])then
13.        t[v] ← t[u] + tt_{u,v} (t[u])
14.        h[v] ← h(v)
15.        π [v] ← u
16.    else
17.       Q ← u
18.
19. function h(n){
20.  return t[u] + (md(u, v, t[u]) / S(t[u]))/max_E(tt)
21. }
```

6 TRAFFIC SIMULATION

6.1 Overview

The complexity of traffic stream behavior and the difficulties in performing experiments on real transportation networks makes computer simulation an important analysis tool for evaluating and testing traffic solution before they are implemented in practice. The physical propagation of traffic flows can be specifically described using traffic flow models. These models allow to simulate large scale real-world situations in great detail. Depending on the level of detailing, traffic flow models are classified into microscopic, mesoscopic and macroscopic models. Microscopic models give attention to individual vehicles and their interactions, a microscopic model of traffic flow attempts to analyze the flow of traffic by modelling driver-driver and driver-road interactions within a traffic stream. Macroscopic models view the traffic flow as a whole, while the microscopic models fall in between these two. This work uses SUMO, which is a microscopic simulation framework, to verify the validity of the generated route in terms of following traffic directions and reaching the desired destination, and also to obtain the required times for traveling between locations.

6.2 SUMO

SUMO, which stands for 'Simulation of urban mobility', is a microscopic simulator, and it is an open source program. SUMO provides a lot of functionality regarding importing and altering networks from different sources, creating vehicles and defining routes for them. It also has an implementation of the well know shortest path algorithms Dijkstra and A* that can be used to find the shortest path in the defined networks. Of course SUMO is not the only open source traffic simulator, it

has been used in this work because it is free, allows importing maps from different sources, it has a remote interface (TraCI) to communicate with the simulation through Java, and Java is the language used for implementing the algorithms and the experiments in this research.

6.3 When to use SUMO?

To understand SUMO in practice, the literature was consulted to understand how to use SUMO and in which contexts, this section gives an overview about some use cases that made use of SUMO either as a part of a final solution, or as testbed.

In [47] a dynamic route planning framework is suggested, the routing algorithm can interact with the dynamic changing environment, like road networks, and adapts the route assigned to a vehicle according to the new updates (i.e. change in congestions level or incidents) received from the traffic management systems. The process of updating the best route is triggered from time to time. The algorithm was tested by defining a network using SUMO, also Dijkstra algorithm was implemented in a way that allows the algorithm to access the road network to generate routes. A framework to simulate vehicles traveling on the road network was developed. The framework allows to define vehicles with initial routes, during the simulation, and according to the updates received from a traffic management systems (i.e. change in congestions level, incidents etc.), the algorithm adapts the best routes assigned to a vehicle according to the new updates. The work uses TraCI interface that provides access to a running road traffic simulation, and it allows to retrieve values of simulated vehicles and to manipulate their route "on-line".

In [48] a hardware-in-the-loop simulation platform for emulating large-scale intelligent transportation systems is proposed. The platform allows building large-scale traffic simulations where the real cars on the roads can be part of that simulation; in this platform, the data is collected from the vehicle's gateway and sent to the server using a smartphone. The information collected from the vehicles on the road is processed on the server and behavioral recommendations are sent back to the vehicles (like speeds and reroutes).

6.4 Simulation Results in SUMO

A traffic simulation scenario in SUMO is performed using a file that contains all vehicles with their routes/trips and a road network defined in a separate file. By running the simulation, the vehicle will depart from their starting point toward the destination point, the required time to complete the trip is affected by many factors like the traffic in the road, facing simulated car accident, the lights on the road, etc. By the end of the simulation, SUMO creates an output file which contains information like arrival time, arrival position, trip duration, waiting steps, the time loss (due to traffic jam, waiting on signals), etc.

6.5 SUMO Road Network Definition

6.5.1 Network Format

A SUMO network file describes the traffic-related part of a map, the roads and intersections the simulated vehicles run along or across. In SUMO a road network is represented as a directed graph, ”nodes” usually named "junctions" and they represent intersections, and "edges" represent roads. Specifically, the SUMO network contains the following information:

- Coordinates and alignment of the original map
- Every street (edge) as a collection of lanes, including the position, shape and speed limit of every lane
- Traffic light logics referenced by junctions
- Junctions, including their right of way regulation
- Connections between lanes at junctions (nodes)

Figure 20 shows a road network defined using SUMO, the figure shows junctions highlighted in green, and the street crossing them. Also, the figure shows a roundabout.

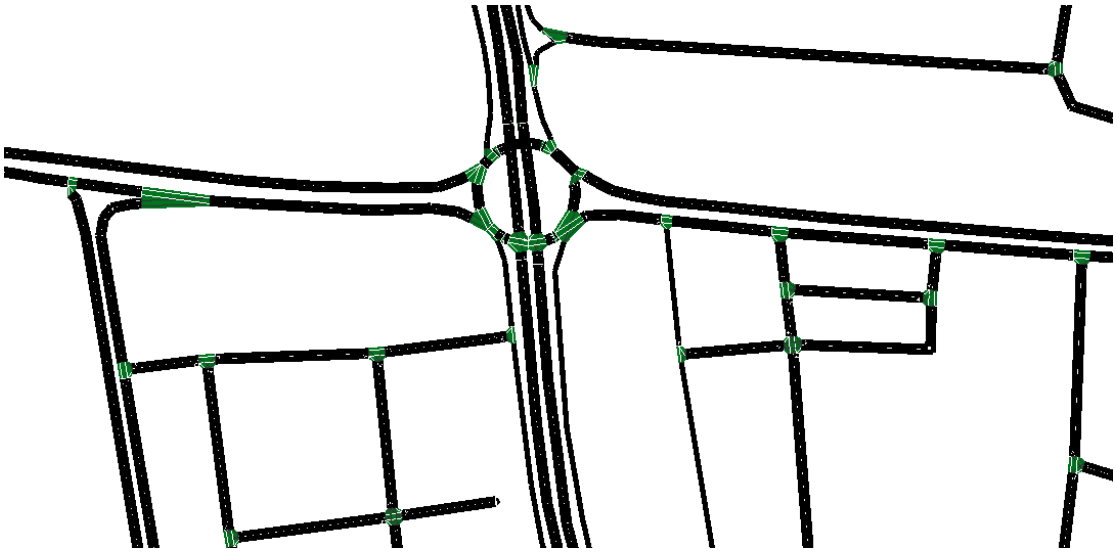


Figure 20 – Road Network in SUMO

6.5.2 Coordinates and Alignment

In SUMO, the networks use Cartesian or metric coordinates where the leftmost node is at x=0 and the node being most at the bottom is at y=0. This means that when real road network are imported, e.g. from OSM, then NETCONVERT [49] is

projecting the network, first, if the original network was not using Cartesian and/or metric coordinates. Then, they move the road network to the origin at (0,0).

6.5.3 Edges and Lanes

A normal edge is a connection between two nodes "junctions", and each edge includes the definitions of lanes it consists of.

6.5.4 Traffic Light Program

NETCONVERT [49] generates traffic lights and programs for junctions during the computation of the networks. These computed programs are different from the real ones, however, it is allowed to provide the simulation with real traffic light programs.

6.5.5 Junctions

Junctions represent the area where different roads cross, including the right-of-way rules vehicles have to follow when crossing the intersection.

6.5.6 Plain Connections

Plain connections describe how streets crossed in an intersection are connected together. By linking together the incoming lanes and the outcoming lanes in the two streets.

6.6 Vehicle Interaction

The relation between the simulated vehicles is defined using a car-following model and a lane changing model [50]. The car-following model specifies how a car adapts its speed according to the distance to the car in front of it.The lane change model defines when a vehicle should change lanes, this happens whenever the current lane of the vehicle has no connection to the next edge of the route. SUMO is designed

to be collision free, which means that any vehicle is able to avoid a collision with the cars in front of it.

6.7 Vehicle Configuration Parameters

Vehicles defined in SUMO have many configuration parameters; following is a short overview of important parameters:

- Id: the name of the vehicle.
- Route: the id of the assigned route. Routes are defined in a separate file; each route is a set of connected edges.
- Depart: the time step at which the vehicle should enter the network
- DepartLane: the lane on which the vehicle should enter the network
- DepartSpeed: the speed with which the vehicle should enter the network
- ArrivalSpeed: the speed with which the vehicle should leave the network
- Type: the type defines both, vehicle's physical parameters (like length, color), and also the used car-following model's parameters.

7 EXPERIMENTS

This chapter presents an experimental assessment of the algorithms QPA, QPHA and SAA. The objectives of the experiments were: (1) Evaluate the performance of QPA and QPHA in generating the quickest route between two locations using different performance metrics, thereby indicating whether SAA should use the Dijkstra approach or A* approach for generating the quickest and safest route. (2) Verify the ability of the SAA algorithm to select a safe route without sacrificing travel time by testing SAA with different combinations of β and ∞ . It was expected that the higher the value of β, the longer the generated route would be. We began by developing a method to integrate any road network with the algorithm through the following steps: (1) obtaining a real road network from OSM as an xml file, (2) parsing the xml file to generate a digital map in SUMO format and (3) parsing and loading the digital map as a Java object using a third-party library.

7.1 Obtaining Road Networks

SUMO provides several ways to obtain road networks: simple networks can be defined by hand, and real world maps can be imported from different sources such as OpenStreetMap, VISUM, VISSIM and Navteq. In this work, the map of Qatar was imported from OSM [44]. OSM is a free editable map of the world that is being built by imports from different sources [45] and has been released with an open-content license. A disadvantage of OSM is that the data can be inaccurate, because the map can be updated by anyone. For this project, we verified some areas in Doha using Google Maps and found the OSM to be highly accurate. OSM maps contain the

required data for this work, including speed limits, the number of lanes in each road, traffic lights, junctions and turn restrictions.

This work used fours maps to evaluate the perofrmacne of QPA, QPHA algorithms, the maps are: a central area in Doha, a remote area in Doha, Doha city map and Glasgow city map (Glasgow, UK).

7.2 Parsing and Loading Road Networks

In this step, the aim was to make a SUMO road network accessible to the algorithms by representing them as vertices and edges. To achieve this, a third-party library was used [46] to parse the road network file generated through SUMO. The road network was then stored in a data structure representing the vertices, edges, lanes, maximum speed on the road, turn directions, safety level, and other factors.

7.3 Algorithms Performantce Evaluation

The performance of QPA and QPHA was evaluated using metrics such as scalability and computation time. The algorithms were used to generate the quickest routes between multiple origin-destination pairs in different network sizes. In the next step, the generated routes were verified using SUMO, and simulation results were used to evaluate the performance of the algorithm. The following sections explain the maps used and routes in more detail.

7.4 Maps with Different Scalability Levels

The work uses some maps with different sizes, that are : a central area in Doha, a remote area in Doha, Doha city map and Glasgow city map. Figure 21, Figure 22,

Figure 23 and Figure 24 respectively show those maps, and Table 4 shows the number of junctions and roads in each map

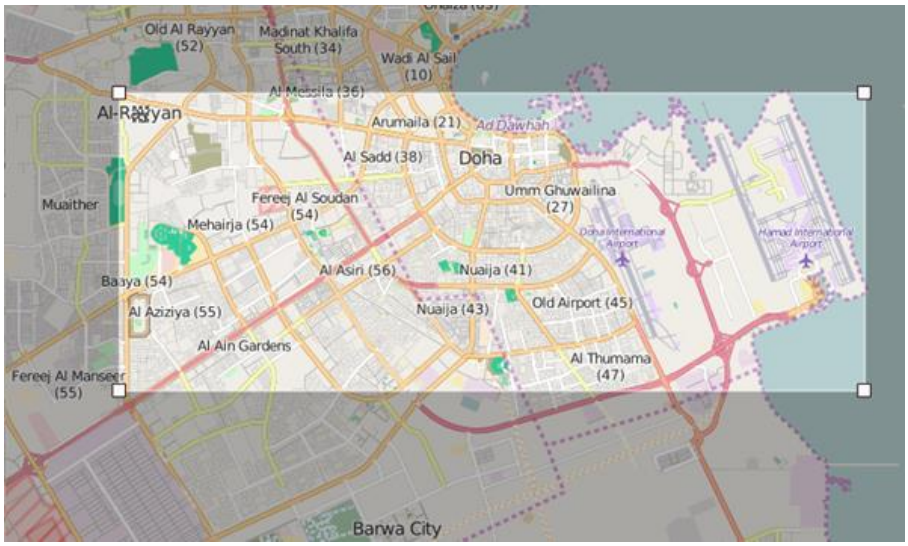


Figure 21 – Doha - Center Area

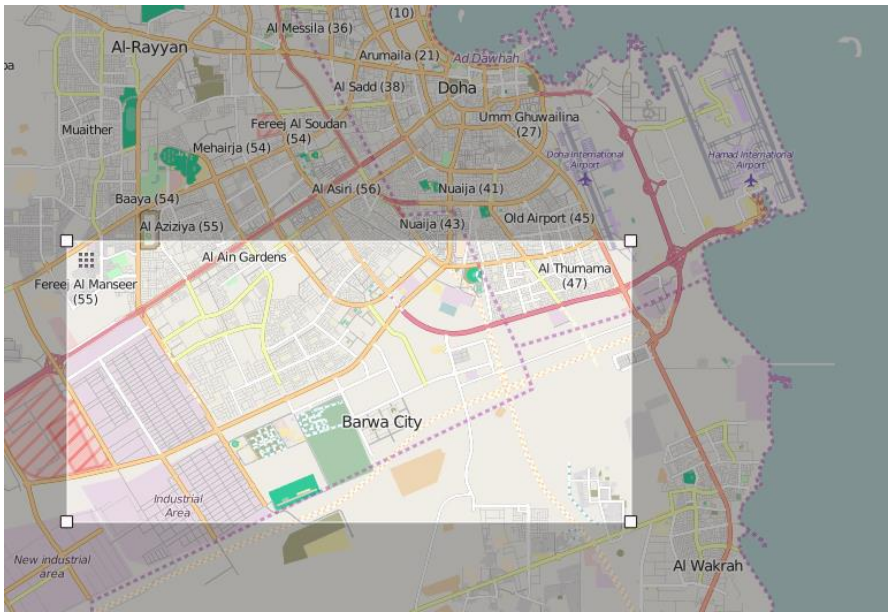


Figure 22 – Doha - Remote Area

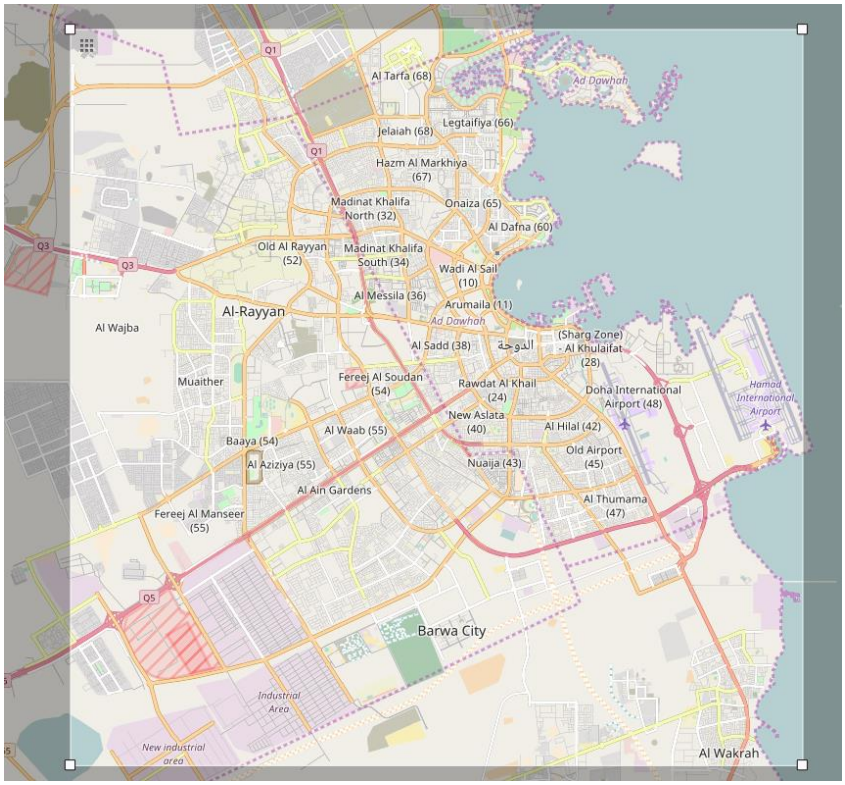


Figure 23 – Doha Map

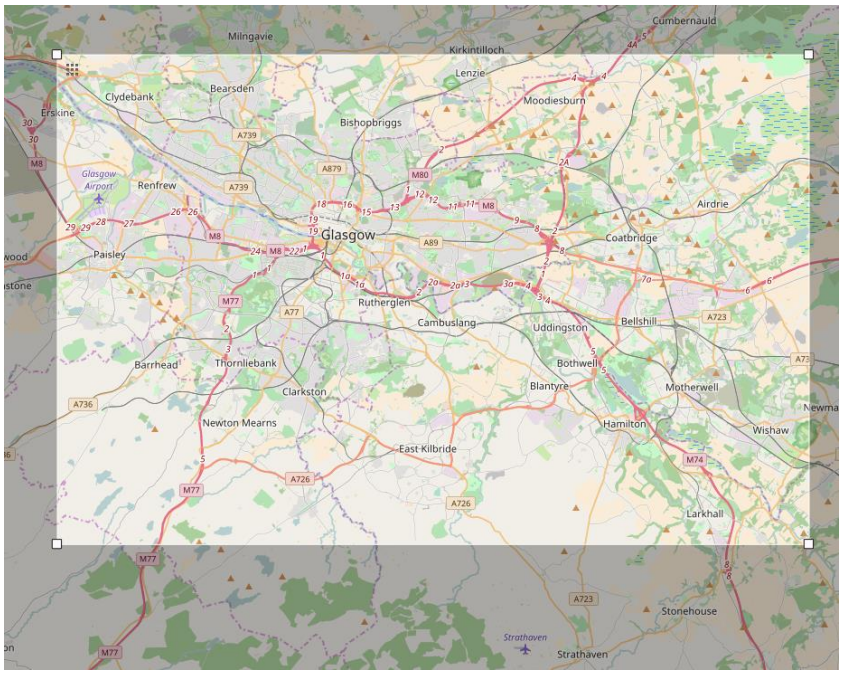


Figure 24 – Glasgow Map

Table 4 - Number of junctions and roads in the maps

Map	Number of junctions	Number of roads
Doha - central area	11615	26502
Doha - remote area	3693	8869
Doha city	35669	82418
Glasgow city	122491	261700

7.4.1 Road Length

To investigate the impact of trip length on the algorithms' performance, the quickest route for four trips of different lengths were generated, with length defined as the airline distance between the start location and the destination. In the case of the central and remote areas in Doha , the lengths of the trips were 4 km, 6 km, 8 km and 10 km, and the trips were defined in the central area and the remote area. Figure 25 and Figure 26 respectively show the trips in the two areas.

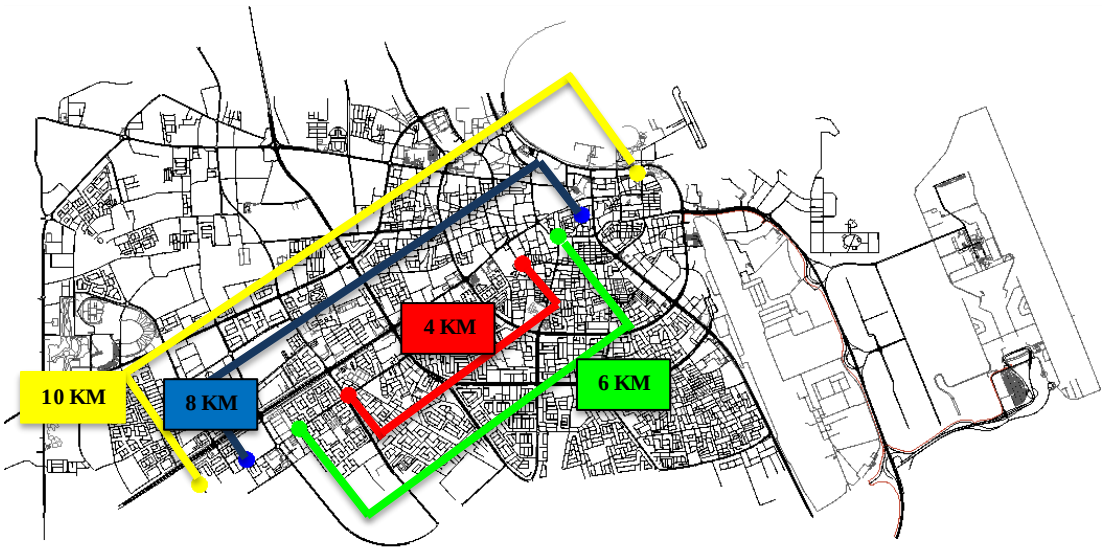


Figure 25 – Four Routes in Central Area

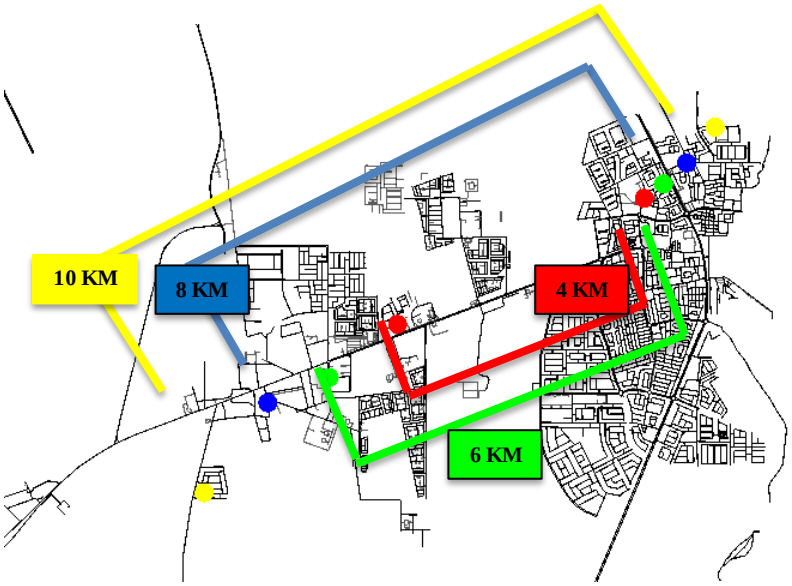


Figure 26 – Four Routes in Remote Area

In the case of Doha cotu and Glasgow city, the length of the routes are : 10 km, 12 km, 14 km and 16 km.

7.4.2 Performance Analysis

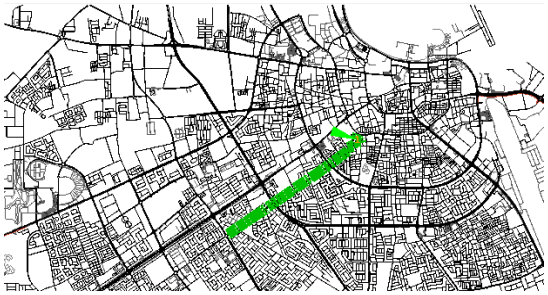
7.4.2.1 Generated Routes Validity

7.4.2.1.1 Central Area

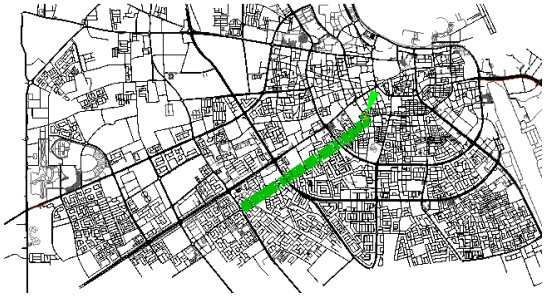
In the central area, both QPA and QPHA generated the same routes for the trips of length 4 km, 6 km, 8 km and 10 km. Figure 27 shows the generated routes. The generated routes were verified using SUMO, and the outputs of this simulation, including trip duration, are shown in Table 5 (s=second, m = meter).

Table 5 - Trips Duration and Length in Central Area

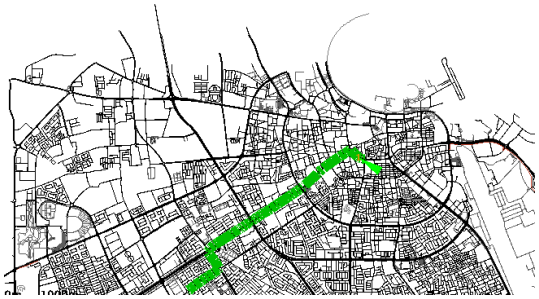
	4 KM		**6 KM**		**8 KM**		**10 KM**	
Algorithm	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)
QPA	276	4524	375	7014	522	8610	701	11691
QPHA	276	4524	375	7014	522	8610	701	11691



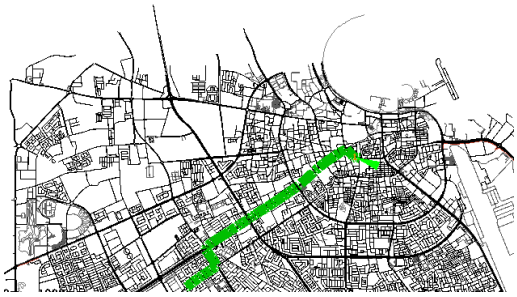
(a) Quickest Path using QPHA - 4KM



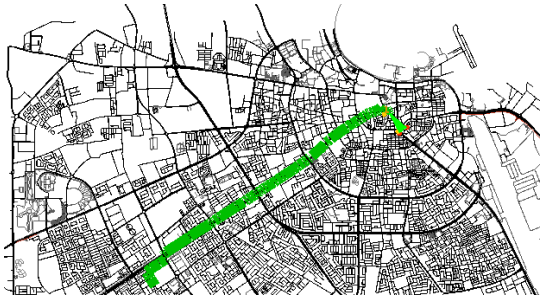
(b) Quickest Path using QPA - 4KM



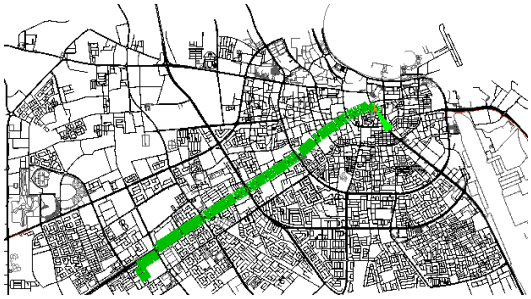
(c) Quickest Path using QPHA - 6KM



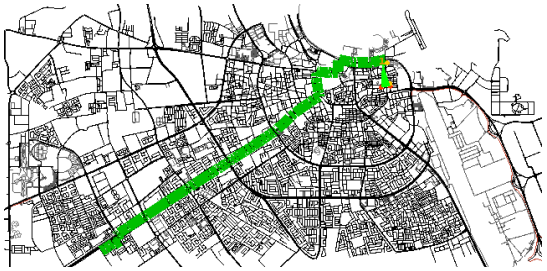
(d) Quickest Path using QPA - 6KM



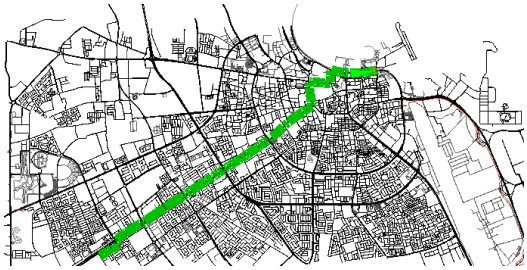
(e) Quickest Path using QPHA - 8KM



(f) Quickest Path using QPA - 8KM



(g) Quickest Path using QPHA - 10KM



(h) Quickest Path using QPA - 10KM

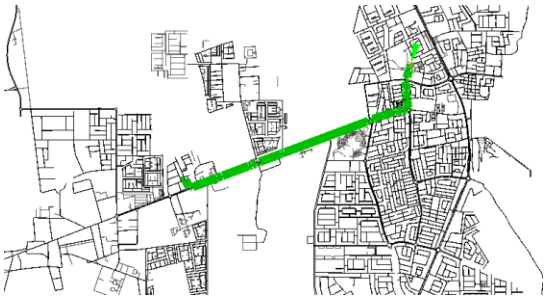
Figure 27 – The Generted Routes in Central Area

7.4.2.1.2 REMOTE AREA

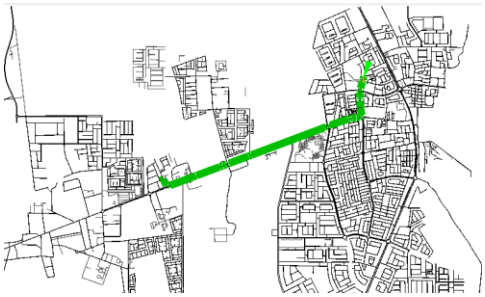
In the remote area, both QPA and QPHA generated the same routes for the trips of length 4 km, 6 km, 8 km and 10 km. Figure 27 shows the generated routes. The generated routes were verified using SUMO, and the outputs of this simulation, including trip duration, are shown in Table 6 (s=second, m=meter).

Table 6- Trips Duration and Length in Remote Area

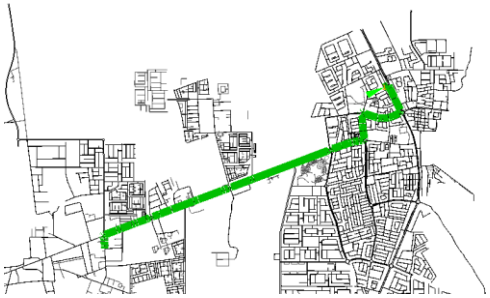
	4 KM		6 KM		8 KM		10 KM	
Algorithm	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)
QPA	276	4524	375	7014	522	8610	701	11691
QPHA	276	4524	375	7014	522	8610	701	11691



a) Quickest Path using QPHA - 4KM



b) Quickest Path using QPA - 4KM



c) Quickest Path using QPHA - 6KM

d) Quickest Path using QPA - 6KM

e) Quickest Path using QPHA - 8KM

f) Quickest Path using QPA - 8KM

g) Quickest Path using QPHA - 10KM

h) Quickest Path using QPA - 10KM

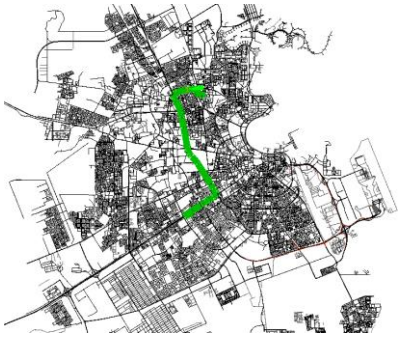
Figure 28 – The Generted Routes in Remote Area

7.4.2.1.3 DOHA

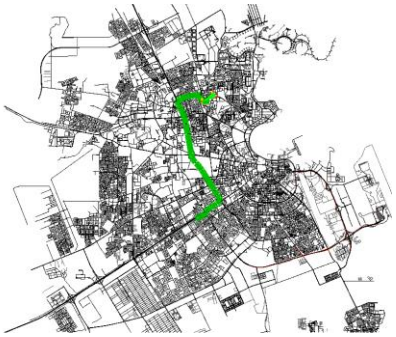
In the map of Doha city, both QPA and QPHA generated the same routes for the trips of length 10 km, 12 km, 14 km and 16 km. Figure 29 shows the generated routes. The generated routes were verified using SUMO, and the output result of this simulation, including trip duration, are shown in Table 7 (s=second, m=meter).

Table 7 - Trips Duration and Length in Doha

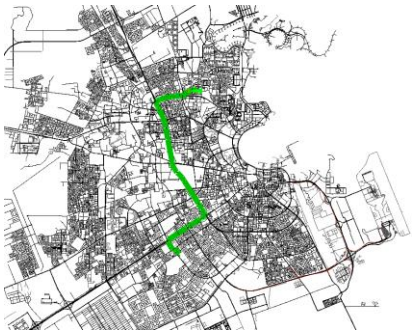
	9 KM		**11 KM**		**14 KM**		**16 KM**	
Algorithm	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)
QPA	680	14097	873	20377	968	21114	1204	25250
QPHA	680	14097	873	20377	968	21114	1204	25250



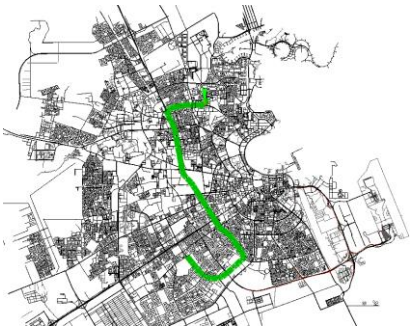
a) Quickest Path using QPHA - 10KM



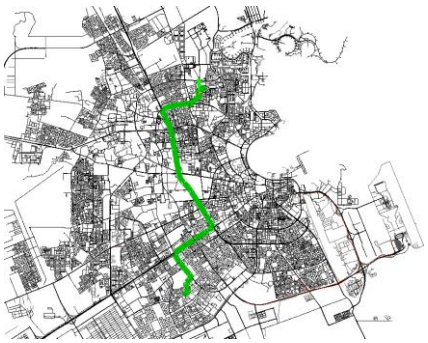
b) Quickest Path using QPA - 10KM



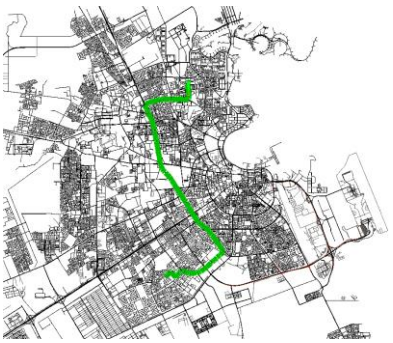
c) Quickest Path using QPHA - 12KM



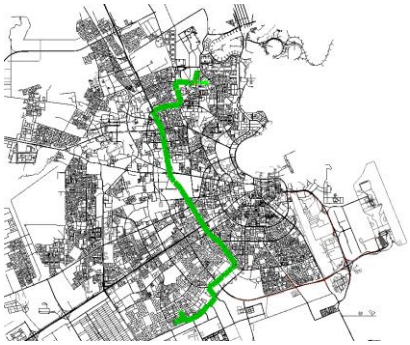
d) Quickest Path using QPA - 12KM



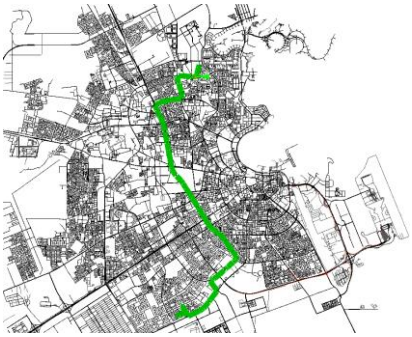
e) Quickest Path using QPHA - 14KM



f) Quickest Path using QPA - 14KM



g) Quickest Path using QPHA - 16KM



h) Quickest Path using QPA - 16KM

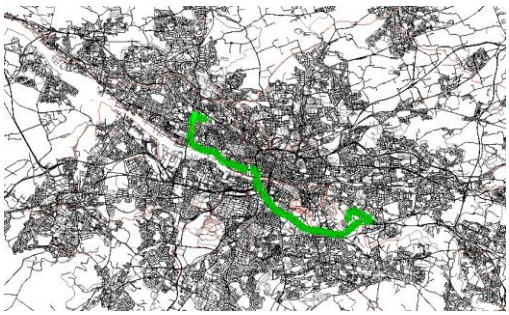
Figure 29 – The Generated Routes in Doha Map

7.4.2.1.4 GLASGOW

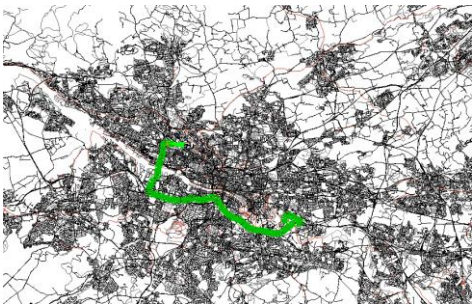
In the map of Glasgow city, both QPA and QPHA generated the same routes for the trips of length 12 km, 16 km, however, the generated routes were different for the trips of length 14 km and 16 km. Figure 30 shows the generated routes. The generated routes were verified using SUMO, and the output result of this simulation , including trip duration, are shown in Table 8 (s=second, m=meter).

Table 8 - Trips Duration and Length in Glasgow

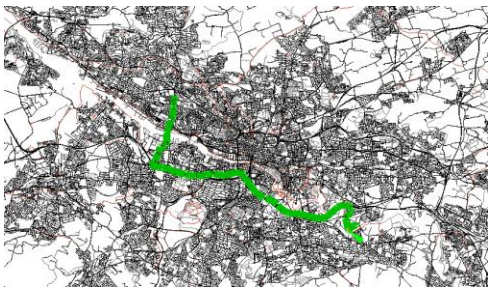
	9 KM		**11 KM**		**14 KM**		**16 KM**	
Algorithm	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)	Duration (s)	Length (m)
QPA	1120	18563	1079	18558	1327	21346	1701	23654
QPHA	1029	17180	1079	18558	1237	19964	1701	23654



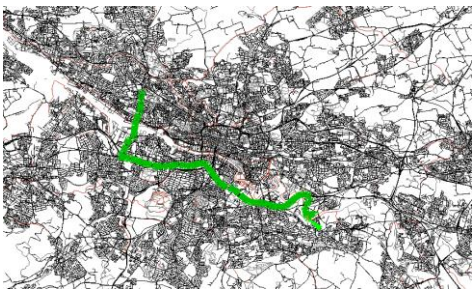
a) Quickest Path using QPHA - 10KM



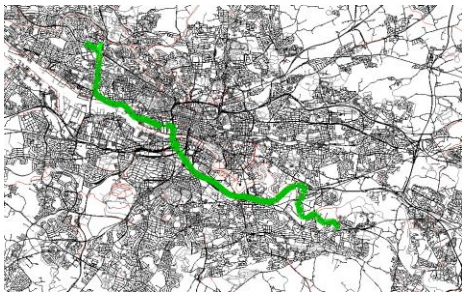
b) Quickest Path using QPA - 10KM



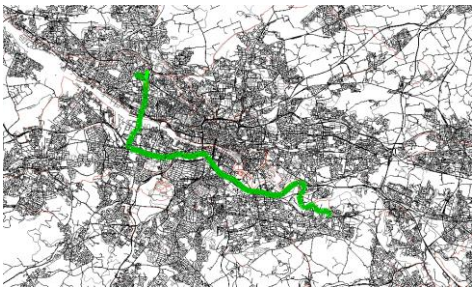
c) Quickest Path using QPHA - 12KM



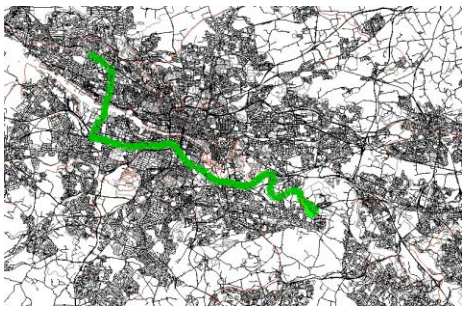
d) Quickest Path using QPA - 12KM



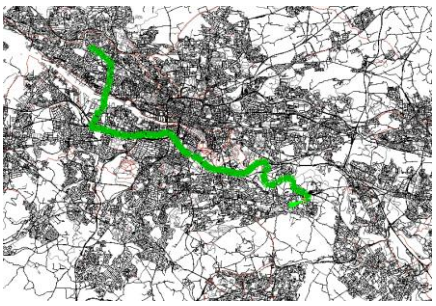
e) Quickest Path using QPHA - 14KM



f) Quickest Path using QPA - 14KM



g) Quickest Path using QPHA - 16KM



h) Quickest Path using QPA - 16KM

Figure 30 - The Generated Routes in Glasgow Map

7.4.2.2 Computation Time

This section describes the required time for QPA and QPHA to generate the quickest route. As mentioned in section 5.2.1, the heuristic function has a major impact on both the computation time and the result of the algorithm. Hence, this experiment used three versions of the QPHA algorithm, with each version using a different equation to estimate the required time to reach the destination and the three equations different in their assumed travel speeds. Table 10 shows the used equations (m/s = meter per second, k/h=kilometer per hour)

Table 9 - QPHA Versions

Algorithm Version	Heuristic Function's Equation
QPHA_V1	$h(n) = t[u] + (md(u, v, t[u]) / 5.55556)$ 5.55556 m/s ~ 20 k/h
QPHA_V2	$h(n) = t[u] + (md(u, v, t[u]) / 2.77778)$ 2.77778 m/s ~ 10 k/h
QPHA_V3	$h(n) = t[u] + (md(u, v, t[u]) / 1.38889)$ 1.38889 m/s ~ 5 k/h

Figure 31, Figure 32, Figure 33 and Figure 34 show the required times to generate the routes in the central area and the remote area, respectively, using QPA and the three versions of QPHA.

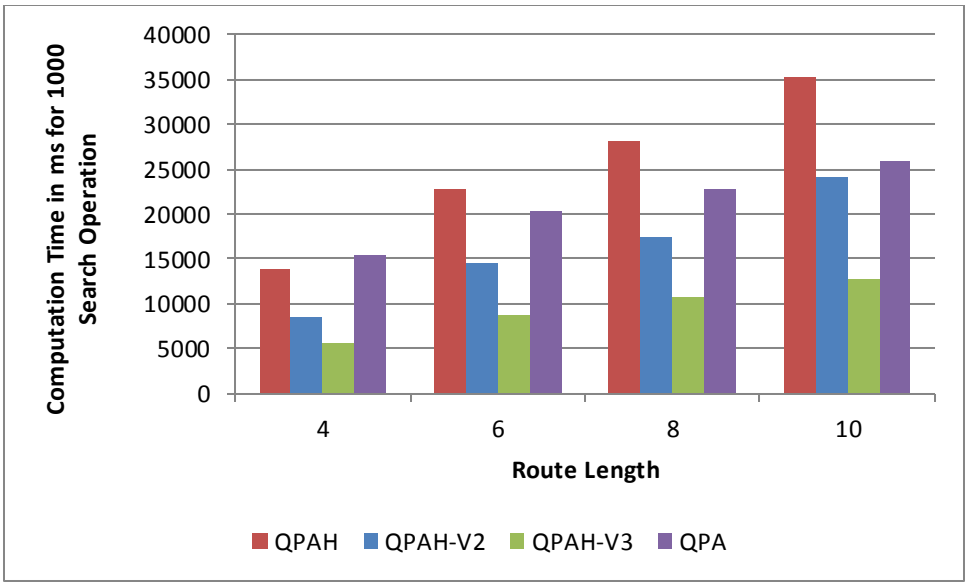


Figure 31 – Algorithms Computation Time in Central Area

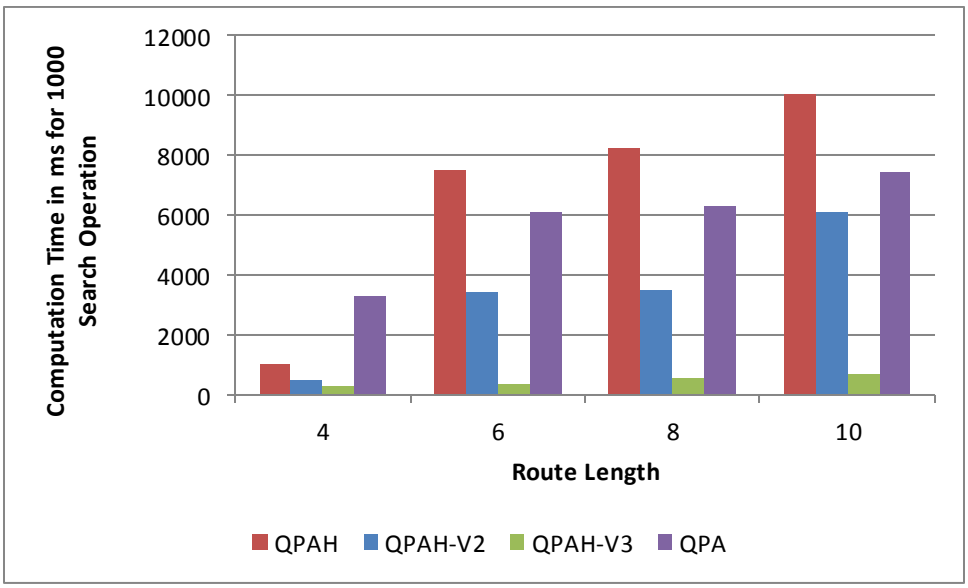


Figure 32 - Algorithms Computation Time in Remote Area

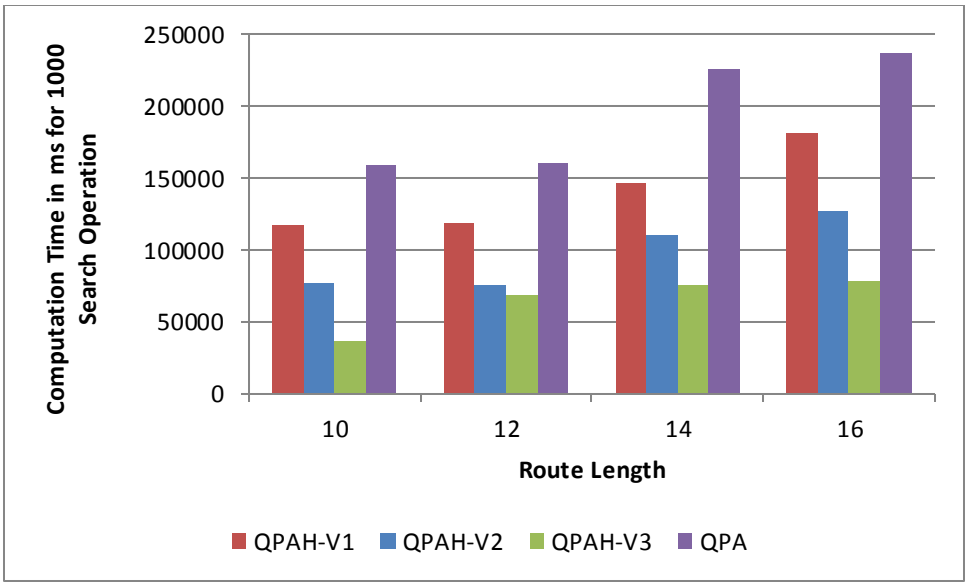


Figure 33 - Algorithms Computation Time in Doha

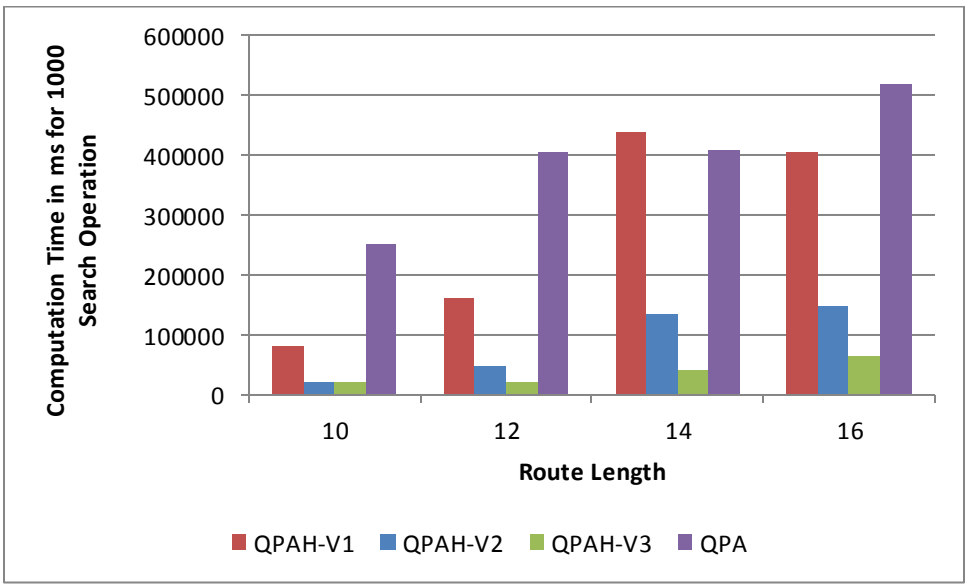


Figure 34 - Algorithms Computation Time in Glasgow

7.4.2.3 Results Anlysis

Figure 31 and Figure 32 illustrate the impact of the heuristic function on the performance of QPHA: the lower the estimated cost to reach the destination, the slower the algorithm becomes as it expands more nodes to find the optimal route. The figure also shows that the smallest computation time is reached using the third version of the algorithm. Comparing the performance of QPA and QPHA with regard to computation time, the third version of QPHA is two times faster than QPA in finding the quickest path in the central area. QPHA also outperforms QPA in finding the optimal routes in the remote area. As shown in Figure 32, QPHA was thirteen times faster at calculating the route of the 4 km trip, around ten times faster for the 6 km and 8 km trips, and five times faster for the 10 km trip.

Comparing the required times to find the best route for trips of the same length in the central area and the remote area, it is of note that the size of the map impacts the computation time. The more junctions there are in the map, the more time is required to find the optimal route. In general, the third version of QPHA was around twenty times faster at finding the quickest routes in the remote area than in the central area. However, even for the longest trip in the central area, search time did not exceed 13 milliseconds, a reasonable time compared to that of the Google routing service and OSM routing service.

In the case of Doha map and Glasgow map, QPHA outperform QPA in all trip scenarios. As show in Figure 35 and Figure 36 QPHA is able to find the fastest route in a short time comparing to QPA which is due to the heuristic approach in QPHA.
The results indicate that QPHA outperforms QPA in finding the optimal routes in both the central and remote areas. QPHA performs well for both long and short trips

due to its heuristic nature and the optimal travel time estimation provided by the heuristic function.

7.5 Verifying SAA Routes

7.5.1 Approach

The objective of this experiment was to verify the ability of the SAA algorithm to select safe routes without sacrificing travel time. The experiment began by generating the quickest route using the QPHA algorithm, and then used the SAA algorithm to generate the optimal route using different combinations of β and ∞ to show the impact of high values of β on the generated route. It was expected that higher values of β would result in longer and safer routes.

7.5.2 Details

The experiment was conducted on three maps that are : the map of the central area, Doha map and Glasgow map.

7.5.2.1 Central Area

In the first step, the algorithm QPHA was used to generate the quickest route between two locations in the center area (route1). Figure 35 shows route1 with an edge (*e1*) with risk factor of 5. In the next step, the algorithm SAA was used to generate the optimal route (route2) with the objective of obtaining the quickest route without regard to safety level. The quickest route was generated using SAA by assigning β with 0, and ∞ with 1. Figure 36 shows route2; route2 and route1 use the same edges, including *e1,* because neither considers safety level.

The next step was to generate a route that avoided *e1*. SAA was used to generate a route between the two locations many times, with the value of β increasing

gradually. Figure 37 shows the route generated when (route3). As presented in the figure, SAA selected different edges to generate route3. The edge *e1* was excluded in the new route; however, route3 has an edge *e2* with a risk factor of 4, lower than the risk factor of *e1*. To generate a safer road that excludes *e1* and *e2*, the value of β was gradually increased, and at the value of 0.5, the generated route (route4), shown in Figure 38 , excluded the edges.

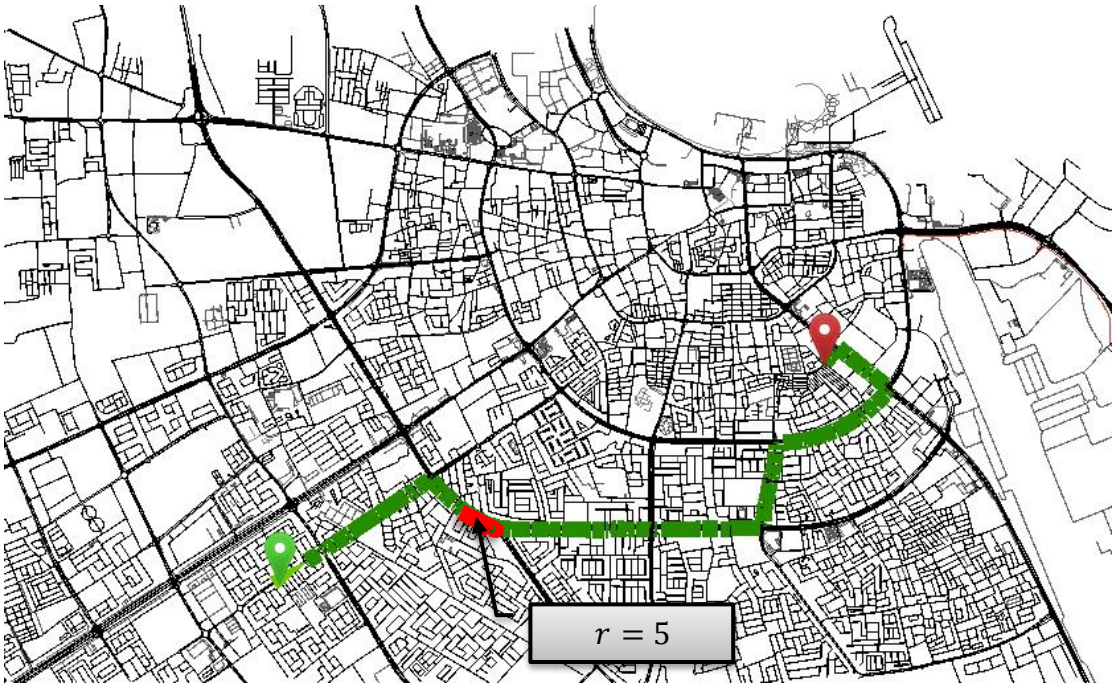


Figure 35 – Safety Aware Experiment – Central Area - Route1

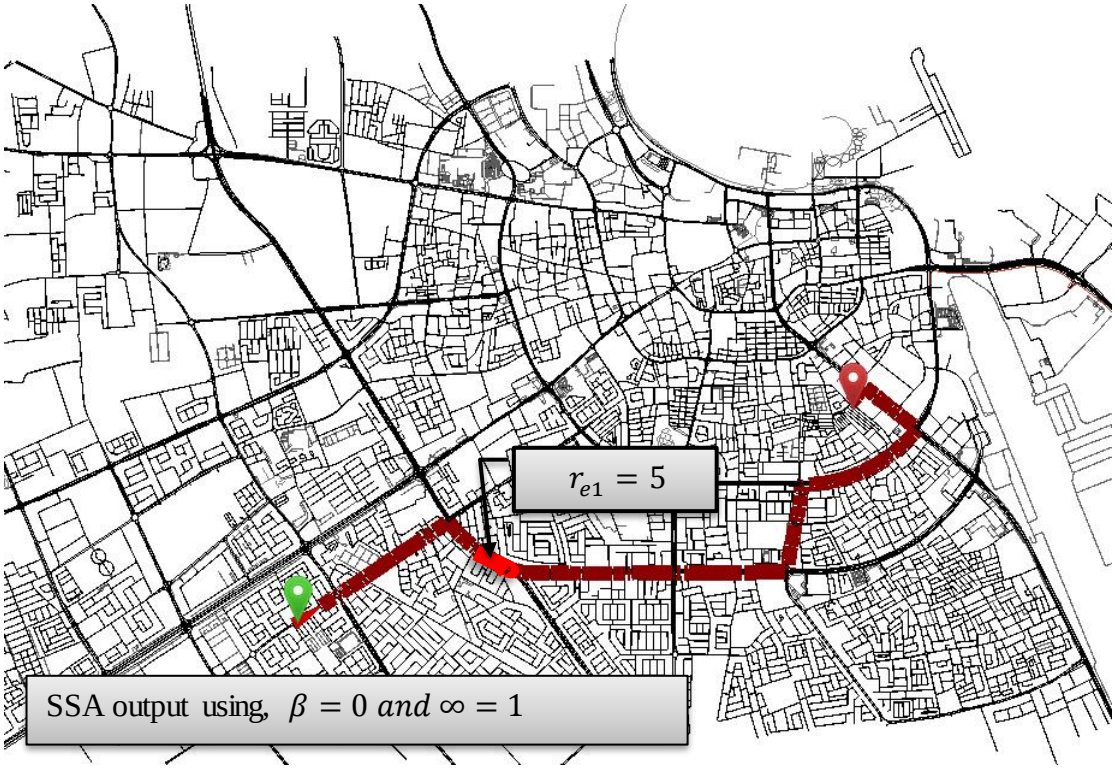


Figure 36 – Safety Aware Experiment – Central Area - Route2

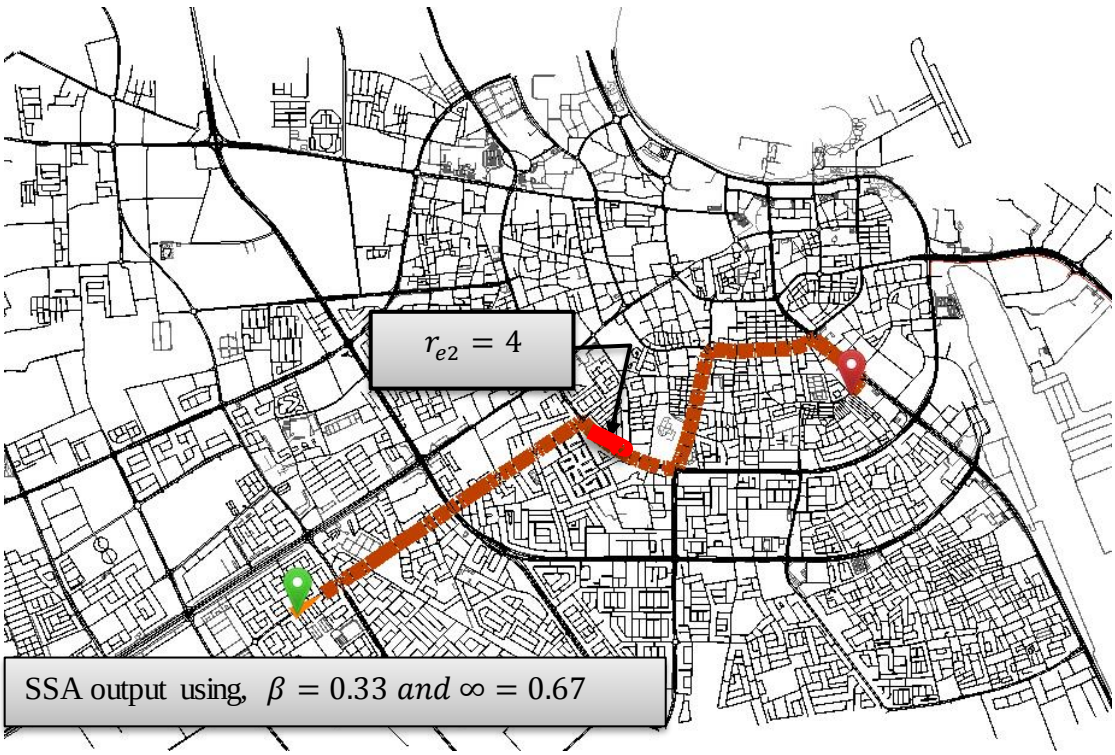


Figure 37 – Safety Aware Experiment – Central Area - Route3

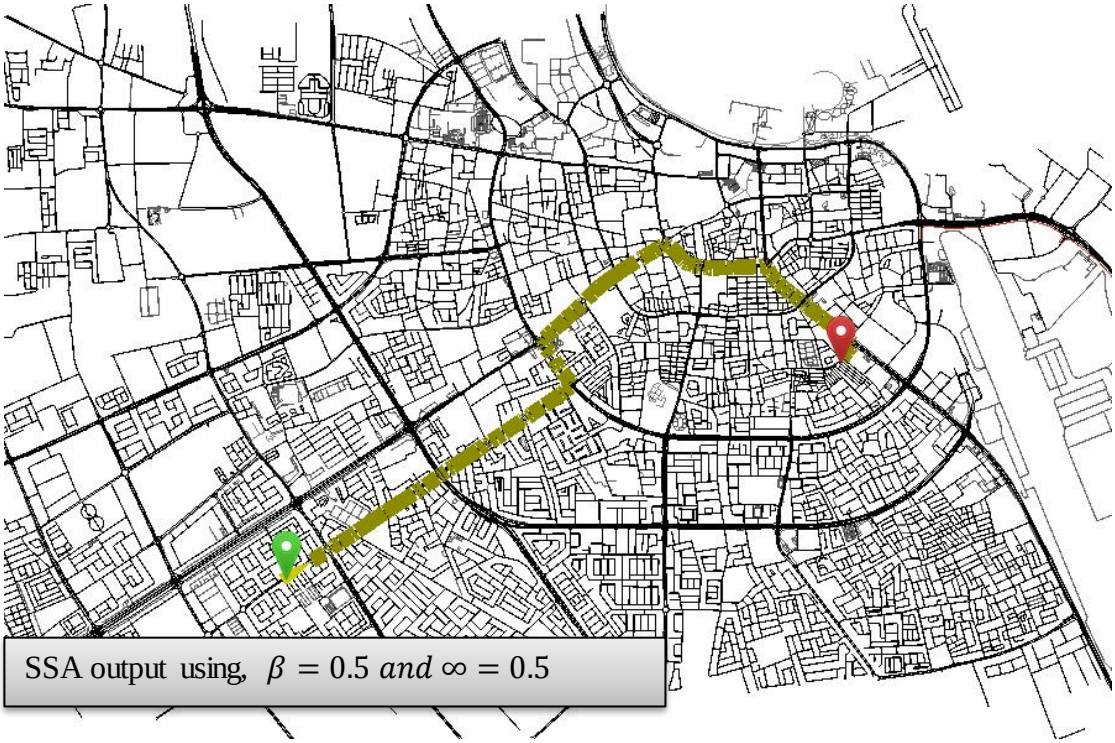


Figure 38 – Safety Aware Experiment – Central Area - Route4

7.5.2.2 Central Area Result Analysis

Table 10 shows the collected information from the simulation of route1, route2, route3 and route4. The results show that in central area, the high values of β increase the travel time and the safety level along the route.

Table 10- Safety Aware Experiment – Central Area - Routes Information

Route	Duration	Length	$\boldsymbol{\beta}$	Max. Risk Factor
route1	355.00	6517.11	not supported by QPHA	1
route2	355.00	6517.11	0	5
route3	422.00	5937.56	0.33	4
route4	449.00	5458.32	0.5	1

7.5.2.2.1 DOHA

In the first step, the algorithm QPHA was used to generate the quickest route between two locations in Doha (route1). The route has a group of consecutive edges with risk factor of four (*e1 group*). The route has also another group of consecutive edges with risk factor of five (*e2 group*). Figure 39 shows route1 and the two groups of edges.

The objective now is to generate a safer route which avoid the group *e2* and includes the group *e1*. SAA is used to generate the safer route by increasing the value of β gradually. Figure 39Figure 37 shows that the group *e2* was replaced by a new group of edges when the value of β is 0.16.

To generate a safer route which excludes both *e1* and *e2*. This result was achived when increasing the value of β to reach 0.22. Figure 41 shows that the group *e1* was replaced by a new group of edges.

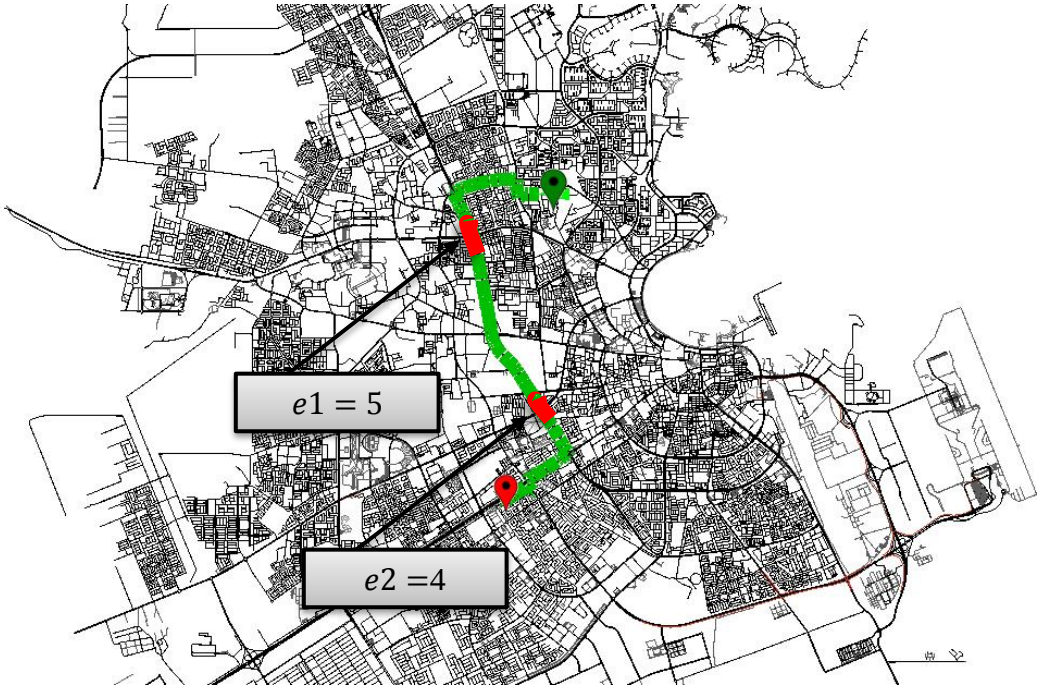


Figure 39 – Safety Aware Experiment – Doha - Route1

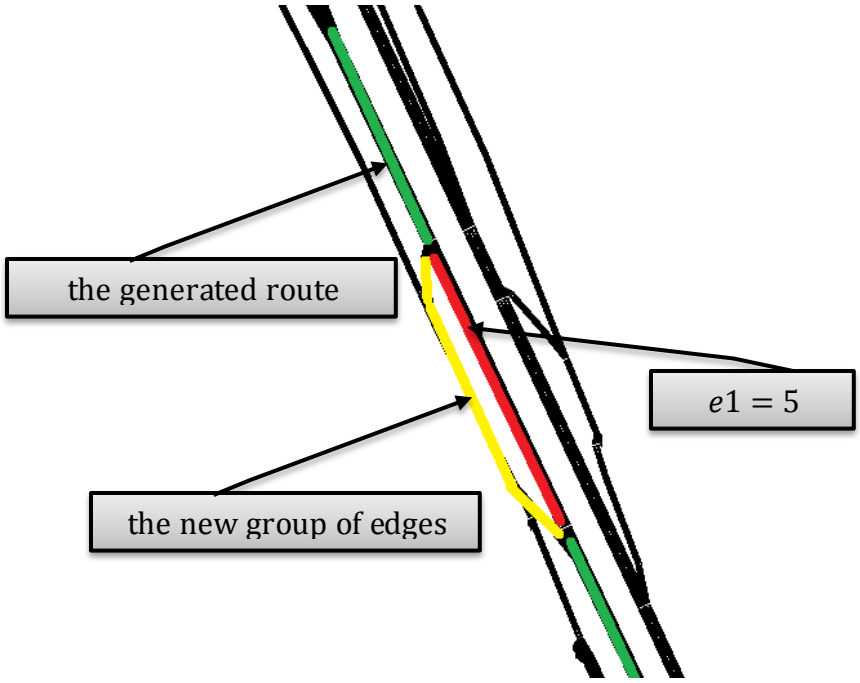


Figure 40 – Safety Aware Experiment – Doha – Route2

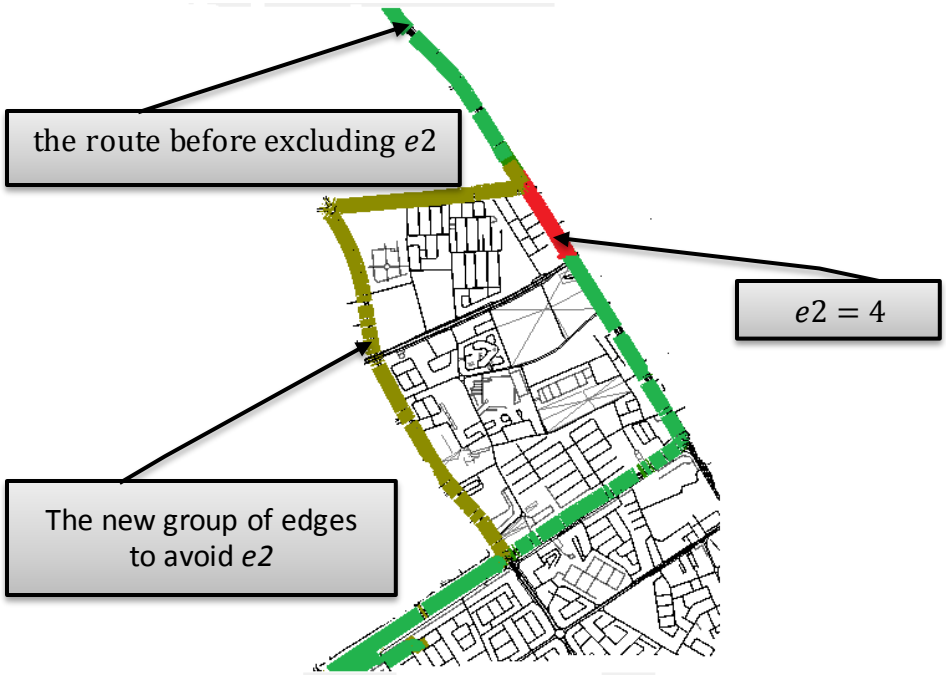


Figure 41 - Safety Aware Experiment – Doha – Route3

7.5.2.3 Doha Result Analysis

Table 11 shows the collected information from the simulation of route1, route2, and route3. The result shows that the high values of β increased the travel time between and the safety level along the route.

Table 11- Safety Aware Experiment – Doha – Routes Information

Route	Duration	Length	$\boldsymbol{\beta}$	Max. Risk Factor
route1	686	14098	not supported by QPHA	1
route2	701	14102	0.22	5
route3	720	15083	0.16	5

8 CONCLUSIONS

This work provides a thorough performance evaluation of two vehicle routing algorithms followed by analysis and comparison of the results. This evaluation of quickest routing algorithms was carried out using data from real transportation networks. The performance assessments for different scalability levels and trip lengths were used to identify the most suitable algorithms; the results showed that QPHA outperformed QPA in finding the optimal routes. QPHA performed well due to its heuristic nature and the optimal travel time estimation provided by the heuristic function. This project also formalized the SQPP problem as a bi-objective optimization problem and provided a solution to generating the optimal route by combining the two objectives, the travel time and the risk of the route, into a single objective. The work used the iRap road assessment method to quantify the risk on the streets and the information provided by OSM to predict the travel time. The algorithm SAA was designed to solve the SQPP problem, generating the safest path without sacrificing travel time given specification of the accepted safety level. The algorithm was implemented and tested in different scenarios. The results showed the efficacy and the efficiency of SAA in selecting edges from different risk groups to balance the safety and travel time objectives according to their respective specified weighting factors. By synthesizing safety data, traffic data, and algorithmic solutions, the research developed an algorithm that can help people drive more safely and thereby reduce the number of accidents and deaths and injuries from road accidents.

9 FUTURE WORK

Future work can further optimize the performance of the algorithm and add useful features. Integration with research centers would help obtain real-time data. Specifically, integration with QTTSC and QMIC would help obtain road safety levels and travel times on the roads, respectively. The algorithm could be improved by adding the option of finding a shortest path with the best departure time. Future research could also evaluate speed-up techniques to deal with large networks. For example, an efficient hierarchical routing algorithm in which a given road network is organized as a multiple-layer hierarchy has been proposed [47], and another study has suggested an efficient hierarchical routing algorithm that finds a near-optimal route, with a network pruning technique incorporated into the algorithm to reduce the search space [48].

References

[1] eea.europa.eu, "Car ownership rates projections," 10 9 2010. [Online]. Available: http://www.eea.europa.eu/data-and-maps/figures/car-ownership-rates-projections. [Accessed 23 11 2015].

[2] W. H. Organization, "Global status report on road safety 2015," [Online]. Available: http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/. [Accessed 24 10 2016].

[3] F. H. Administration, "FHWA Road Safety Audit Guidelines," Federal Highway Administration, [Online]. Available: http://safety.fhwa.dot.gov/rsa/guidelines/appendix_a.htm. [Accessed 24 September 2016].

[4] S. Kumar and D. Toshniwal, "A data mining approach to characterize road accident locations," *Journal of Modern Transportation,* vol. 24, no. 1, pp. 62-72, 2016.

[5] U. F. H. A. R. a. Technology, "Prediction of the Expected Safety Performance of Rural Two-Lane Highways," Federal Highway Administration, 13 August 2016. [Online]. Available: https://www.fhwa.dot.gov/publications/research/safety/99207/02.cfm. [Accessed 24 September 2016].

[6] "Introduction to Road Safety Assessments," Maricopa Association of Governments (MAG), 3 December 2010. [Online]. Available:

http://azmag.gov/Documents/TSC_2010-12-03_Introduction-to-Road-Safety-Assessments.pdf. [Accessed 5 July 2016].

[7] "iRAP," [Online]. Available: http://www.irap.net/. [Accessed 01 May 2015].

[8] Ashghal, "Signing a MoU with the International Road Assessment Programme," Ashghal, December 2014. [Online]. Available: http://www.ashghal.gov.qa/en/MediaHub/News/Pages/Signing-a-Memorandum-of-Understanding-with-the-International-Road-Assessment-Programme.aspx. [Accessed 11 07 2016].

[9] S. Djahel, J. Murphy and V. T. Ngoc Nha, "A Comparative Study of Vehicles' Routing Algorithms for Route Planning in Smart Cities," *2012 First International Workshop on Vehicular Traffic Management for Smart Cities (VTM),* 2012.

[10] K. L. Cooke and E. Halsey, "The shortest route through a network with time-dependent internodal transit times," *Journal of Mathematical Analysis and Applications,* vol. 14, no. 3, pp. 493-498, 1966.

[11] N. T. S. Committee, "National Road Safety Strategy," National Traffic Safety Committee, Doha, 2013.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Cambridge (Massachusetts): The MIT press, 2007, pp. 492-508.

[13] P. N. a. E. D. S. Russell, Artificial intelligence, Upper Saddle River, NJ: Pearson, 2009.

[14] F. B. Zhan and C. E. Noon, "Shortest Path Algorithms: An Evaluation Using Real

Road Networks," *Transportation Science,* vol. 32, pp. 65-73, 1998.

[15] S. Wang, S. Djahel, J. McMains, C. McKenna and L. Murphy, "Comprehensive performance analysis and comparison of vehicles routing algorithms in smart cities," in *Global Information Infrastructure Symposium, 2013*, Trento, 2013.

[16] I. Chabini, "Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithms with Optimal Run Time," *Transportation Research Record: Journal of the Transportation Research Board,* vol. 1645, pp. 170-175, 1999.

[17] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operations Research,* vol. 16, no. 3, pp. 395-412, 1969.

[18] B. C. Dean, "Shortest Paths in FIFO Time-Dependent Networks: Theory and Algorithms," 2004.

[19] S. Felix and J. Galtier, "Shortest paths and probabilities on time-dependent graphs — Applications to transport networks," in *ITS Telecommunications (ITST), 2011 11th International Conference on,* St. Petersburg, 2011.

[20] F. Benjamin Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures," *Journal of Geographic Information and Decision Analysis,* vol. 1, no. 1, pp. 69-82.

[21] Mingjun Wei and Yu Meng, "Research On The Optimal Route Choice Based On Improved Dijkstra," in *IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA),* 2014.

[22] Lingkui Meng, Zhenghua Hu, Changqing Huang, Wen Zhang and Tao Jia,

"Optimized Route Selection Method based on the Turns of Road Intersections: A Case Study on Oversized Cargo Transportation," *ISPRS International Journal of Geo-Information*, vol. 4, pp. 2428-2445, 2015.

[23] L. Rosyidi, H. Pradityo, D. Gunawan and R. Sari, "Timebase dynamic weight for Dijkstra Algorithm implementation in route planning software," in *Intelligent Green Building and Smart Grid (IGBSG)*, Taipei, 2014.

[24] R. Btllman, "ON A ROOTING PROBLEM," 20 December 1956. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/606258.pdf. [Accessed 20 July 2016].

[25] B. Ding, J. Xu Yu and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT '08 Proceedings of the 11th international conference on Extending database technology*, New York, 2008.

[26] H. Kingsbury, D. Harris and P. Durdin, "Journey Optimisation by Safest Route," in *Australian Road Safety Conference*, Queensland, 2015.

[27] E. Galbrun, K. Pelechrinis and E. Terzi, "Safe Navigation in Urban Environments," in *The 3rd International Workshop on Urban Computing*, New Your, 2014.

[28] J. M. S. Grau, E. Chalkia, E. Bekiaris, G. Ayfandopoulou, C. Ferarini and E. Mitsakis, "Transport Research Arena," in *Routing Algorithms for the Safe Transportation of Pupils to School Using School Buses*, Paris, 2014.

[29] "SAFEWAY2SCHOOL," [Online]. Available: http://www.safeway2school-eu.org/. [Accessed 25 August 2016].

[30] H. Kingsbury, D. Harris and P. Durdin, "Journey Optimisation by Safest Route," in

Australasian Road Safety Conference , Perth, 2015.

[31] W. Jigang, S. Jin, H. Ji and T. Srikanthan, "Algorithm for Time-dependent Shortest Safe Path on Transportation Networks," in *The International Conference on Computational Science*, Singapore, 2011.

[32] I.-C. Chang, H.-T. Tai, F.-H. Yeh, D.-L. Hsieh and S.-H. Chang, "A VANET-Based Route Planning Algorithm for Travelling Time- and Energy-Efficient GPS Navigation App," *International Journal of Distributed Sensor Networks,* vol. 2013, pp. 1-14, 2013.

[33] A. H. Khosroshahi, P. Keshavarzi, Z. D. KoozehKanani and J. Sobhi, "Acquiring real time traffic information using VANET and dynamic route guidance," *2011 IEEE 2nd International Conference on Computing, Control and Industrial Engineering,* 2011.

[34] A. Skriver and K. Andersen, "A label correcting approach for solving bicriterion shortest-path problems," *Computers & Operations Research,* vol. 27, no. 6, pp. 507-524, 2000.

[35] M. R.Gary and D. S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, New York: W. H. Freeman & Co, 1979.

[36] C. Massimiliano and D. Paolo, Multi-objective Management in Freight Logistics, Springer, 2008.

[37] A. Skriver and K. Andersen, "A label correcting approach for solving bicriterion shortest path problem," *Computera & Operations Research,* vol. 27, no. 6, pp. 507-524, 2000.

[38] P. Hansen, "Bicriterion Path Problems," in *Multiple Criteria Decision Making Theory and Application*, vol. 177, Springer Berlin Heidelberg, 1980, pp. 109-127.

[39] R. Dial, F. Glover, D. Karney and D. Klingman, "A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees," *Networks*, vol. 9, no. 3, pp. 215-248, 1979.

[40] J. Mote, I. Murthy and D. L. Olson, "A parametric approach to solving bicriterion shortest path problems," *European Journal of Operational Research*, vol. 53, no. 1, pp. 81-92, 1991.

[41] A. Raith and M. Ehrgott, "A comparison of solution strategies for biobjective shortest path problems," *Computers & Operations Research*, vol. 36, no. 4, pp. 1299-1331, 2009.

[42] C. Mohamed, J. Bassem and L. Taicir, "A genetic algorithms to solve the bicriteria shortest path problem," *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 851-858, 2010.

[43] D. E. Kaufman and R. L. Smith, "Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application," *Journal of Intelligent Transportation Systems*, vol. 1, no. 1, pp. 1-11, 1993.

[44] A. orda and R. rom, "Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length," *Journal of the ACM*, vol. 37, no. 3, pp. 607-625, 1990.

[45] I. Chabini and S. Lan, "Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 3, no. 1, pp. 60-74, 2002.

[46] A. Patel, "Amit's A* Pages," 2010. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html. [Accessed 1 June 2016].

[47] V. T. N. N. S. Djahel and J. Murphy, "A comparative study of vehicles' routing algorithms for route planning in smart cities," in *Vehicular Traffic Management for Smart Cities (VTM)*, Dublin, 2012.

[48] "A Large-Scale SUMO-Based Emulation Platform".

[49] "http://sumo.dlr.de/wiki/NETCONVERT," SUMO, 30 09 2014. [Online]. Available: http://sumo.dlr.de/wiki/NETCONVERT. [Accessed 30 10 2015].

[50] "Definition of Vehicles, Vehicle Types, and Routes," SUMO, 20 9 2014. [Online]. Available: http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes#Car-Following_Models. [Accessed 15 5 2016].

[51] "About OpenStreetMap," [Online]. Available: http://wiki.openstreetmap.org/wiki/About_OpenStreetMap. [Accessed 29 3 2016].

[52] "Import/Catalogue," [Online]. Available: http://wiki.openstreetmap.org/wiki/Import/Catalogue. [Accessed 29 3 2016].

[53] S. Wiki, "Contributed/SUMO Traffic Modeler," SUMO, 12 September 2011. [Online]. Available: http://sumo.dlr.de/wiki/Contributed/SUMO_Traffic_Modeler. [Accessed 5 May 2015].

[54] G. Jagadeesh and T. Srikanthan, "Route computation in large road networks: a hierarchical approach," *IET Intelligent Transport Systems*, vol. 2, no. 3, p. 219,

2008.

[55] G. Jagadeesh, T. Srikanthan and K. Quek, "Heuristic techniques for accelerating hierarchical routing on road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 4, no. 301-309, p. 3, 2002.

APPENDIX A: ALGORITHMS

Quickest Path Algorithm

The java code of QPA Algorithm

```
public class QPA implements RoutingAlgorithm{

// The road network
private SimulationNetwork simulationNetwork;

// The set of settled junctions
private Set<Junction> settledJunctions;

// The set of un settled junctions
private FibonacciHeap<JunctionKey, Junction> unSettledJunctions;

// A map to access FibonacciHeap entries in the 'unSettledJunctions' by
     junction id
private Map<String, Heap.Entry<JunctionKey, Junction>> fibonacciEntries;

// A map to store the predecessor for each junction.
private Map<Junction, Junction> predecessors;

// A map to store the cost from the source point to a junction.
private Map<Junction, Double> nodesCost;

public QPA(SimulationNetwork simulationNetwork) throws Exception {
     this.simulationNetwork = simulationNetwork;
 }

/**
 * Calculate the lowest cost between the source node and all other nodes in
     the map.
 * @return
 */
public QPA execute(String junctionId, String destinationId) {
       Junction source =
       simulationNetwork.getJunctionsByIdMap().get(junctionId);

       settledJunctions = new HashSet<Junction>();
       nodesCost = new HashMap<Junction, Double>();
       predecessors = new HashMap<Junction, Junction>();
       unSettledJunctions = new FibonacciHeap<JunctionKey, Junction>(new
       JunctionKeyComparator());
       fibonacciEntries = new HashMap<String,
       Heap.Entry<JunctionKey,Junction>>();
```

```
        fibonacciEntries.put(source.getId(), entry);

        for(Junction junction:simulationNetwork.getJunctions()){
               entry = unSettledJunctions.insert(new
       JunctionKey(junction.getId(), Double.MAX_VALUE, null), junction);
               fibonacciEntries.put(junction.getId(), entry);
               nodesCost.put(junction, Double.MAX_VALUE);
        }

        nodesCost.put(source, 0d);

        while (unSettledJunctions.getSize() > 0) {
               entry =  unSettledJunctions.extractMinimum();

               if (entry.getValue().getId().equals(destinationId)){
                      // Done
                      break;
               }

               settledJunctions.add(entry.getValue());
               extractMin(entry.getValue(),
               entry.getKey().getFollowedEdgeId());
        }

        return this;
 }

 /**
   Finds the edge, in the unSettledJunctions, with the minimum cost to each
   the destination
 */
 private void extractMin(Junction node, String fromEdge) {
        List<Edge> adjacentNodes = getNeighbors(node, fromEdge);

        for (Edge edge : adjacentNodes) {
           Double newCost = nodesCost.get(node) + getEdgeCost(edge);

           // Check if the current known cost to reach the adjacent
           // node is better/worst than the calculated one.
           if (nodesCost.get(edge.getToJunction()) > newCost) {
               nodesCost.put(edge.getToJunction(), newCost);
               unSettledJunctions.decreaseKey(fibonacciEntries.get(edge.getTo
               Junction().getId()), new
               JunctionKey(edge.getToJunction().getId(), newCost,
               edge.getId()));
               predecessors.put(edge.getToJunction(), node);
           }
        }

 }
```

```
public Double getEdgeCost(Edge edge) {
	return edge.getTravelTimeInSeconds() +
	edge.getToJunction().getAvgTravelTime(edge.getId());
}


/**
 * Returns the adjacent junctions to the parameter 'node'
 */
private List<Edge> getNeighbors(Junction node, String fromEdge) {
	List<Edge> edges = new ArrayList<Edge>();
	for(Edge edge: node.getFromEdges()){

		if (!settledJunctions.contains(edge.getToJunction()) &&
		  (fromEdge == null ||
		  simulationNetwork.getConnections().contains(
		  new Connection(fromEdge, edge.getId())))){
			edges.add(edge);
		}
	}
	return edges;
}

public Set<Junction> getSettledJunctions() {
	return settledJunctions;
}

public Map<Junction, Junction> getPredecessors() {
	return predecessors;
}

}
```

Quickest Path Algorithm-Heuristic

The java code of QPHA V1 Algorithm

```
public class QPHAV1 implements RoutingAlgorithm {
// The road network
private SimulationNetwork simulationNetwork;

// The set of settled junctions
private Set<Junction> settledJunctions;

// The set of unsettled junctions
private FibonacciHeap<AStarJunctionKey, Junction> unSettledJunctions;

// A map to access Fibonacci Heap entries in the 'unSettledJunctions' by
junction id
private Map<String, Heap.Entry<AStarJunctionKey, Junction>>
fibonacciEntries;

// A map to store the predecessor for each junction.
private Map<Junction, Junction> predecessors;

// A map to store the cost from the source point to a junction.
private Map<Junction, Double> nodeCosts;

private static final double SPEED_IN_METER_PER_SECOND = 5.55556;  //20KPH

public QPHAV1(SimulationNetwork simulationNetwork) throws Exception {
      this.simulationNetwork = simulationNetwork;
}

@Override
public QPHAV1 execute(String sourceId, String destinationId) {
      Junction source =
      simulationNetwork.getJunctionsByIdMap().get(sourceId);
      Junction target =
      simulationNetwork.getJunctionsByIdMap().get(destinationId);

      settledJunctions = new HashSet<Junction>();
      nodeCosts = new HashMap<Junction, Double>();
      predecessors = new HashMap<Junction, Junction>();
      unSettledJunctions = new FibonacciHeap<AStarJunctionKey, Junction>
       (new AStarJunctionKeyComparator());

      fibonacciEntries = new HashMap<String,
      Heap.Entry<AStarJunctionKey,Junction>>();

      Heap.Entry<AStarJunctionKey,Junction> entry =
      unSettledJunctions.insert(new AStarJunctionKey(source.getId(), 0D,
      null), source);
```

```
            fibonacciEntries.put(source.getId(), entry);
            nodeCosts.put(source, 0d);
            while (unSettledJunctions.getSize() > 0) {
                  entry =  unSettledJunctions.extractMinimum();
                  if (entry.getValue().getId().equals(target.getId())){
                        // Done
                        break;
                  }
                  settledJunctions.add(entry.getValue());
                  extractMin(entry.getValue(),
                  entry.getKey().getFollowedEdgeId(), target);
            }

            return this;
      }

      /**
       * Finds the edge, in the unSettledJunctions, with the minimum cost to
      reach the destination
       */
      private void extractMin(Junction fromNode, String fromEdge, Junction
      target) {
            List<Edge> adjacentNodes = getNeighbors(fromNode, fromEdge);
            for (Edge edge : adjacentNodes) {
                  Double newCost = nodeCosts.get(fromNode) + getEdgeCost(edge);
                  if
                  (fibonacciEntries.containsKey(edge.getToJunction().getId())){
                  // Check if the current known cost to reach the adjacent
                  // node is better/worst than the calculated one.
                        if (nodeCosts.get(edge.getToJunction()) > newCost) {
                              nodeCosts.put(edge.getToJunction(), newCost);
                              unSettledJunctions.delete(fibonacciEntries.get(
                              edge.getToJunction().getId()));
                              predecessors.put(edge.getToJunction(), fromNode);
                        }
                  }else{
                        nodeCosts.put(edge.getToJunction(), newCost);
                        predecessors.put(edge.getToJunction(), fromNode);
                  }

                  Heap.Entry<AStarJunctionKey,Junction> entry =
                  unSettledJunctions.insert(new
                  AStarJunctionKey(edge.getToJunction().getId(),
                  getHeuristicValue(fromNode, target, newCost, edge),
                  edge.getId()), edge.getToJunction());
                  fibonacciEntries.put(edge.getToJunction().getId(), entry);
            }
      }
```

```
/**
 * Returns the cost to reach 'fromNode' +  the heuristic cost to the target
junction
 */
public double getHeuristicValue(Junction fromNode, Junction target,
            Double newCost, Edge edge) {
      return newCost + (getManhattanDistance(fromNode, target)/
SPEED_IN_METER_PER_SECOND);
}


 public Double getEdgeCost(Edge edge) {
      return edge.getTravelTimeInSeconds() +
      edge.getToJunction().getAvgTravelTime(edge.getId());
}

public Double getManhattanDistance(Junction source, Junction target) {
      return Math.abs(target.getBounds().x - source.getBounds().x)
            + Math.abs(target.getBounds().y - target.getBounds().y) ;
}


/**
 * Returns the adjacent junctions to the parameter 'node'
 */
private List<Edge> getNeighbors(Junction node, String fromEdge) {
      List<Edge> edges = new ArrayList<Edge>();
      for(Edge edge: node.getFromEdges()){
            if (!settledJunctions.contains(edge.getToJunction()) &&
            (fromEdge == null ||
            simulationNetwork.getConnections().contains(new
            Connection(fromEdge, edge.getId())))){
                  edges.add(edge);
            }
      }

      return edges;
}

}
```

The java code of QPHA V2 Algorithm

```
public class QPHAV2 extends QPHAV1 {

private static final double SPEED_IN_METER_PER_SECOND = 2.77778;  //10KPH


public QPHAV2(SimulationNetwork simulationNetwork) throws Exception {
            super(simulationNetwork);
}

@Override
public double getHeuristicValue(Junction fromNode, Junction target,
            Double newCost, Edge edge) {
      return newCost + (getManhattanDistance(fromNode, target)/
      SPEED_IN_METER_PER_SECOND);
}
```

The java code of QPHA V3 Algorithm

```
public class QPHAV3 extends QPHAV1 {

private static final double SPEED_IN_METER_PER_SECOND = 1.38889;  //5KPH

public QPHAV3(SimulationNetwork simulationNetwork) throws Exception {
            super(simulationNetwork);
}

@Override
public double getHeuristicValue(Junction fromNode, Junction target,
            Double newCost, Edge edge) {
      return newCost + (getManhattanDistance(fromNode, target)/
      SPEED_IN_METER_PER_SECOND);
}
}
```

Safety Aware Algorithm

The java code of SAA Algorithm

```
public class SafetyAwareAStar extends QPHAV3{

private float travelTimeWeight;
private float riskFactorWeight;
private int maxRiskFactor;

private static final Integer MAX_TRAVEL_TIME_IN_SECONDS = 1500;
private static final Integer MAX_RISK_FACTOR = 5;
private static final double SPEED_IN_METER_PER_SECOND = 1.38889;  //5KPH

public SafetyAwareAStar(SimulationNetwork simulationNetwork,
              float travelTimeWeight, float riskFactorWeight, int
              maxRiskFactor) throws Exception {
       super(simulationNetwork);
       this.travelTimeWeight = travelTimeWeight;
       this.riskFactorWeight = riskFactorWeight;
       this.maxRiskFactor = maxRiskFactor;
}

public Map<String, Double> weights = new HashMap<String, Double>();
       @Override
       public Double getEdgeCost(Edge edge) {
              return
              (((double)super.getEdgeCost(edge)/MAX_TRAVEL_TIME_IN_SECONDS)
              * travelTimeWeight ) +
              ((new Double(edge.getRiskFactor())/MAX_RISK_FACTOR) *
              riskFactorWeight );
       }

@Override
public double getHeuristicValue(Junction fromNode, Junction target,
              Double newCost, Edge edge) {
              double weight = Integer.MAX_VALUE;

              if (edge.getRiskFactor() <= maxRiskFactor){
                     weight = newCost + ((getManhattanDistance(fromNode,
                     target)/ SPEED_IN_METER_PER_SECOND)/
                     MAX_TRAVEL_TIME_IN_SECONDS);
              }

              return weight;
       }
}
```

APPENDIX B: SUMO UTILITIES

Generating Road networks

OSM allows downloading maps for different cities, and SUMO provides a tool to parse OSM files and generate network files, the tool is Netconvert. In this work, Netconvert is used to generate the road network for both the central area and the remote area. The following steps describe the process of generating the road network of the central area:

- In the page https://www.openstreetmap.org/export, the central area is selected, then the button 'Export' is used to export the map and store it in the file central_area.osm

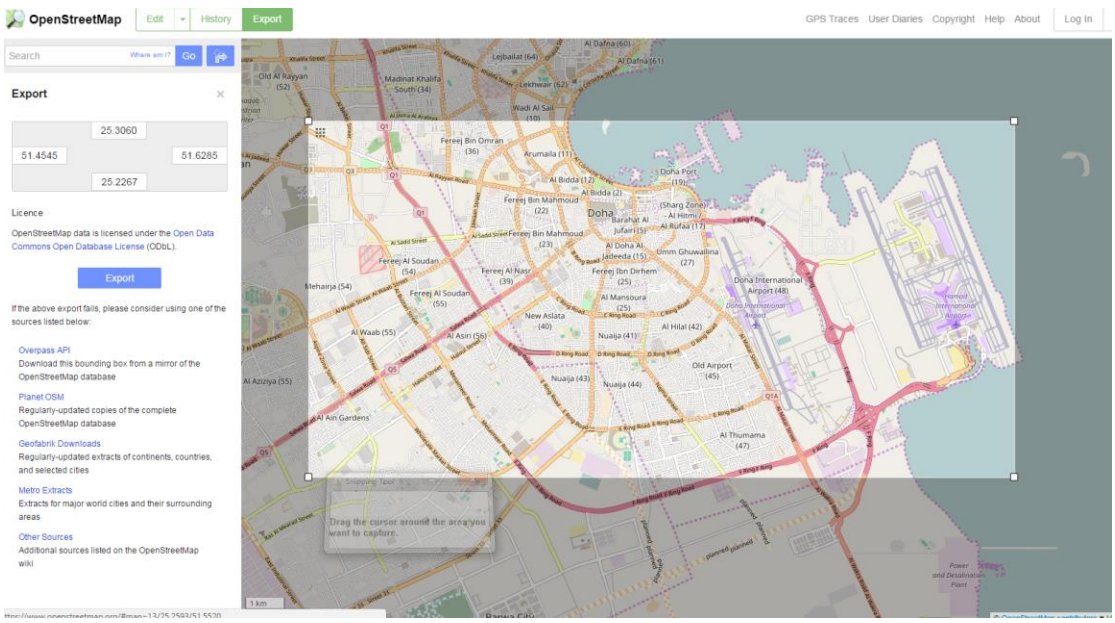


Figure 42 – OSM – Central Area

- The following command is used to run Netconvert and generate the road network file central_area.net.xml

```
SUMO>netconvert --osm-files --output-file central_area.net.xml
```

Running Simulations

Traffic simulation is achieved using a configuration file, .sumo.cfg, which specifies the input files of the scenario, the required input files are

- The road network file that will be used in the simulation.
- The vehicles, the type of the vehicles and their routes.
- Simulation time period, where the period is specified by a number of steps.

The following example shows a simulation scenario in the central area

```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">
   <input>
      <net-file value="central_area.net.xml"/>
      <route-files value="central_area.rou.xml"/>
   </input>
   <time>
      <begin value="4990"/>
      <end value="6000"/>
   </time>
   <time-to-teleport value="-1"/>
</configuration>
```

A traffic simulation in SUMO can be conducted either using the GUI or with command line. The following example, shows how to run the simulation scenario described in the file central_area.sumo.cfg and stores simulation results in the file simulation_result.xml

```
SUMO>sumo -c central_area.sumo.cfg --tripinfo-output simulation_result.xml
```

The next figure shows how the same simulation can be conducted using the GUI