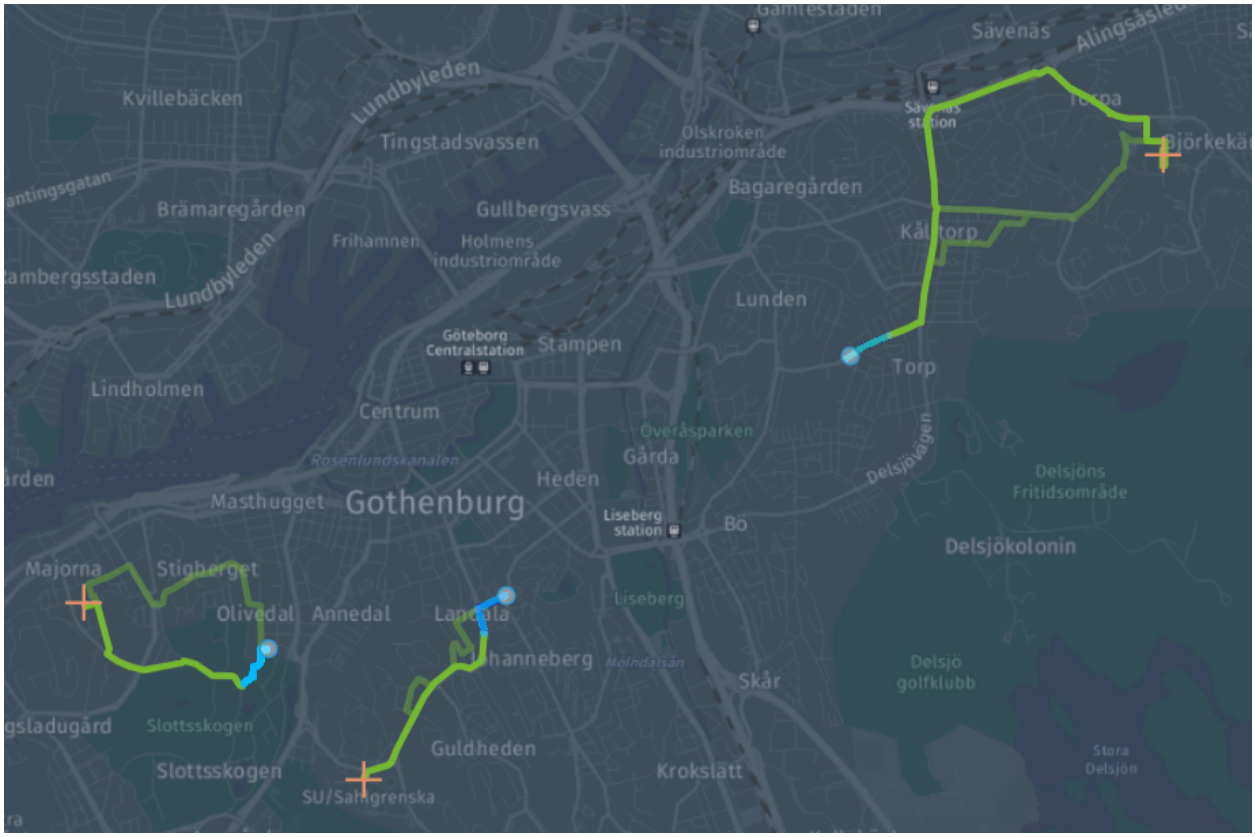
CHALMERS
UNIVERSITY OF TECHNOLOGY



Real-time route prediction of emergency vehicles

An implemented algorithm for predicting the route choices of emergency vehicles, with the purpose of sending preemptive warnings to other drivers

Master's thesis in Mathematics

ALFRED ARVIDSSON
JAKOB HENDÉN

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2022

www.chalmers.se

Master's thesis 2022

Real-time route predicion of emergency vehicles

An implemented algorithm for predicting the route choices of emergency vehicles, with the purpose of sending preemptive warnings to other drivers

ALFRED ARVIDSSON
JAKOB HENDÉN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Algebra and Geometry
Chalmers University of Technology
Gothenburg, Sweden 2022

Real-time route prediction of emergency vehicles
An implemented algorithm for predicting the route choices of emergency vehicles, with the purpose of sending preemptive warnings to other drivers
ALFRED ARVIDSSON
JAKOB HENDÉN

Supervisors:
Marina Axelson-Fisk, Mathematical Sciences
Anna Brunzell, Carmenta Automotive

Examiner:
Martin Raum, Department of Mathematical Sciences

Master's Thesis 2022
Department of Mathematical Sciences
Division of Algebra and Geometry
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A still of a visualisation of the algorithm running. Blue dots are emergency vehicles, orange crosses their destinations, green lines the most probable routes, transparent green other possible routes and blue lines are the area in which to warn drivers.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Real-time route prediciton of emergency vehicles
An implemented algorithm for predicting the route choices of emergency vehicles, with the purpose of sending preemptive warnings to trafficants

ALFRED ARVIDSSON
JAKOB HENDÉN
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Nordic Way 3 is an EU-funded project that aims to develop software solutions for a better and safer traffic environment. One of the companies involved with Nordic Way 3 is Carmenta Automotive who, among other things, are responsible for route prediction of and preemptive warnings about approaching emergency vehicles. By predicting the routes of emergency vehicles and sending preemptive warnings to surrounding drivers this is a contribution to the goals of Nordic Way 3.

In this thesis project, an algorithm for predicting which specific path is chosen by emergency vehicle drivers is developed and implemented. The current position and destination of the emergency vehicle, as well as live data regarding traffic and incidents on the roads is used as basis for the prediction. The modified A*-algorithm is tested using a set of varied realistic emergency missions, and shows promising performance. It generally predicts the most probable path together with a few feasible alternative paths.

Keywords: route, prediction, emergency, vehicle, a-star, a*, algorithm, software

Acknowledgements

We want to thank our supervisors, both at Chalmers and Carmenta Automotive, Marina Axelson-Fisk and Anna Brunzell for their help during the thesis work and their valuable insights. We are also grateful to our opponents Theodor Stenhammar and David Bejmer for insightful feedback. Futhermore we want to thank the other members of the Carmenta Automotive team for their help with implementation. Last but not least, thank you to Martin Raum for great input and making the entire examination process trouble free.

The Authors, May 2022

Contents

List of Figures

List of Tables

1

Introduction

Route planning is the practice of evaluating which path to take through an environment where multiple possible paths will lead to the destination. This is done unconsciously in everyday life when walking, bicycling and driving for example. The evaluated paths can differ in many ways, and in a complicated environment many factors influence the decision. For example speed, road condition, familiarity and certain needs such as bike lanes or gas stations. An emergency vehicle driver choose routes with the goal of arriving at the destination as quickly and as safely as possible.

The way computer scientists solve problems such as route prediction is by developing algorithms that given certain factors produces results that are indicative of the results a human counterpart would produce given the same circumstances. In this thesis project, an algorithm that predicts the path chosen by an Emergency Vehicle driver to their destination will be developed.

1.1 Background

This thesis project will be focused on using software- and data engineering to produce real time route predictions for emergency vehicles, with the purpose of sending preemptive warnings to traffic along this route. The project is a part of a larger initiative called Nordic Way 3, in which several parties with interest in the automotive industry collaborate to develop new technologies centered around connected roads and traffic infrastructure. One of the pilots within the project relates to increased awareness of emergency vehicles on the roads, specifically sending preemptive warnings to the driver's Heads Up Display (HUD) when an emergency vehicle is approaching, as well as evaluating driver behaviour when receiving such warnings.

The purpose of receiving a warning in the driver's HUD is to prepare the driver to be extra careful, and be ready to make way for an overtaking emergency vehicle.

One of Carmenta's roles in Nordic Way 3 is to develop the technology for sending

preemtive warnings to drivers, as well as identifying when and where to send these warnings. As this thesis project aims to create a true-to-life algorithm for predicting route choice by an emergency vehicle driver, as well as implementing this functionality into Carmenta's software product TrafficWatch, it is a direct contribution to Carmenta's work in Nordic Way 3. It is also in a broader sense a contribution towards reaching the interconnected and smart road network of the future, leading to fewer incidents and less damage to infrastructure [1].

Emergency Vehicle Approaching (EVA) messages have been classified as a highly beneficial service for society by the European Commission. The commission have also stated that it should be deployed as soon as possible, as it shortens the response time for emergency vehicles, leads to a better experience for road users, and leads to a smoother and safer traffic flow when an emergency vehicle is approaching [1].

The Swedish National Road and Transport Research Institute (VTI) is conducting research regarding warning regular drivers about approaching emergency vehicles. The results of this thesis will be beneficial to the project, as good predictions of emergency vehicle driving patterns are essential to keeping the false-positive rate low when sending out such warnings [2].

1.2 Purpose

The aim of the thesis is to develop an algorithm that determines different feasible routes along with their relative probability for emergency vehicles in real time. The thesis will investigate which parameters influence route choices and if it is possible to make accurate predictions. The predictions should be usable for sending preemptive warnings to other vehicles along the most probable route.

The prediction algorithm and visualizations of its results are valuable additions to TrafficWatch, as route prediction is a valuable feature for a control-tower application, which is an application used to guide and assist connected vehicles. This plays an important role in improving connected autonomous vehicles' situational awareness. [3]

Connected and Autonomous Vehicles and a smart interconnected road transport system is presented by the Swedish Transport Administration as one important step towards increased accessibility, road safety and reduced climate impact [4]. EVA messages and precise route predictions align well with the development of this vision.

1.3 Limitations

The testing, validation and development of the algorithm is limited to data from the Gothenburg area. It is assumed that this will be representative of most other cities, as well as more rural areas since there is generally a lot fewer factors to take into account there. There is more often than not less traffic, fewer alternative routes and the roads are similarly sized.

The data sources used are limited to the existing data providers currently used by other parts of Carmenta TrafficWatch.

Due to the project's complete length of 20 weeks including research, development, implementation and optimization most phases of the project are limited in terms of what time can reasonably be spent while still ensuring completion. Limitations include the amount of factors taken into account when developing the algorithm, the degree to which the solution is hardware optimized, the algorithm's efficiency beyond functional performance and the evaluation of the algorithm's prediction accuracy.

1.4 Problem Formulation

The project is comprised of two main goals. Firstly, create and optimize an algorithm that produces a set of probabilistically evaluated possible routes for an emergency vehicle given a start and end point, and other real-time traffic related parameters. Secondly, implement the algorithm and a graphical visualization of it as a microservice compatible to be used in TrafficWatch.

In addition to the main goals, the efficacy of the algorithm for implementation in accordance with the specifications of the Emergency Vehicle Awareness pilot in Nordic Way 3 should be discussed.

2

Theory

In this chapter, all concepts and theories this project entails will be presented. These are mainly computer science concepts surrounding algorithm theory, geometry and geospatial systems. In chapter 3, the theory presented here will be put into context.

2.1 Geographic data types

To make computations with and analyze geometric and geographic data, spatial properties such as coordinates and directions need to be represented digitally. In this section, the formats and standards used for storing routes, road networks and paths are explained and defined.

2.1.1 LineString

A LineString is in the context of Geographic Information Systems a geometric object consisting of a series of connected lines. It is formally specified by the sequence of points which act as the connection points between each line segment. A LineString's direction is in the context of this report implied by the order of the points [5].

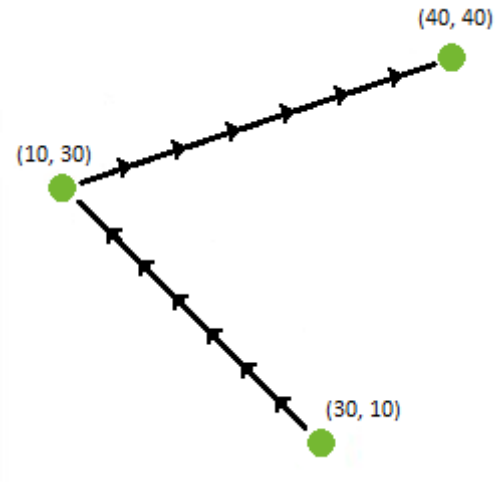


Figure 2.1: An example of a LineString defined by three points.

An example of a street in LineString format is presented below.

Text:
LINESTRING (30 10, 10 30, 40 40)

2.1.2 The GeoJSON standard

GeoJSON is a data format based on JavaScript Object Notation, and is used for exchange of geospatial information. By combining multiple JSON objects, geographic features and their properties are represented by the specific combination sequence [6]. For example, a LineString is a curve specified by a combination of geographic Points expressed in coordinates [7] formatted as a JSON object.

2.2 Graph traversal algorithms

Graph traversal is in computer science the process of traversing vertices in a graph according to some rule, with the purpose of exploring its structure [8]. Example problems that are solved using graph traversal include calculating the shortest path or testing a graph for bipartiteness. Depending on the problem at hand, different rules or traversal algorithms have different efficiency [9, 8].

2.2.1 Greedy algorithms

Algorithms that are greedy follows the problem-solving heuristic of making short-sighted choices that are optimal only given the current state of the environment. By acting locally optimally, greedy algorithms does not evaluate future possibili-

ties or advantages and tend to result in sub-optimal solutions or fail entirely when traversing complex environments [10, 11].

2.2.2 The A* algorithm

The A* algorithm is a popular pathfinding algorithm because of its flexibility and usability in a range of different contexts [12]. The algorithm was originally described by Dijkstra, however in a more general setting [13]. The common implementation of A* differs slightly, for example when choosing in which order to explore the nodes. The A* algorithm is described below in Pseudocode 2.1.

```
openSet := {startNode}
cameFrom := {}

gScore := map with default value PositiveInfinity
gScore[start] := 0

fScore := map with default value PositiveInfinity
fScore[start] := 0

while openSet is not empty:
    current := the node in openSet having the lowest fScore
    if current = goal:
        retreive path by backtracking to startNode

    remove current from openSet
    for each neighbour of current:
        temp_gScore := gScore[current] + d(current, neighbour)
        if temp_gScore < gScore[neighbour]:
            cameFrom[neighbour] := current
            gScore[neighbour] := temp_gScore
            fScore[neighbour] := temp_gScore + h(neighbour)
            if neighbour is not in openSet:
                add neighbour to openSet
```

Pseudocode 2.1: The A* algorithm

The A* algorithm guides its search by evaluating the total cost of the path through each node evaluated, by selecting to explore from the node with the lowest *F-score* [14]. *fScore[n]* should be interpreted as the shortest path from the start to node n.

Since the node with the lowest *fScore* is always chosen first, the optimal paths will be found when the node chosen is the goal node.

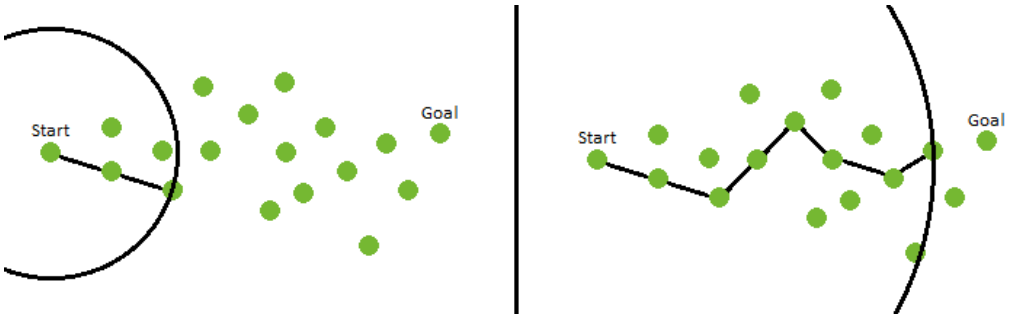


Figure 2.2: Visualization of the A* algorithm searching outwards from the start point.

2.2.2.1 The Bidirectional A* algorithm

The bi-directional A* search algorithm makes two parallel A* searches, one originating in the destination and working backwards, and the other search originating in the start location. When these two searches meet, a good path between the start and goal node is obtained [15] by backtracking outwards from the meeting point (often called midpoint) back to the two searches respective starting points.

A reason for using a bi-directional A*-algorithm is the execution time, since the space searched by the algorithm is generally half compared to a traditional A*-algorithm. The smaller search space is visualized in 2.2 compared to 2.3.

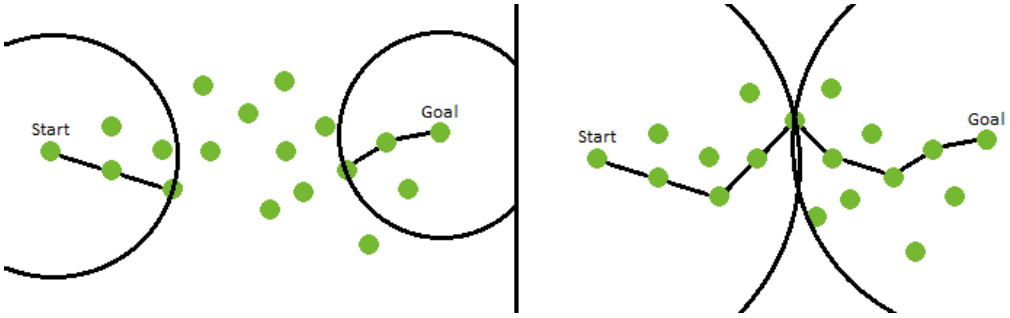


Figure 2.3: The bidirectional a* algorithm searching outwards from both the start and end point.

3

Methods

In the first part of this section, the theoretical methods, definitions, data modification and data structures that were used are described. In the second part, implementations specific to our developed software are described.

3.1 Theoretical methods

In this chapter, all the theoretical models used in this project will be presented. The chapter also includes project-specific definitions, data structures and theoretical concepts. Areas include custom pre-processed data structures and factors, models of a city environment and project-specific statistics used.

3.1.1 Definition of the algorithm's end goal

The algorithm that is developed in this thesis project should compute different routes through a real road network (Gothenburg City) as well as individual relative probabilities of the emergency vehicle driver taking each route using real-time data flows such as the emergency vehicle's current position and other complementary geospatial information.

3.1.2 Definition of an emergency vehicle Mission

An emergency vehicle mission is a task in which an emergency vehicle (such as an ambulance) is supposed to drive to a specific location. The emergency vehicle's location is continuously updated, but the end location remains the same from the point of creation to the point of completion.

3.1.3 Mapping from geospatial data to traversable search tree

In this section the various geospatial data used in this project will be described. The existing geometry data is stored as LineStrings together with metadata including but not limited to their speed class, road type, and a label indicating their allowed driving direction (forward, backwards or both ways). This data was converted into custom data structures chosen for the purpose of providing fast lookup and low graph traversal times.

3.1.3.1 Nodes & Edges

The interconnected road network is represented as a graph of Nodes. In addition to an id and coordinates, each Node contains a list of *inNodes* and *outNodes* representing which adjacent nodes exists in the network. Thus, a connection between two Nodes can be interpreted as a road. These roads (edges) are stored in memory as a dictionary with the start and end node as key, and the weighted distance between them as the value.

```
Definition Node:
    int: id
    double: latitude
    double: longitude
    List of Node: inNodes
    List of Node: outNodes
```

Pseudocode 3.1: Definition of a Node

3.1.3.2 Direction

The definition of a Node, specifically the inclusion of the lists *inNodes* and *outNodes*, implies the direction from which it's possible to arrive into a node, as well as which nodes can be reached when travelling out of a node. The road network structure therefore entails directionality, which is an essential component when depicting a real road network.

Bidirectional roads are created by adding two adjacent nodes to both each others' lists *inNodes* and *outNodes* respectively.

3.1.3.3 Road segment speed classes/categories

The *speed class* indicates the upper speed limit between two nodes. Speed classes exist for the most common speed limits used on Swedish roads. The speed classes are shown in table 3.1.

3.1.3.4 Road types

The *size* indicates what type of road connects two nodes. For example, bicycle roads, country roads or a high-speed motorways are different road types. The purpose of this data is to indicate the ability for an emergency vehicle to exceed the speed limit and easily pass traffic. The road types are described in table 3.2.

Table 3.1: Speed classes

Speed Class Id	Max speed limit
1	130 km/h
2	110 km/h
3	90 km/h
4	70 km/h
5	50 km/h
6	30 km/h

Table 3.2: Road types

Road Type Id	Example
1	Highway
2	Expressway
3	Large road
4	Small road
5	Small accessways, bike lanes, etc.

3.1.4 Speed multiplier

To convert the distance and metadata of a specific road segment to the time it takes to traverse it (which is of greater interest), its distance can be multiplied with the inverse of the speed limit of that road segment. This value is called the *speed multiplier*. Unit analysis clearly describes this rationale:

$$distance[m] * speedMultiplier[1/\frac{m}{h}] = time[h]$$

3.1.5 Road type multiplier

To better model real emergency vehicle driving, the network's weights are scaled depending on the road type of each road segment. This value is called the *road type multiplier*.

Example:
A 90km/h road segment is classified as a large city road with multiple lanes. This road type indicates good ability drive fast and pass other traffic. The weighted

distance of this road segment is adjusted with a road type multiplier $R_m < 1.0$ to make the algorithm interpret this road as shorter than a smaller, but otherwise identical, 90km/h road segment.

3.1.6 Road network weighting

As the goal of the algorithm is to predict the emergency vehicle driver's route choice, the weighting of the road network should represent a road segment's likeliness of being chosen by the driver. The factors speed, road type, distance, together with constants and a collection of penalties (if a segment is not classified as a normal car road, if there are ongoing incidents, etc.), make up the function for the complete evaluation of a road segment. It outputs a weighted score representing the time it would take to traverse it and the likelihood of it being chosen. This weighted score is then used for the tree search that the algorithm performs.

The specific weighting function will not be disclosed in this report.

3.1.7 Available data flows

The live data flows that are implemented is all data from TrafficWatch that is called an "Incident". This data can according be divided into two groups; traffic incidents and road work and maintenance.

Traffic incidents data is information such as the locations and severity of collisions or other unplanned events impacting impacting traffic, like weather or congestion.

Road work and maintenance data contains information such as temporary road closures, maintenance of sidewalks or shafts, asphalt paving or similar. Road work or maintenance can either close a lane, or just obstruct it from normal usage. These cases are distinguished by either removing a road completely if its non-traversable, or penalize the corresponding graph weighting if it is traversable.

3.1.8 The route difference quota

Evaluation of which routes are deemed plausible alternatives to the main route prediction is done using something called the *route difference quota*. This is a statistic that evaluates if an alternative route is sufficiently different from the best route, but still sufficiently good to be a plausible route.

The quota is calculated by dividing the difference in weighted distance (between the fastest route and the route being evaluated) with the total weighted distance of the

route being evaluated that does not overlap with the shortest route. This quota must be below a certain threshold, with the idea being to penalize long detours and prefer varying route choices.

This quota is important as an alternative route needs to be sufficiently different from the main one. If an alternative route could be identical to the main route except for one small turn, it would not be helpful as it does not provide any new indication of what additional areas could be warned. The quota threshold should be set between giving sufficiently diverging paths, and giving too long detours.

3.2 Software methods

This section will describe the software methods used in this project. These include specific software implementations of data flows and details about the microservices' structure.

3.2.1 Calculating distance from geographic coordinates

The distance in meters between two points expressed as geographic coordinates can be calculated by utilizing the spherical law of cosines $acos(sin\phi_1 * sin\phi_2 + cos\phi_1 * cos\phi_2 * cos\Delta\lambda) * R$ [16] where ϕ is latitude, λ is longitude and R is earth's radius.

As the earth is approximately spherical and the mean radius is $6371km$ [17], the distance between two coordinate points is closely enough approximated for most purposes [16] by the following equation:

$$distance = \arccos\Big(\sin\left(\pi * \tfrac{lat_1}{180}\right) * \sin\left(\pi * \tfrac{lat_2}{180}\right) + \cos\left(\pi * \tfrac{lat_1}{180}\right) * \cos\left(\pi * \tfrac{lat_2}{180}\right) * \cos\left(\pi * \tfrac{lon_2 - lon_1}{180}\right)\Big) * 6.371 * 10^6$$

This is the equation used to convert coordinate differences to distance in meters in this project.

3.2.2 Data Loading and pre-processing

To remove redundancy and transform the data to better suit the development of routing algorithms, an SQL query [A] was used to generate a database of each traversable path between every two nodes. In addition to the coordinates of each two neighbouring nodes, each entry in this database contains the pre-computed distance in meters between each node pair as well as its speed category, road type

("size"), and labels indicating if the road segment is traversable by any car ("car"), or only emergency vehicles ("emve").

The resulting database shown in table 3.3 contains all information needed for routing.

road_id	fromLat	fromLon	toLat	toLon	speed	size	car	emve	distMeters
96734	54.3201	10.1425	54.3256	10.1498	7	2	Y	Y	6.77
96734	54.3257	10.1498	54.3201	10.1425	7	2	Y	Y	6.90
96735	54.3257	10.1498	54.3353	10.1636	6	1	Y	Y	20.44
96735	54.3353	10.1636	54.3257	10.1498	6	1	N	Y	18.14
...	...	...	...	...	...	...	...	...	...

Table 3.3: Example rows of the complete routing database table.

3.2.3 Traffic incidents, road work and maintenance data

Traffic incidents, road work and maintenance data was already being distributed on a shared messaging framework in Carmenta's TrafficWatch. The solution implemented is creating a listener that subscribes to these messages, and sends them to the RouteProbabilityService. The messages are then unpacked and the desired information is saved. This is then used to modify the weighted road network accordingly.

3.2.4 The microservice's different functions

The complete system developed in this project consists of two independent services with different responsibilities. Each independent service is described in this section.

3.2.4.1 RouteProbabilityService

This is the main service of this thesis project. This service consumes emergency vehicle mission data such as locations and destinations, extracts the relevant map data from database, builds a the traversable road network tree structure, and computes as well as returns the route predictions used by the front end web application. Route predictions are updated every time a position update from an emergency vehicle is received.

3.2.4.2 AmbulanceSimulator

This service is a substitute for an actual emergency vehicle live mission data flow, used for development, debugging and demo purposes. It sends the mission desti-

nation as well as pre-determined locations with a fixed delay, simulating an actual ambulance sending the data RouteProbabilityService expects. In the 3.1 this service or a live ambulance data feed is called AmbulanceService.

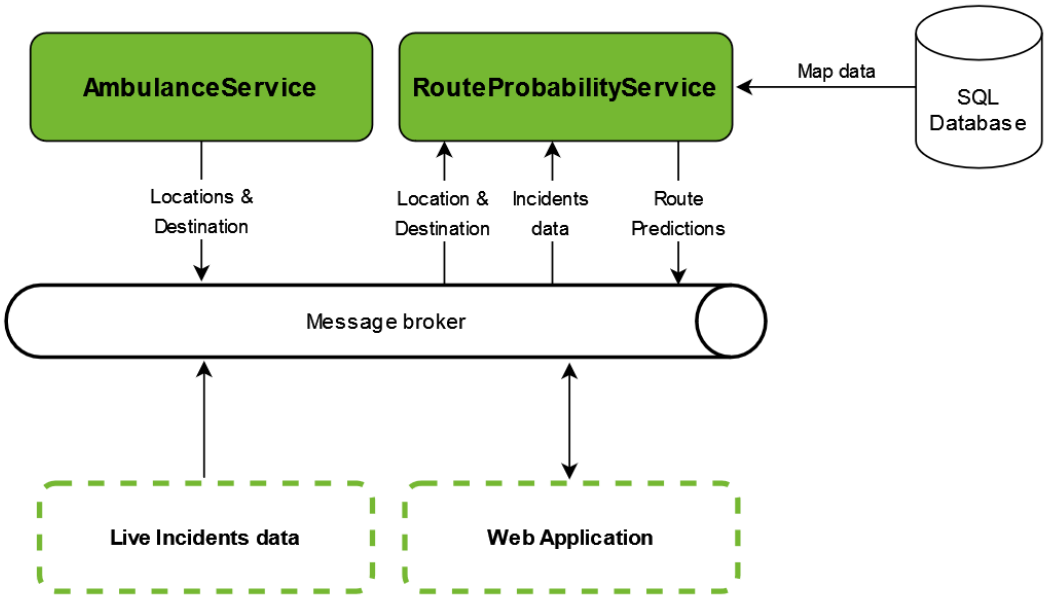


Figure 3.1: Sketch of the software architecture. Solid green services were developed specifically for this project.

3.2.5 Message passing

The messages that are sent between services are sent by *message publishers*, with one publisher for each type of message. The publishers publish their messages to a RabbitMQ *exchange*, which is then responsible for routing the message to the appropriate *queue*. It is these queues that are listened to, meaning a separate program can wait for something to appear on the queue to then take action when it does.

The different messages have different payloads, either GeoJSON objects with coordinate data for points or linestrings, as well as additional attributes such as probability (to customize how the information should be visualized), or an instance of a *Mission*.

An example message that is sent between the RouteProbabilityService and the front end application is shown in Pseudocode (3.2).

```
{
  "Properties": {
    "Id": "2",
```

```
    "Probability": "0.7"
  },
  "Geometry": {
    "Coordinates": "[57.70244, 11.9738357, 57.7036621, 11.1974791]",
    "Type": "LineString"
  },
  "Type": "Feature"
}
```

Pseudocode 3.2: Example payload sent to the front-end application.

3.2.6 Visualization of algorithms calculations and results

The results of the algorithm are multiple calculated routes, where one is the primary and (according to the algorithm) the best. The primary route is visualized as a solid green path on the map, and alternative routes are visualized as weaker transparent green paths. Road segments that should be preemptively alerted are shown as pulsating blue. A snapshot of this can be seen on the cover.

The amount of alternative paths shown is based on the route different quota. This results in more paths being shown in the graphical user interface if there are more feasible alternatives, and less otherwise. The relative probabilities are based on the weighted distances of the routes that pass the threshold.

Figure 3.2 shows an example view of a driver's HUD (Heads Up Display) when getting a preemptive warning message.

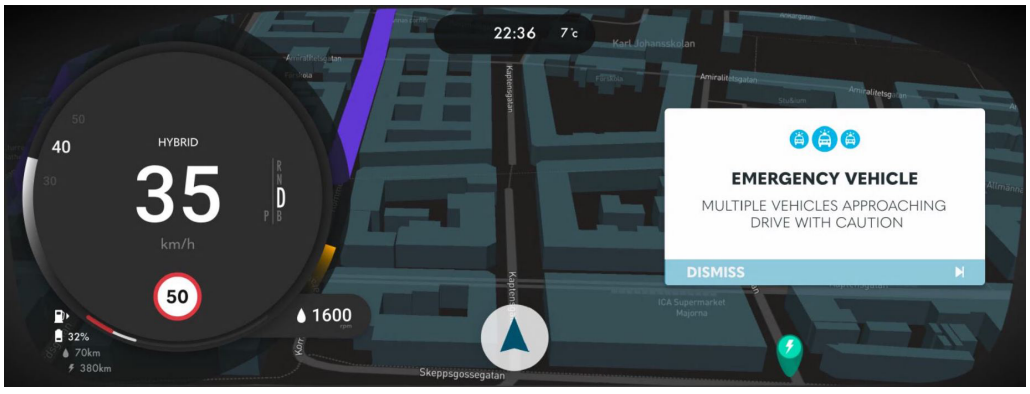


Figure 3.2: A prototype of a preemptive warning message shown in a car HUD

4

Results

This chapter presents the results of the master thesis. Firstly, the focus lies on the iterative development process used when developing the algorithm. Secondly, the final algorithm is presented in pseudocode. After that, a theoretical statement and research analysis is made about software testing and testing adequacy. Finally, the algorithm's performance is exemplified through 9 varied demonstration cases, and the practical utility of the algorithm is reviewed.

4.1 The development of the algorithm

The first algorithm implemented was a naive algorithm that only tried to search its way through the network via directed search. Since the coordinates of the start and end point were known, in each intersection the path that progressed the agent furthest towards the goal (in euclidean distance) was chosen. This worked for easy test scenarios but suffered the inevitable drawbacks of naivety, such as getting stuck in dead-end streets.

Modifying the naive implementation to be able to back out of dead-ends was considered, but was discarded as the inherent problems with naive algorithms was envisioned to bring about future problems and poor performance. Instead focus was put into finding an existing non-naive algorithm to use as a starting point.

When researching path-finding algorithms, multiple articles used modified versions of the A* algorithm. Examples include *Path Planning with Modified a Star Algorithm for a Mobile Robot* [18] and *Path planning of automated guided vehicles based on improved A-Star algorithm* [19]. The A* algorithm is popular in computer science due to its optimality and efficiency, and was a natural choice as a starting point for the algorithm.

A bidirectional version of the A* algorithm was implemented as further improvement, as it theoretically reduces the search space by a factor of two, while still performing similarly.

A problem with all implementations thus far was that they produced only one single result. They were designed to terminate after finding a shortest path. This was problematic since multiple paths are needed to be able to compare them. The comparison is important to be able to calculate relative probabilities.

One way the algorithm could find multiple paths, was by continuing the search until supplementary mid nodes were found. This solution turned out to be inefficient, since the search spaces tended to overlap significantly before feasible route alternatives were identified. Large overlaps means unnecessary calculations and slow execution. The search space and computation time grows exponentially when search radii of the two searches grow.

Notice in figure 4.1 that the search space (sum of the circles' areas) is very large when finding a distinctly different route.

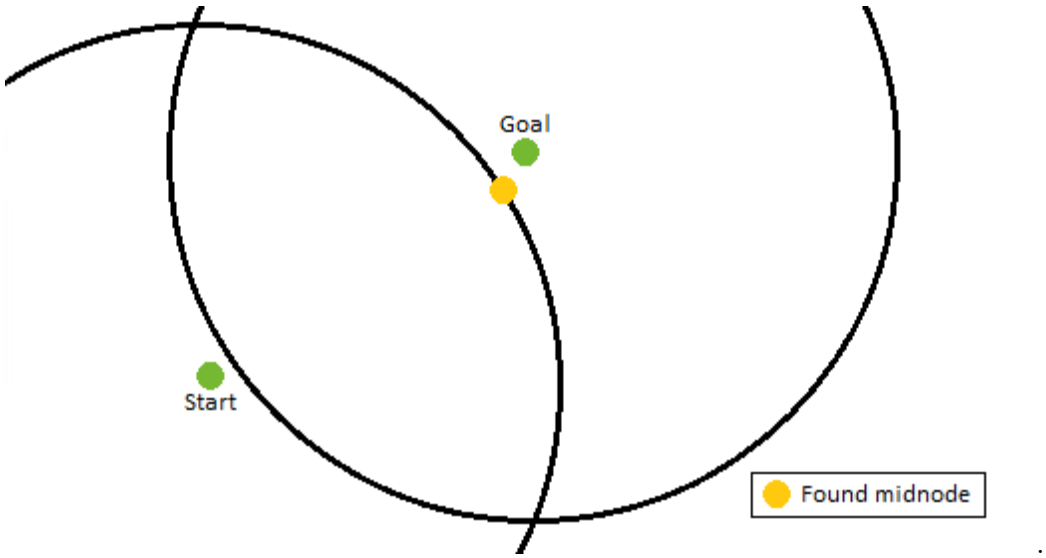


Figure 4.1: An illustration of large overlapping search spaces leading to large circle areas

Another modification making it possible to find multiple routes, was to not terminate when a route was found. Instead, the midpoint that was found was removed from the road network, and a new search was made. This resulted in alternative paths being found, which was a step in the right direction, but the alternative routes that were found often differed very little, see 4.2. The routes tended to differ mostly in the area where the two searches overlapped, leading to the alternative paths being identical close to the start and destination. This is likely not representative of the actual route options a driver might consider.

Figure 4.2 shows an example of how similar routes this method returns.

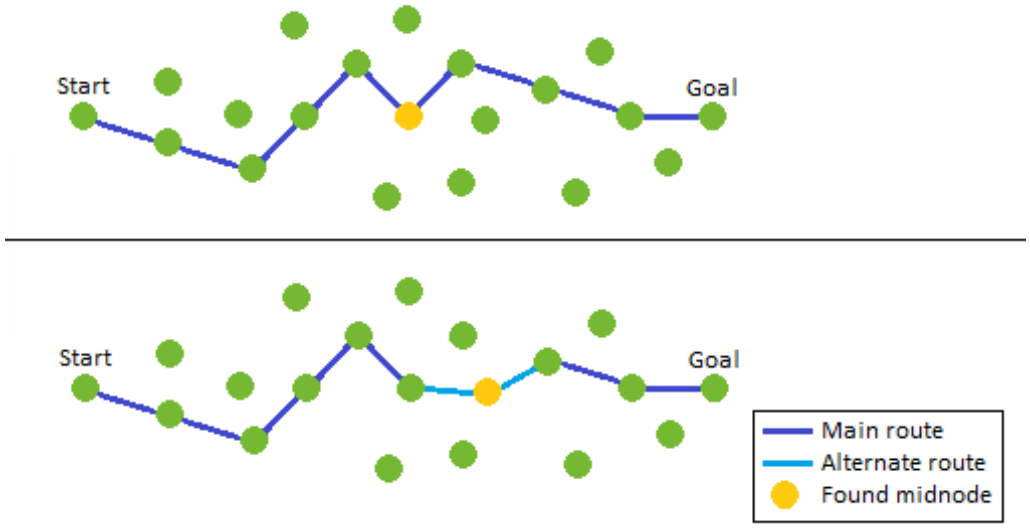


Figure 4.2: An illustration of a generated alternative route that barely differs from the main one

A new solution for finding multiple routes was implemented, where certain nodes along the already found routes were cut from the road network before making a new search. In essence, this is equivalent to the question; "If you're not allowed to drive on this road, which route would you choose instead?". By definition it's sufficient to remove only one node from the network to be guaranteed a new route with the next search. This is because if the node removed is included in the found route, it's now impossible to traverse this node and find the exact same route.

The first route found is, in practice, near enough always the fastest (assuming correct weighting of the graph). This can be called the main route. Nodes are cut from the main route one at a time, and for each node cut a new search is made. If the resulting alternative route does not fulfill our criteria for being a plausible option, another node is cut from the part of the alternative route not overlapping with the main route.

The algorithm was at this stage performing satisfactory, but the weighting of the road network was not yet fine-tuned. If the road network is weighted incorrectly, even a perfect algorithm would be unusable in practice. To make accurate predictions it's necessary to have a road network with weights that correspond to actual traversal times. The road network should be weighted by more than just distance. Additional parameters should influence an emergency vehicle's ability to traverse the road quickly. For practical use, the parameters must also be defined for every road.

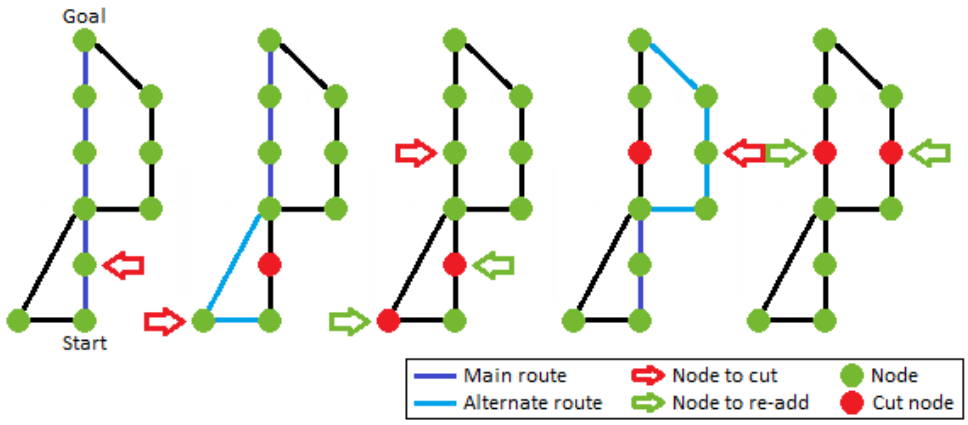


Figure 4.3: An illustration of nodes being cut and new routes found as a result.

To account for the time taken to traverse a road, each distance was multiplied with a *speed multiplier* based on the speed limit of the road. Since emergency vehicles can drive above the speed limit, some property indicating the ability to drive over the limit was also desired. The *road type* and corresponding *road type multiplier* adjusts for the size and type of road.

When dividing roads into different types, consideration was put into getting a division rightfully representing different abilities to speed over the limit. By the same principle, small country roads or bicycle lanes were assumed to negatively impact the driving speed, where narrow gaps or tight turns could slow down passage. Therefore, penalizing factors were defined for the roads of smaller size, and boosting factors were defined for large roads.

While running the algorithm on a road network grid, differences in preference between horizontal and vertical roads of the same length were noticed. This was an apparent problem, as it favored driving north/south rather than east/west. The reason for this discrepancy was that distance was at this point calculated using the euclidean formula on the geographical coordinates. A degree latitude was wrongfully assumed to draw an equally long line on the earth's surface as a degree longitude. The problem was mitigated after recomputing the coordinate-differences to actual distances.

As an emergency vehicle can drive against the rules of a normal car in an emergency, we wanted to add extra flexibility to the routing possibilities. At least in Gothenburg city, we recognized that emergency vehicles can drive in bus lanes & parkways for example. One of Carmenta's data providers did have data indicating which types of vehicles can drive on each road segment. Labels such as *pedestrian*, *car* and *emve* indicated if pedestrians, normal cars and emergency vehicles were allowed

on the road in question. A large bicycle road would presumably have the label combination $\{pedestrian = True, car = False, emve = True\}$ for example. This data was combined with the existing data set, leading to the final database structure as shown in table 3.3.

In certain cases we noticed strange behaviour, where routes would go off certain elevated highways like there were no height difference. If two LineStrings were defined using some shared exact points the search could not distinguish between a flat intersection and two roads crossing each other at different heights (such as an elevated highway passing above a road). Upon further thought this should not have been surprising, since the data set only contained two-dimensional *lat, lon*-geographical coordinates. See figure 4.4 for a visual explanation.

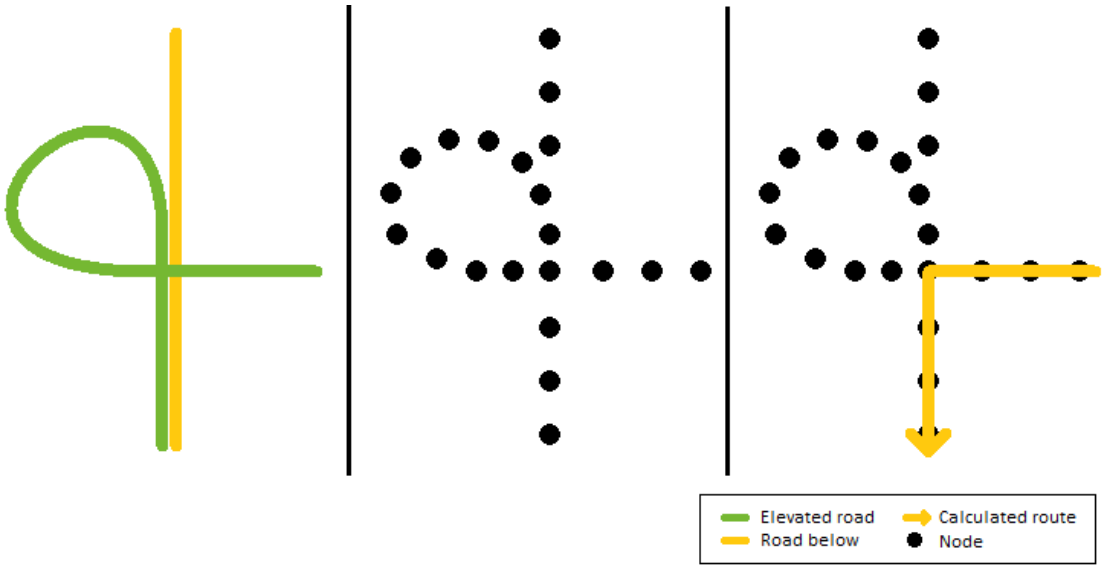


Figure 4.4: Illustration of how converting road LineStrings to individual nodes loses the implied height information.

As height information is non-existent, this problem is unsolvable with the current map data source. The temporary workaround used in the few cases found was to remove problematic nodes manually, meaning no routes could cross them at all, and therefore not causing this problem. It should be mentioned that height data is available but was not obtained due to time constraints, and the problem affected far from all test cases.

4.2 The final algorithm

Below the final algorithm is presented in pseudocode. Some operations are simplified, such as which nodes to cut, due to confidentiality requirements or irrelevance.

The implementation is fairly straight forward, however the performance of the algorithm is largely determined by the weighting of the road network and which nodes are chosen to be cut between iterations.

4.2.1 Helper classes

To make the algorithm more readable we use a helper class, here called SearchUtility. An instance of SearchUtility contains the data structures required to do an A* search, and since we will be searching from both the start and the end node we need two such Search Utilities.

```
Definition SearchUtility:
  queue := sorted queue of Node sorted by double
  dists := map of Node to distance from start/end
  visited := set of Node
  otherSearch := SearchUtility

Defintion GenerateSearchUtilities():
  searchUtilityStart := new SearchUtility
  add startNode to searchUtilityStart.queue with value 0.0
  searchUtilityStart.dists[startNode] = 0.0

  searchUtilityEnd := new SearchUtility
  add endNode to searchUtilityEnd.queue with value 0.0
  searchUtilityEnd.dists[endNode] = 0.0

  searchUtilityStart.otherSearch := searchUtilityEnd
  searchUtilityEnd.otherSearch := searchUtilityStart
  searchUtilities := {searchUtilityStart, searchUtilityEnd}
```

Pseudocode 4.1: Helper classes used to simplify the algorithm

4.2.2 Algorithm

```
minDist := positive infinity
nodesToCut := list of Node
routes := list of routes

do for desired number of different route options:
```

```
continueSearch := True
GenerateSearchUtilities()
while continueSearch:
  for each searchUtility in searchUtilities
    if length of searchUtility.queue > 0
      node := first Node in searchUtility.queue
      if node is in searchUtility.otherSearch.visited
        route := backtrack route from node to startNode and endNode
        dist := searchUtility.dists[node] +
                searchUtility.otherSearch.dists[node]
        if dist < minDist
          minDist = dist
        if this is the first route found
          nodesToCut := some selected nodes from route
        else
          cut a node from the part of the route that does not
          overlap with the shortest route
        uniqueDist := distance of the route that does not
        overlap with the shortest route
        quota := (dist - minDist) / uniqueDist
        if qouta < some threshold
          add route to routes
          re-add cut nodes
          cut new node from nodesToCut
        continueSearch := False
        break
      add node to searchUtility.visited
      for each neighbour to node
        newDist := searchUtility.dists[node] +
                   distance between node and neighbour
        add neighbour to searchUtility.queue with value newDist
        searchUtility.dists[neighbour] := newDist
    else if length of searchUtility.otherSearch.queue = 0
      No route could be found
      continueSearch := False
      break
```

```
return routes
```

Pseudocode 4.2: The final algorithm

The cutting of nodes is essential for producing multiple routes. The selection of nodes to cut happen in two different ways, one for selecting nodes from the primary route (the first route found), and another for nodes in alternative routes (all except the first route found).

From the primary route, only one node at a time is cut. In other words, the cut node is always re-added before a new node is cut. A selection of nodes to be cut is identified in the same iteration as the first route is found. There are many ways of selecting these nodes, an example would be to select every node at some specified interval, either absolute or relative to total route length. Another alternative would be to cut nodes directly after crossings, to force another route through intersections. In our implementation, the number of nodes selected in this step dictates the maximum number of alternative routes, since we stop searching for alternatives once a route that meets the threshold value is found. Instead of searching for more alternatives, the node cut from the primary route is re-added, and the next one is cut instead.

When an alternative route is found that does not meet the threshold value, an additional node is cut. To be certain of a new unique route in the next iteration, any node that is part of the alternative route but not a part of the primary route could be cut. Again there are several ways to select which specific node to cut, for example the first, last, or in the middle. In our implementation we continue to cut nodes from the alternative routes until either an alternative that meets the threshold value is found, or there are no more possible routes to find. When either of those things happen, the cut nodes are re-added, and a new search is started with the next node to be cut from the primary route.

4.3 Testing the software

Evaluating the functionality of software is not a trivial task, as it is the process of executing programs with the intent of finding unintentional errors made by the developers [20]. Software quality can not be tested directly, but instead related factors such as correctness and usability can be tested to indicate software quality in terms of functionality. The adaptability (future quality) of a program largely depends on the factors flexibility, reusability and maintainability, and is largely important for the program's future expansion, improvement and reliability [21].

4.3.1 Functionality

Hall and May (1997) presents three approaches to software functionality testing of which the most relevant for this project is *structural testing*. Structural testing is the practice of evaluating the coverage of elements in the structure or the specification of the program [22].

One form of structural testing is *specification-based structural testing*. Instead of testing the software against specific measurable performance criteria or pre-defined program flow-graphs, it is based on comparing program outputs with what can be considered to be correct according to its specification, which in turn suggests test adequacy [22]. Specification-based structural testing is suitable in this case due to the unquantifiable nature of its ground-truth correctness. Because the projects purpose is clear, to compute fast routes for emergency vehicles between two points in a city environment, the visualized program outputs can be evaluated and analyzed in terms of their adequacy for an emergency vehicle to take.

Path Coverage is a criterion in software testing that requires all execution paths of a programs flow graph's entry to its exit to be tested [22]. The purpose is to not leave execution paths untested, and therefore susceptible to unwanted behaviour. It does not guarantee correctness, since a limited set of tests has to be chosen for it to be practically useful in most programs [22]. The impracticality of path coverage applies to the path calculation algorithm developed in this project.

The software takes the same path through its flow graph and it is rather the calculation made by the route calculation algorithm that is to be tested for sufficiency. The path coverage criterion is therefore more suitably applied to a set of different routes for the software to calculate rather than program execution paths.

Four parameters have been identified as the key differentiators between routes. These are route length, road types, speed limit, as well as amount of segments traversed. To approximate algorithm path coverage, nine different emergency scenarios have been produced. The nine scenarios start at different hospitals, and the destinations contains the Gothenburg city center, as well as different neighbouring areas. The nine scenarios contains variations of all key differentiators, and can therefore as a group reasonably be expected to approximate path coverage of the algorithm. This, in turn, means that the performance shown in the test scenarios is expected to reasonably reflect the performance of the algorithm in most emergency events in the Gothenburg area.

4.3.2 Adaptability & Future quality

The typical software quality factors for adaptability & future quality are *flexibility*, *reusability* and *maintainability* [21].

The software developed in this project has had future quality in mind both during the design process, as well as during the actual development process. The software is divided into multiple classes, each with separate areas of responsibility. One example of this is the *RouteProbabilityServiceHandler*, which acts as a Controller and is responsible for delegating work and maintaining connection with the front-end as well as the message broker. Another example is the class *RoadMap*, which is responsible for calculating routes and building the road network data structure. Reusable code components should be self-contained and have clearly defined boundaries with respect to what their purposes are [23]. This structure of high cohesion and loose coupling was intentional, with the purpose of simplifying reusability and maintainability.

The program structure of a detached and isolated calculations microservice that listens to a separate stream of emergency vehicle position messages means that the main program is independent from and unaware of the origin of these messages. This allows emergency vehicle messages to be simulated during development, and the emergency vehicle simulator to be replaced with a live feed without additional modification of the main program.

It is a conflict of interests that the algorithm's performance is evaluated by its creators. This is suboptimal, and should be adressed by letting an independent third party evaluate the performance without input from the creators. But since this project is performed by the authors of this report, as well as in a limited time frame, diligence has been put into making fair evaluations to the best of our ability and to highlight advantages and disadvantages with the same regard.

4.4 Demonstration of execution

The performance of the route prediction algorithm was evaluated and verified using nine different synthetic scenarios. Real world data from SOS Alarm did not become available in time for the project deadline, therefore scenarios were constructed to be a realistic reflection of various destinations where an emergency vehicle might respond to an emergency. The locations are spread out across Gothenburg and all three major hospitals are included.

To the determine if the routes predicted by the algorithm are reasonable a number of soft criteria were used. For example; are there any obvious shortcuts missed, any unreasonable detours chosen, are large roads preferred? Combining this with our local knowledge of Gothenburg's road network it's possible to make statements about the algorithm's performance.

The biggest issue found in testing could be easily remedied with better underlying map data. It was however not possible to obtain because of the time frame, but it is readily available. This will be further explained under chapter 5. Discussion.

4.4.1 Sahlgrenska hospital to Långedrag

Figure 4.5: Sahlgrenska hospital to Långedrag

This scenario results in a both a main route and alternative routes that are reasonable. The predictions mostly go along large higher-speed roads, which are likely choices. Notice that the main route found is strikingly similar to a strictly shortest path. Some of the alternative routes are long detours from the main one (for example seen in the top right).

4.4.2 Sahlgrenska hospital to Eriksberg

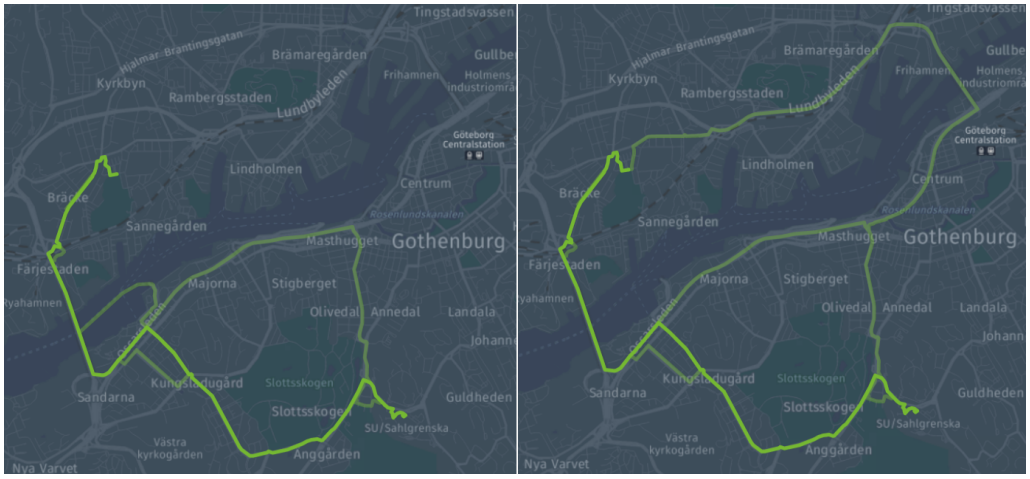


Figure 4.6: Sahlgrenska hospital to Eriksberg, before and after node removal

This scenario is an example of problems in the underlying map data. One of the alternative routes suggested includes first taking a ferry and then teleporting from the water to the top of the bridge Älvsborgsbron. This is obviously not possible and stems from the lack of height information in the data set. When a node is manually removed to prevent this, reasonable routes are suggested.

4.4.3 Sahlgrenska hospital to inner town

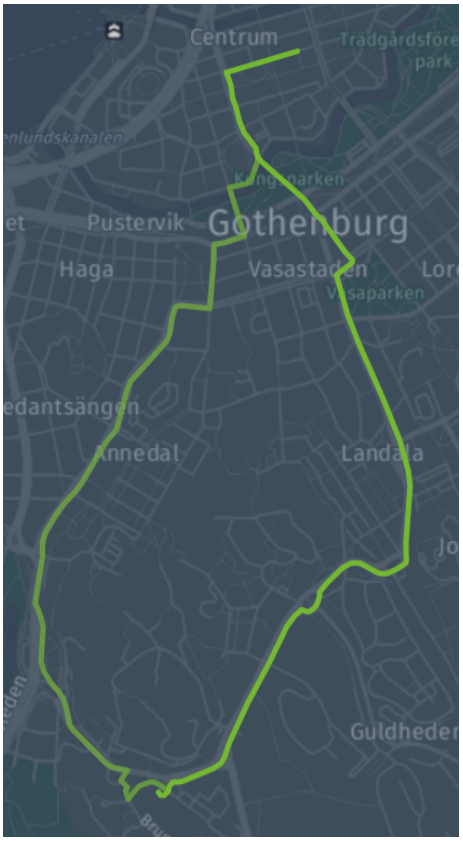


Figure 4.7: Sahlgrenska hospital to inner town

The routes predicted in this scenario are valid. The predictions are nearly equal in length, and travel mainly along high speed roads. Due to previous knowledge of possible traffic jams or long waiting times at red lights in the Vasastaden area, the alternative route could sensibly be preferred over the main route. Though this knowledge should not impact the prediction as a traffic jam would be reported as an incident in TrafficWatch, making the algorithm take this into consideration. An emergency vehicle can also run a red light.

4.4.4 Sahlgrenska hospital to Örgryte

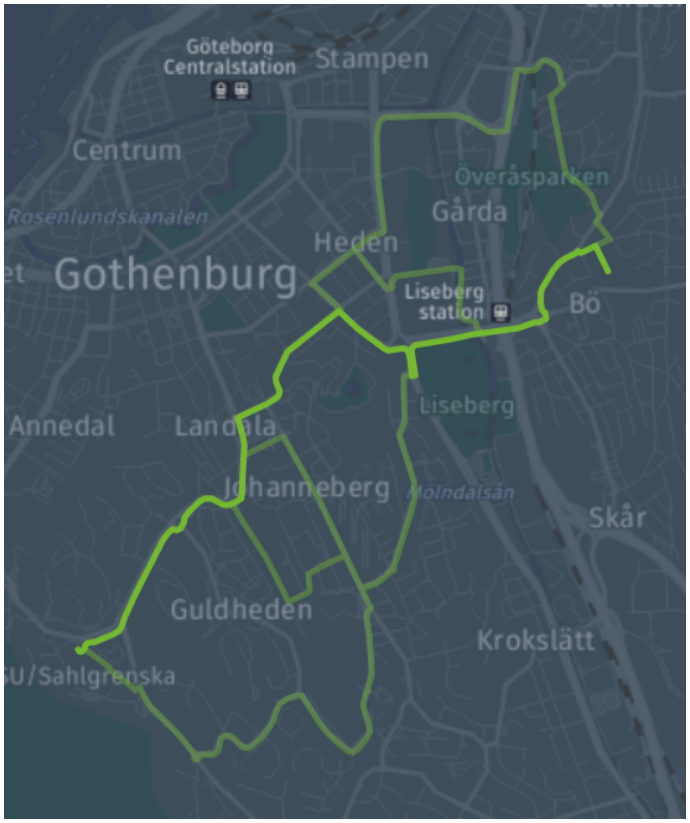


Figure 4.8: Sahlgrenska hospital to Örgryte

Many alternative routes are shown. Both the topmost and lowermost alternative routes are long detours. Their speed could be similar, as the results indicate, but these are considered unlikely and should probably have been filtered out.

4.4.5 Sahlgrenska hospital to Vasastaden

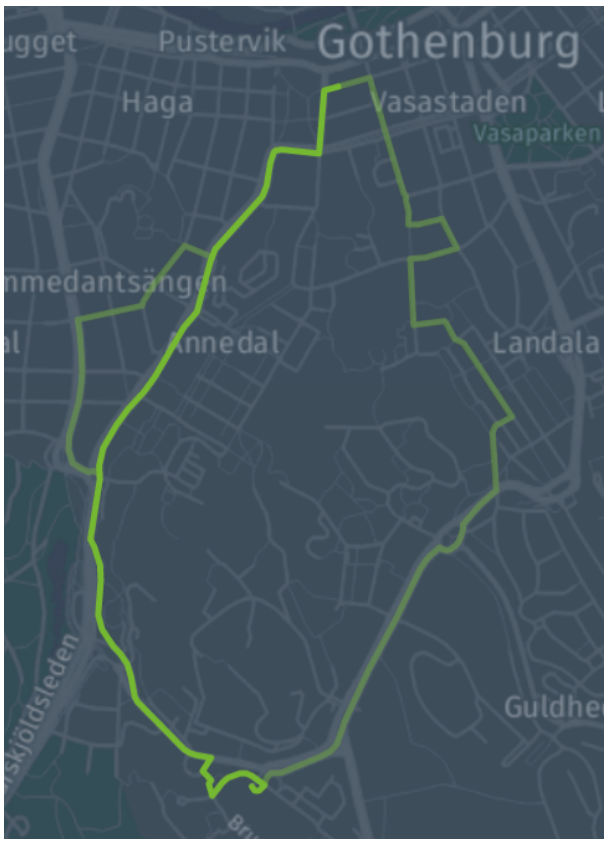


Figure 4.9: Sahlgrenska hospital to Vasastaden

The main route and full alternative route predictions are the only reasonable routes. The extra alternative turn that goes around Annedal is an unlikely choice when the main route's road continues and goes straight towards the goal.

4.4.6 Östra hosptial to Sävedalen

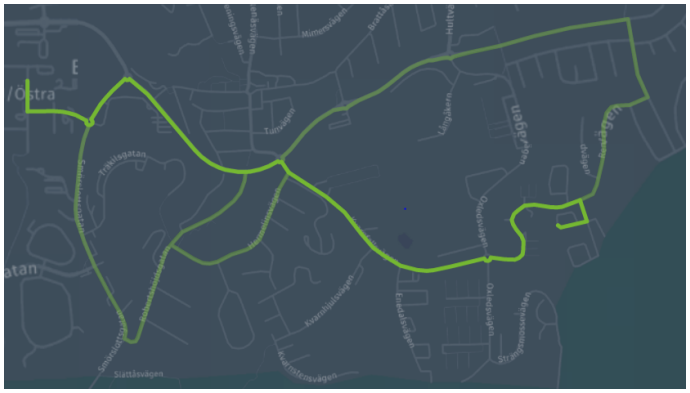


Figure 4.10: Östra hospital to Sävedalen

The primary route is likely the best one. The alternatives routes are fairly long detours in comparison.

4.4.7 Sahlgrenska hospital to Majorna

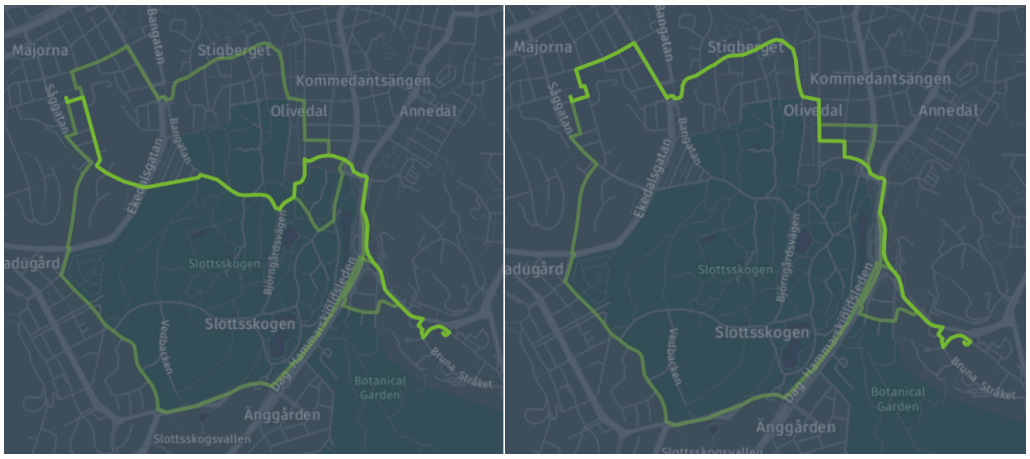


Figure 4.11: Sahlgrenska hospital to Majorna, before and after node removal.

This scenario shows how the underlying map data is essential for good route predictions. The primary route suggested goes straight through Slottsskogen park, which is technically allowed but highly unlikely to be chosen due to both pedestrians and road barriers. When a node is manually removed to prevent this specific route, the primary route instead becomes one of the alternative, more realistic, routes. This indicated that the penalty for driving on roads where cars are not allowed should possible be higher.

4.4.8 Mölndal hospital to Rävekärr

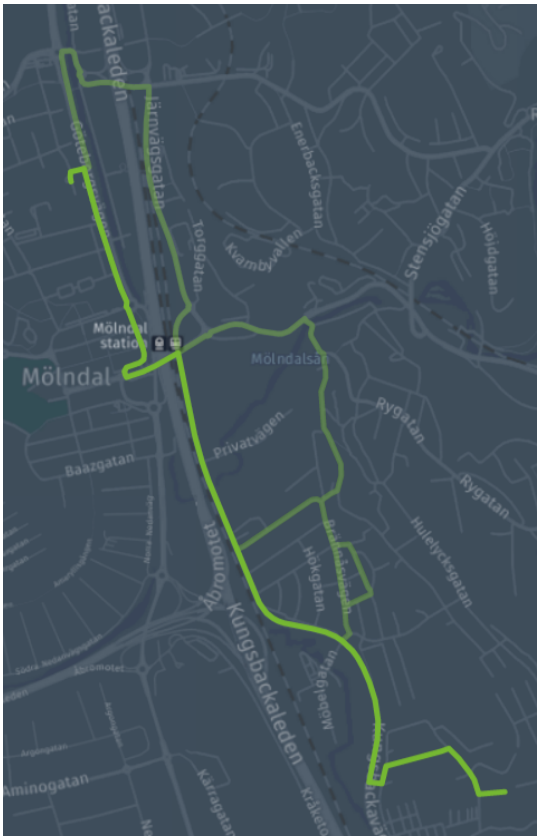


Figure 4.12: Mölndal hospital to Rävekärr

The primary route is a good choice, but once again we see that some alternative routes could be considered to be to much of a detour to be realistic options.

4.4.9 Mölndal hospital to Tolltorpsdalen

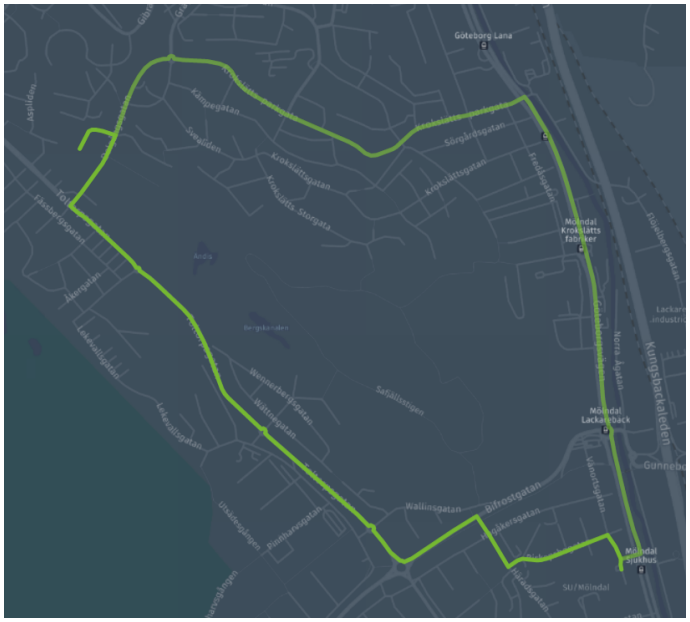


Figure 4.13: Mölndal hosptital to Tolltorpsdalen

In this scenario, the only two feasible route options are correctly identified and the fastest is chosen as the primary one.

4.5 Performance

Generally, the primary route identified is the fastest one. Alternative routes are often feasible options, however some could be argued to be to long detours. This suggests that the threshold for accepting alternative routes as a feasible option needs to be altered. Choosing the threshold, and the calculation used for producing the statistic to compare with, is not trivial and there is still room for improvement in this area.

4.6 Utility

The algorithm generally identifies the fastest route, which does contain the best area in which to send preemptive warnings. Though, since the manual intervention is needed in certain cases, such as manually removing nodes when the underlying data set is imperfect, the software at its current state as well as the current data source needs further refinement before being ready for production deployment.

5

Discussion

The map data sourced from one of Carmenta's existing data providers was imperfect. This was demonstrated during the test scenarios, where some impossible emergency vehicle paths were predicted, such as driving off of elevated highway roads. This problem, along with most problems highlighted in the test scenarios are due to imperfect map data. The missing information to solve this is height data for each recorded coordinate. If height data existed either fully or just as an approximation, it could be used to only allow possible ways to turn in each node, solving the teleportation problems.

In certain cases the data was not incorrect, but rather incomplete. In section 4.6 a route recommendation contains a ferry ride over the water. Technically an emergency vehicle would have been allowed on the ferry, and since the distance is short the scoring function considered this a feasible alternative despite the ferry having a slow speed limit. Situations like this does not indicate the algorithm needs alteration, but rather that the data needs updating so that unreasonable "roads" such as ferry rides are excluded. Taking a ferry ride would undeniably lead to a substantial amount of overhead both in boarding, leaving land and docking at the destination. And this is assuming the optimal case where the ferry leaves immediately at emergency vehicle arrival.

In a production setting it's possible that a better approach to finding alternate routes would be to cut more nodes close to the emergency vehicles current location and fewer far away. This is because it's really only interesting to know where the emergency vehicle might travel in the very near future. Since it's possible to re-evaluate route predictions as new location updates from the emergency vehicles become available it might not be necessary to find alternative routes far away from the emergency vehicles current position.

An interesting extension to the algorithm which was excluded due to time limitations was to incorporate penalties for turning. Not only is there time overhead implied in each turn an emergency vehicle makes due to deceleration, acceleration, and

cautiousness by the driver. Turning an emergency vehicle also spawns tangible safety hazards for unstrapped emergency vehicle staff inside an ambulance, and also to a potential physically harmed patient. Due to this, it is not unreasonable to think that an emergency vehicle driver would prefer to stay on straighter roads. A turning penalty could be incorporated in our software by penalising the amount of unique roads in a path. This value will in practice be equal to "the amount of turns made". A critical component in making this result in more accurate predictions in practice is to find the suitable penalty for a turn.

Graphhopper [24] is an open-source routing algorithm that has use cases similar to the algorithm developed in this thesis. It supports multiple routing algorithms including Dijkstra's as well as another variant of the bidirectional A* algorithm. Being a much more complete and optimized routing engine with support for mobile navigation clients and other nice-to-haves, it is in a production state. The A* algorithm that Graphhopper uses has many similarities to our algorithm, but differs in for example the weighting of the network. Interestingly, Graphhopper incorporates a type of turning penalty like proposed above, called turn weight. It is a weight addition that approximates the time a turn adds, with values stored for edges and nodes in a *turnCostStorage*. Because of the large amount of factors impacting the time a single turn will add to a route, the approximations will likely never be quite right. A thorough comparison between arbitrary score boosts multiplied by the amount of unique roads like proposed by us, the turn weight approach used by Grasshopper, and real driving data would be interesting.

We have found few established approaches and existing articles regarding the problem of finding alternative paths, or multiple shortest paths using the A* algorithm. It is established that the algorithm can be extended to find the N shortest paths, but debated whether it can be extended to do so in polynomial time. This could explain the lack of work using an A*-algorithm to find multiple paths. The approach taken in this thesis is limited in terms of computation time by having a low fixed threshold of node-cutting and re-searching iterations between terminating and cancelling the search for alternatives. This makes it a feasible alternative for real-world use regardless of time complexity.

The natural next step for this project is to validate the algorithm outputs against real world data. It's possible that the real world performance is sufficient, but changes might need to be implemented. The first thing to alter would likely be the weighting of the road network, since this is what determines which routes the algorithm deems fast. The goal of the weighting is to be an analog to the time required to travel

the corresponding route in the real world, since short time to the destination is the number one priority for emergency vehicles. There might be other parameters with great effect on the time taken that have not been considered in this project.

Our assessment is that the necessary infrastructure for practical implementation of sending preemptive emergency vehicle warnings is in place. GPS coordinates are accurate enough for determining emergency vehicle positions. The wireless connectivity standards used today is sufficient in speed and latency for deploying an application like this, since the data sent (warnings and positions) do not require substantial bandwidth. The warnings are to be sent 10-15 seconds in advance, meaning no specific requirements for latency exists.

6

Conclusion

The results indicate that probable emergency vehicle routes can be predicted using a modified bidirectional A*-algorithm. The problems encountered originate from a bad data source, rather than the algorithm design. The performance demonstrated in 9 varied test cases indicate that the algorithm performs sufficiently well to generally identify the most probable path, containing the road segment to send warnings along. The problems demonstrated can be remedied using a better data source which is readily available but were not included in this project due to time limitations.

For this to be confirmed, validation against real emergency scenarios needs to be performed, and the accuracy of warned areas needs to be low enough to comply with the results of VTI's research of acceptable false-positive rates for driver warnings, so that trust is established in the system and it can fill its intended purpose.

Bibliography

[1] Anna Johansson Jacques. *Emergency Vehicle Warnings*. Swedish Transport Administration. URL: https://www.nordicway.net/flagship/emergency-vehicles (visited on 04/26/2022).

[2] Carmenta Group. *SE NordicWay Emergency Vehicle Approaching C ITS Service*. 2019. URL: https://www.youtube.com/watch?v=gqN2aABeOPs.

[3] Kristian Jaldemark. Private Communication. Gothenburg, 03, SE, 2022.

[4] Louise Olsson. *Färdplan - digitaliserat vägrtransportsystem version år 2022 (kort version)*. Trafikverket. Mar. 18, 2022. DOI: http://trafikverket.diva-portal.org/smash/get/diva2:1651949/FULLTEXT01.pdf.

[5] John R. Herring. "OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture". Version 1.2.1. In: (May 2011).

[6] Howard Butler et al. "The GeoJSON Format". In: (2016). DOI: https://www.rfc-editor.org/rfc/pdfrfc/rfc7946.txt.pdf.

[7] Tom Proctor. *Polygonal Chain*. URL: http://wiki.gis.com/wiki/index.php/Polygonal_chain (visited on 03/02/2022).

[8] Riccardo Di Sipio. "Quick Guide to Graph Traversal Analysis". In: (). URL: https://towardsdatascience.com/quick-guide-to-graph-traversal-analysis-1d510a5d05b5 (visited on 05/11/2021).

[9] Thomas H. Cormen et al. *Introduction to Algorithms*. MIT Press, July 2009.

[10] Paul E. Black. *Greedy algorithm*. Feb. 2005.

[11] Jørgen Bang-Jensen, Gregory Gutin, and Anders Yeo. "When the greedy algorithm fails". In: (2004). DOI: https://doi.org/10.1016/j.disopt.2004.03.007.

[12] Amit Patel. *Introduction to A**. Stanford University. URL: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (visited on 03/08/2022).

[13] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[14] W. Zeng and R.L. CHURCH. “Finding shortest paths on real road networks: the case for A*”. In: *International Journal of Geographical Information Science* (Dec. 12, 2007). DOI: https://doi.org/10.1080\%2F13658810801949850.

[15] Amit Patel. *Variants of A**. Stanford University. URL: http://theory.stanford.edu/~amitp/GameProgramming/Variations.html (visited on 03/08/2022).

[16] Chris Veness. *Calculate distance, bearing and more between Latitude/Longitude points*. Movable Type Ltd. URL: https://www.movable-type.co.uk/scripts/latlong.html (visited on 03/08/2022).

[17] David R. Williams. *Earth fact sheet*. National Aeronautics and Space Administration. URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html (visited on 03/08/2022).

[18] František Duchoň et al. “Path Planning with Modified a Star Algorithm for a Mobile Robot”. In: *Procedia Engineering* 96 (Dec. 27, 2014).

[19] Chunbao Wang et al. “Path planning of automated guided vehicles based on improved A-Star algorithm”. In: *2015 IEEE International Conference on Information and Automation*. 2015, pp. 2071–2076. DOI: 10.1109/ICInfA.2015.7279630.

[20] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley Sons, Sept. 2011.

[21] Jiantao Pan. “Software Testing”. In: *18-849b Dependable Embedded Systems* (1999). DOI: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/software-testing.pdf.

[22] Patrick A. V. Hall and John H. R. May. “Software Unit Test Coverage and Adequacy”. In: *ACM Computing Surveys, Vol. 29, N. 4* (Dec. 1997).

[23] B. Jalender B., A. Govardhan, and P. Premchand. “DESIGNING CODE LEVEL REUSABLE SOFTWARE COMPONENTS”. In: *International Journal of Software Engineering Applications (IJSEA), Vol.3, No.1* (Jan. 2012). DOI: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1069.6383&rep=rep1&type=pdf.

[24] Peter karussell. *Graphhopper*. https://github.com/graphhopper/graphhopper. 2022.

A

Appendix 1 - Batched road database creation query

MSSQL Query

```
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name = 'RoadSegments' AND xtype = 'U')
        CREATE TABLE RoadSegments (
        road_id INT NOT NULL,
        fromLat FLOAT NOT NULL,
        fromLon FLOAT NOT NULL,
        toLat FLOAT NOT NULL,
        toLon FLOAT NOT NULL,
        dist FLOAT NOT NULL,
        speed INT NOT NULL,
        size INT NOT NULL,
        car NVARCHAR(1),
        emve NVARCHAR(1),
        PRIMARY KEY (fromLat, fromLon, toLat, toLon));

DECLARE @batches INT = (SELECT COUNT(*) FROM dbo.RoadNetworkWithNodes) / 2500;
DECLARE @batch_loop INT = 1;
DECLARE @Geoms TABLE(road_id INT, geom GEOMETRY, speed INT,
  size INT, dir NVARCHAR(1), car NVARCHAR(1), emve NVARCHAR(1), batch INT);
DECLARE @Geoms2 TABLE(road_id INT, geom GEOMETRY, speed INT,
  size INT, dir NVARCHAR(1), car NVARCHAR(1), emve NVARCHAR(1), rownr INT);
DECLARE @rows INT;
DECLARE @row_loop INT;
DECLARE @line GEOMETRY;
DECLARE @points INT;
DECLARE @point_loop INT;
DECLARE @point1 GEOMETRY;
DECLARE @point2 GEOMETRY;
DECLARE @road_id INT;
DECLARE @speed INT;
DECLARE @size INT;
```

```
DECLARE @dir NVARCHAR(1);
DECLARE @car NVARCHAR(1);
DECLARE @emve NVARCHAR(1);
DECLARE @dist FLOAT;

INSERT INTO @Geoms(road_id, geom, speed, size, dir, car, emve, batch)
SELECT LINK_ID, GEOM, SPEED_CAT, FUNC_CLASS, DIR_TRAVEL, AR_AUTO, AR_EMVE,
  NTILE(@batches) OVER(ORDER BY (SELECT 0)) AS batch FROM dbo.RoadNetworkWithNodes;

WHILE @batch_loop <= @batches
  BEGIN
  INSERT INTO @Geoms2(road_id, geom, speed, size, dir, car, emve, rownr)
  SELECT road_id, geom, speed, size, dir, car, emve, ROW_NUMBER() OVER(ORDER BY (SELECT 0))
    FROM @Geoms WHERE batch = @batch_loop;
  SET @rows = (SELECT MAX(rownr) FROM @Geoms2);
  SET @row_loop = 1;
  WHILE @row_loop <= @rows
    BEGIN
    SELECT @line = geom, @road_id = road_id, @speed = speed, @size = size, @dir = dir,
      @car = car, @emve = emve FROM @Geoms2 WHERE rownr = @row_loop;
    SET @points = @line.STNumPoints();
    SET @point_loop = 2;
    WHILE @point_loop <= @points
      BEGIN
      SET @point1 = @line.STPointN(@point_loop-1);
      SET @point2 = @line.STPointN(@point_loop);
      SET @dist = SQRT(SQUARE(@point1.STY-@point2.STY)+SQUARE(@point1.STX-@point2.STX));
      IF (@dir = 'B')
        BEGIN
        INSERT INTO dbo.RoadSegments(road_id, fromLat, fromLon, toLat, toLon, dist,
          speed, size, car, emve) VALUES (@road_id, @point1.STY, @point1.STX, @point2.STY,
          @point2.STX, @dist, @speed, @size, @car, @emve);
        INSERT INTO dbo.RoadSegments(road_id, fromLat, fromLon, toLat, toLon, dist,
          speed, size, car, emve) VALUES (@road_id, @point2.STY, @point2.STX, @point1.STY,
          @point1.STX, @dist, @speed, @size, @car, @emve);
        END
      ELSE IF (@dir = 'F')
        INSERT INTO dbo.RoadSegments(road_id, fromLat, fromLon, toLat, toLon, dist,
          speed, size, car, emve) VALUES (@road_id, @point1.STY, @point1.STX, @point2.STY,
          @point2.STX, @dist, @speed, @size, @car, @emve);
      ELSE
        INSERT INTO dbo.RoadSegments(road_id, fromLat, fromLon, toLat, toLon, dist,
          speed, size, car, emve) VALUES (@road_id, @point2.STY, @point2.STX, @point1.STY,
          @point1.STX, @dist, @speed, @size, @car, @emve);
      SET @point_loop = @point_loop + 1;
```

```
76        END;
77      SET @row_loop = @row_loop + 1;
78      END;
79    SET @batch_loop = @batch_loop + 1;
80    DELETE FROM @Geoms2 WHERE 0 = 0;
81    END;
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY