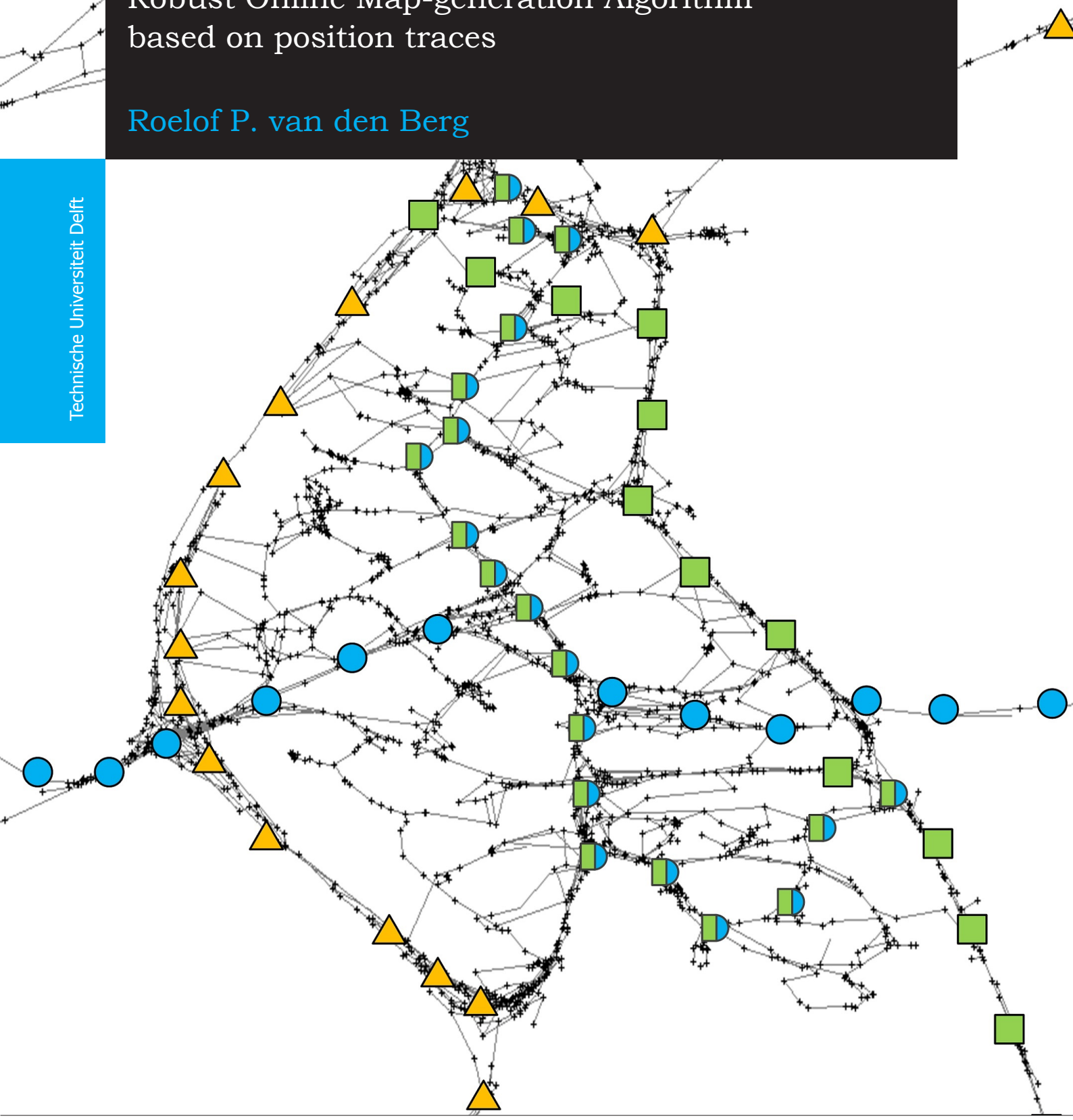
All traces lead to ROMA

Design and evaluation of a Robust Online Map-generation Algorithm based on position traces

Roelof P. van den Berg

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

All traces lead to ROMA

Design and evaluation of a Robust Online Map-generation Algorithm based on position traces

by

Roelof P. van den Berg

in partial fulfilment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended on Friday March 6, 2015 at 2:00 PM.

Supervisor:	Prof. dr. C. M. Jonker,	TU Delft
Thesis committee:	Prof. drs. dr. L. J. M. Rothkrantz,	TU Delft
	Dr. M. C. A. van der Sanden	TU Delft
	F. van Polen, MSc,	Sense Observation Systems

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

Preface

This thesis is the result of my graduation project for my masters programme for Computer Science at the Interactive Intelligence research group of Delft University of Technology. This makes for a double degree in combination with the Science Communication track of the masters programme Science Education and Communication at the same university. The research presented in this thesis was performed as part of an internship at Sense Observation Systems in Rotterdam where I also did my graduation research for Science Education and Communication.

This thesis is primarily intended for researchers in computer science and geographical information systems. For those interested in the developed algorithm for dynamic map-generation, I suggest reading chapter 4. A novel approach to the quantitative comparison of vector-maps is proposed in chapter 3. Reflection on both methods and their generalisability is given in chapter 7. A more elaborate outline to this thesis is given in section 1.7.

I would like to express my gratitude to prof. dr. Catholijn Jonker, my primary supervisor, for her guidance through the process and the ever-present enthusiasm for my research and future work. Besides my primary supervisor, I would like to thank the rest of the thesis committee and my supervisors: Prof. drs. dr. Léon Rothkrantz, dr. Maarten van der Sanden, and Freek van Polen MSc, for their encouragement, constructive criticism on my research, and for supporting me in my first steps as an independent researcher. My sincere thanks also goes to the people at Sense, for their contribution to this project. Especially the opportunity to perform 'field research' on JP's car was an adventure.

Last but not least, I would like to thank my family, friends, and fellow students for the endless support and numerous discussions. A special thank you to Sietske whose help in developing all the figures was of great importance to the readability of this report. Thank you all for driving along and discovering new roads.

Roelof P. van den Berg
Delft, February 2015

Summary

Keeping maps up to date is quite costly because road geometry changes over time and mapping by professional surveyors is an expensive operation. With the presence of GPS sensors on mobile phones and in most cars today it becomes possible to measure the location of roads with many measurements from sensors of lower accuracy. In this thesis we explored a method to use these less-accurate measurements for the development and maintenance of a dynamic road map. The main research question is:

How to create and maintain
an accurate and dynamic map
based on position traces?

Method

To answer the main research question, we have developed a quadratic time robust online map-generation algorithm (ROMA) for creation a dynamic vector-representation of the world where position signals are gathered. The world and GPS signals were simulated so that an absolute ground truth representation off the world was available. For the evaluation of map quality we developed a cubic time metric which finds a common edge subgraph of the world and the developed map. This subgraph indicates the true positives and enabled us to determine both precision and recall for the generated map. Using the harmonic mean of precision and recall we were able to indicate map quality in one number on the interval [0,1]. We tested the dynamic behaviour of ROMA on scenarios of road introduction and road removal.

Results

Each run of the road introduction experiment showed an increase in map quality followed by stabilisation or even a decline in map quality. Map quality generally reached a value of 0.5 within 5 minutes of road introduction. In all experiments the recall was constantly higher than the measured precision. Recall generally reached a maximum of over 0.8 whereas precision stayed at 0.5. For the road removal experiment we introduced roadblocks at the peak of measured map quality for traffic densities showing a clear decline afterwards. Road removal was detected within 30 minutes after the introduction of roadblocks for removal of up to 50% of all roads in the world. All experiments showed a build up of web-like road structures around the location of roads in the world. This build-up was stronger present in experiments with higher traffic densities.

Discussion and conclusion

This research has provided new insight into incremental map-development by looking at the temporal aspects of the process. Within this research we have provided insight in the formation over time of web-like structures within trace-merging algorithms. Due to the investigation of this phenomenon we have also provided suggestions for the decrease of this effect over time by merging parallel roads. The novel graph based method for map comparison offers advantages to existing approaches and is especially suitable for incremental maps while the road takes shape.

ROMA offers a method for dynamic map generation which robustly creates and maintains a map representation of the world through the collection of GPS traces. The experiments show that a navigable map is generated which detects new roads being travelled and also forgets roads no longer travelled. We therefore conclude that ROMA is a decent proof of concept for dynamic vector-map generation in an online fashion.

Samenvatting

Het up to date houden van kaarten is redelijk kostbaar omdat de geometrie van wegen over tijd verandert en het maken van kaarten door professionele landmeters erg kostbaar is. Met de opkomst van GPS-sensoren op mobiele telefoons en in de meeste auto's wordt het mogelijk om de locatie van de weg te meten met behulp van veel metingen van een lagere nauwkeurigheid. In dit verslag verkenden wij een methode om door middel van deze metingen een dynamische wegenkaart te bouwen en onderhouden. De hoofdvraag van het onderzoek luidt:

Hoe creëer en onderhoud je een accurate
en dynamische kaart op basis van positie-traces?

Methode

Om de hoofdvraag te beantwoorden hebben we een robuust online map-generatie algorithme (ROMA) met kwadratische complexiteit ontwikkeld voor het maken van een dynamische vector-representatie van de wereld waarop positiedata verzameld wordt. De wereld en GPS-signalen zijn gesimuleerd om zo een objectieve waarheid van de wereld te bewerkstelligen. Voor het meten van de kaartkwaliteit hebben we een metriek met derdegraads complexiteit ontwikkeld welke een *common edge subgraph* vindt van de wereld en de ontwikkelde kaart. Deze subgraaf geeft een indicatie van de *true positives* en maakte dat wij zowel precisie als recall kunnen bepalen voor de geproduceerde kaart. Met behulp van het harmonisch gemiddelde van de precisie en recall verkregen wij een kwaliteitsmaat voor de kaart op het interval [0,1]. Wij hebben het dynamische gedrag van ROMA getest op scenario's met zowel introductie als ook verwijdering van wegen.

Resultaten

Elke run van het experiment met introductie van wegen toonde een toename in kaartkwaliteit gevolgd door stabilisatie of zelf afname van deze kwaliteit. De kaartkwaliteit bereikte over het algemeen een waarde van 0.5 binnen 5 minuten na de introductie van wegen. In alle experimenten was de recall consistent hoger dan de gemeten precisie. Recall bereikte over het algemeen een maximumwaarde van meer dan 0.8 terwijl de precisie bleef steken op 0.5. Voor het experiment met verwijdering van wegen hebben wij blokkades toegevoegd op het moment van maximale kaartkwaliteit voor verkeersdichtheden waarbij afname in kwaliteit zichtbaar was. Binnen 30 minuten na toevoeging van blokkades werden deze gedetecteerd bij het verwijderen van tot aan 50% van alle wegen in de wereld. Alle experimenten toonden een opbouw van web-achtige wegstructuren rondom de ligging van wegen in de wereld. Deze opbouw was sterker aanwezig in experimenten met hogere verkeersdichtheid.

Discussie en conclusie

Dit onderzoek heeft nieuwe inzichten opgeleverd in incrementele kaartontwikkeling door te kijken naar de temporele aspecten in het proces. In dit onderzoek hebben wij inzichten gegeven in de vorming over tijd van web-achtige structuren binnen *trace-merging* algorithmes. Door het onderzoek naar dit verschijnsel hebben we tevens suggesties verstrekt voor de vermindering van dit effect in de tijd door parallelle wegen samen te voegen. De nieuwe graafgebaseerde methode voor kaartvergelijking biedt voordelen ten opzichte van bestaande methodes en is in het bijzonder geschikt voor incrementele kaarten waarin de wegvorm nog ontwikkelt.

ROMA biedt een methode voor dynamische kaartgeneratie welke op robuuste wijze een kaart van de wereld maakt en onderhoudt door het verzamelen van GPS-traces. De experimenten tonen aan dat een navigeerbare kaart gemaakt wordt welke nieuwe wegen herkent en tevens niet langer bereden wegen vergeet. Wij concluderen daarom dat ROMA een proof of concept is voor het op incrementele wijze maken van dynamische vectorkaarten.

Contents

1

Introduction

The process of creating a map by surveyors or from aerial images is quite complex and makes that most available digital maps are expensive to produce and update. Next to the complexity it is also quite costly to keep these maps up to date because road geometry changes over time [5]. With the availability of low-cost positioning devices and the development of wireless communication systems, integrated data gathering and processing become feasible at a large scale. These methods are able to continuously update the map as new information becomes available [1, 3, 25]. The direct motivation for the research in this thesis is found in disaster scenarios where a priori maps are no longer accurate.

This thesis presents a proof of concept for a robust online map-generation algorithm (ROMA) applicable to situations without a priori maps and the evaluation of this algorithm. The research problem is described in section 1.1 followed by the research goal for this thesis in section 1.2. The formalisation of the research problem and goal is given in section 1.3 where the research questions are defined. We approach the research questions using the methodology described in section 1.4. The relevance of this research to both theory and practice is given in section 1.5, and the focus of this research is explained in section 1.6. The chapter ends with the outline of the report in section 1.7.

1.1. Problem description

Position measurements provide the information needed to create and maintain a dynamic map of a previously unmapped environment. Together with modern wireless communication we are able to integrate these measurements into an existing map. Through the abundance of data, the inaccuracy of cheap positioning devices can be compensated. The resulting roadmap contains more detail and is more accurate compared to maps created by traditional methods such as satellite imagery and specialized surveillance. Furthermore, these methods are able to continuously update the map as new information becomes available [1, 25].

Methods of determining the road network from a multitude of position traces have been researched. However, most of this research uses a one-time processing of all traces and keeps the created topology intact over time. In a disaster relief scenario where a map representing the current situation should be provided as soon as possible, an on-line method would be better suited since it provides a best guess of the observed world at any time. If such a map is updated in real-time, rescue-workers can follow detected paths without having to spend valuable time searching for them. Maintaining an up-to-date map also increases the situation awareness of the rescue-workers.

The success of a map for a dynamic and hazardous environment has two essential elements: spatial accuracy and temporal accuracy. The first, spatial accuracy makes sure that vehicles using the map for navigation can be placed on the map as best as possible. Many authors [5, 11, 23, 24] state that combination of measurements from low-precision data sources provides equal or even of better spatial accuracy compared to the expensive use of a high-precision data source as used by professional surveyors.

The second element, temporal accuracy makes sure that the road representation best represents the actual road layout. Because subsequent events in a disaster area can make previously available paths unavailable, we wish to research how to deal with changes in road layout. An answer to the question of how to determine whether or not a road still exists has not been found in literature but is relevant for the case of exploration in a disaster scenario.

1.2. Research goal

It is needed for modern road maps to deal with the temporal aspects of their content. This should be done by 1) dealing with changing road shapes where the shape should best represent the actual situation, but also be as spatially accurate as possible; and 2) detecting new roads and removing roads which no longer exist. These two elements have not been researched to a great extend and will therefore be new additions to research on incremental ad-hoc mapping. The goal of this research is therefore to provide a proof of concept for a dynamic road map which incorporates the temporal and spatial aspects needed when no a priori map is present and both topology and geography of the map may change over time.

1.3. Research question

The problem description in section 1.1 states that the temporal aspect of a dynamic road map in terms of topology and geography are less researched. It is however needed to attend to these temporal aspects to develop a map useful for a changing environment where a priori knowledge is missing and a best guess is better than no map. The research question based on this setting is defined as:

How to create and maintain
an accurate and dynamic map
based on position traces?

We state several sub questions to aid in answering the main research question. Having an accurate map both temporally and spatially is essential for the use of such a map in a changing environment. Since we want to see if the developed solution is an accurate representation of the world, we therefore ask:

SQ1: How to measure spatial and temporal accuracy of a map?

After an answer for this question is obtained, we can come to develop an algorithm which is capable of creating such a spatial and temporally accurate map. The second sub question is thus:

SQ2: How to create a spatial and temporally accurate map?

With the developed proof-of-concept map-generation algorithm, we can then test the performance of the algorithm and find out how the characteristics in terms of spatial and temporal accuracy are. This allows us to answer the following question:

SQ3: How spatially and temporally accurate can a dynamic map be?

The developed algorithm might be influenced by external factors and we would like to know how the performance of the algorithm can be influenced. We therefore define the following question:

SQ4: How do spatial and temporal accuracy correlate ?

Given this thorough examination of the algorithm's performance under different circumstances and experience with the subject matter, we finally pose the last sub question as:

SQ5: Which domains can benefit from the developed method for dynamic mapping?

1.4. Methodology

Information on automated mapping in relation with spatial and temporal accuracy will be sought in literature and used to build a method which can be used to create a spatially and temporally accurate map. Because no graph based approach to map comparison is known, such a method will also be developed based on what has been found in literature.

Because the research focuses on the spatial and temporal accuracy of the map derived from position traces, modules in the prototype dealing with temporal accuracy are evaluated more in depth. Multiple solutions for obtaining spatial and temporal accuracy are tested and compared. For other modules, a method will be used from previous research wherever possible.

To measure the quality of the generated map, an absolute ground truth is needed. We therefore use a simulator to simulate traffic on a known ground truth map and generate GPS-position measurements. The simulator is developed in a modular fashion to accommodate comparisons with other developed methods. The developed method for comparison of two vector-maps will be used to compare the ground truth and generated map with one another.

Two experiments are run to test the algorithm for dealing with new roads, and with the removal of roads from the ground truth. The experimental design with relations between independent and dependent variables researched in this thesis are shown in fig. 1.1. For these experiments, multiple traffic intensities will be used in both experiments as independent variable. In the second experiment, the amount of roads removed might influence the performance of the algorithm, and is therefore tested as independent variable. After introducing changes in the ground truth we expect map quality to either stabilize over time or show overfitting such as seen in machine learning. We use the moment that either overfitting or stabilization occurs as indication that the map has adapted to changes in the world.

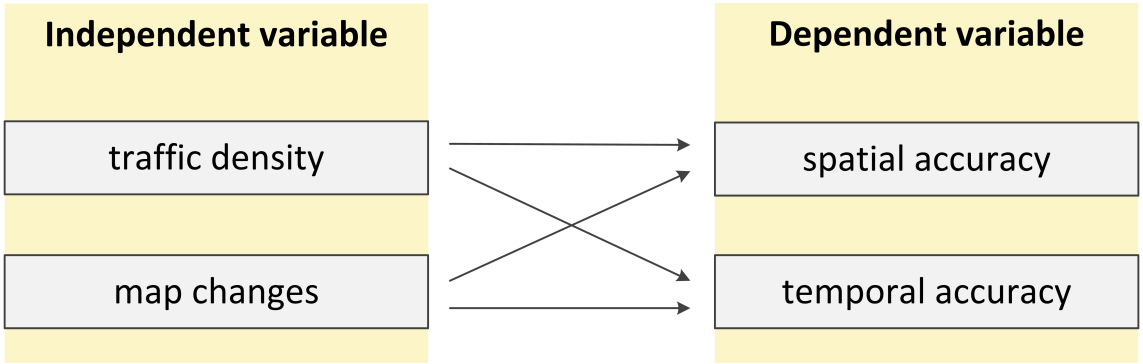


Figure 1.1: Structural diagram for the experimental design. The predictive relationships from independent to dependent variables are explored in the experiments as well as the correlation between spatial and temporal accuracy.

1.5. Scientific and practical relevance

The research performed in this thesis provides a proof of concept for a novelty in dynamic map generation by adding a method of both growth and decay of the map topology. This online approach to mapping is explored in some but often without integration of traffic information and decay of topology. By adding dynamic decay to the map topology, a map can ideally be kept and updated without having to rebuild the entire map.

The evaluation of the developed map-generation method during execution provides novel insight in the incremental application of trace-merging algorithms. With no known literature on the development of map quality over time this temporal aspect is a novelty to the field of Geographic Information Systems. The obtained insight in the process of development of web-like structures is an addition to the known literature on trace merging methods.

Because evaluation of two maps is mostly done by raster comparison and the available vector-based comparison of Biagioni and Eriksson [3] does directly compare graphs, we believe that the method for comparison of two vector maps using a graph-based approach is a novelty in the field. The developed metric is an advantage to the method of Biagioni and Eriksson for its ability to deal with displaced maps which occur in dynamic map generation.

The practical value of this research is found in the fact that the developed algorithm can be applied to situations where no map is present or current maps are no longer of use such as in disaster response or relief works. The development of a map by all of its users and the speed by which this map provides an up-to-date and dynamic map is an advantage to such cooperative efforts requiring active user actions. Without the need for confirmation, ROMA prevents extra cognitive stress for the users who need to focus on the task at hand (i.e. driving a vehicle).

1.6. Research focus

Within this research we focus on the algorithm for development of a dynamic map. Using a simulated environment, we simulated central processing without having to communicate the resulting map to any clients. We were also able to directly provide an error-free signal to the algorithm. In a real application errors occur during transmission of data and non-continuous connections make that data needs to be buffered at the server or client-side. The GPS-measurements simulated on a world-model are fed as raw data and application of extensive filters and smoothening of the GPS-trace are left outside of this research. In a real application measurements might be obtained by sensor-fusion in order to improve position estimates, but that is outside the scope of this research. For more on sensor fusion, please refer to [26]. The GPS-data simulated in this thesis is timestamped, and has an indication of measurement accuracy.

ROMA produces a vector-map for navigational purposes. Cao and Crumm [7] and Wing [29, pp. 32-43] indicate that a vector-map is most suited for navigation since it provides information on topological connectivity and geometry of the road. According to Schroedl et al. [24], digital maps are usually represented as graphs which show the topological connectivity.

The disaster relief setting described in section 1.1 makes that we require ROMA to be able to operate without a priori map and deal with changes in the map even as long as new measurements are fed to ROMA. Both the spatial and temporal accuracy of the developed map are the focus of this research. Spatial accuracy is the similarity of the map to the real world at a given time, and temporal accuracy is measured by the speed of the map to respond to changes in the topology or geography of the world.

Drivers (and rescue workers) operate in an already complex environment and it is advised not to ask drivers for direct feedback when it can be solved otherwise [25]. We therefore focus the research on an algorithm for fully automated mapping.

Given the focus described above, we defined the following requirements to ROMA:

- use GPS-measurements as input
- represent the map as a graph
- require no a-priori map
- recognize changes in the world
- require no active user-input

1.7. Outline

The outline of this thesis follows the methodology described in section 1.4. Chapter 2 lists background knowledge for an enhanced understanding of the matter and reviews related work that forms the foundation of the measure and algorithm developed in this thesis. Chapter 3 shows the developed measure for comparison of two vector-maps and chapter 4 describes in detail the design and workings of the developed algorithm, ROMA. In chapter 5 we briefly describe the simulator which is used for the experiments which are described together with their results in chapter 6. We discuss these results in chapter 7 and conclude this thesis in chapter 8.

2

Background and related work

This chapter explores literature on mapping and related fields to provide a decent background on which this thesis is based. Each section of this chapter concludes with the selection of appropriate methods which are later used in this thesis. Since the development of a dynamic map is central in this thesis, we first explore the possibilities for storing a map in section 2.1.

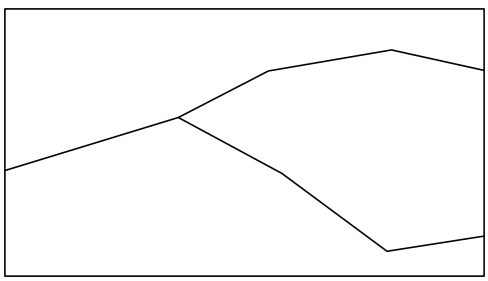
To automatize the process of collecting traces and using them to update maps, client–server frameworks have been developed such as the ActMAP–FeedMAP framework developed by Thomas et al. [25]. Since we focus on the algorithmic aspect of map inference, we use the sequential steps defined by Edelkamp et al. [14]. Their architecture is based upon a detailed analysis of previous work and can be used for automatic map generation with or without an initial base map. In short their approach follows these steps, with the sections in which we describe them between parentheses:

- filtering of incoming traces (section 2.2)
- matching traces to the existing map (section 2.3)
- creating a new road (section 2.4) or updating existing roads (section 2.5)

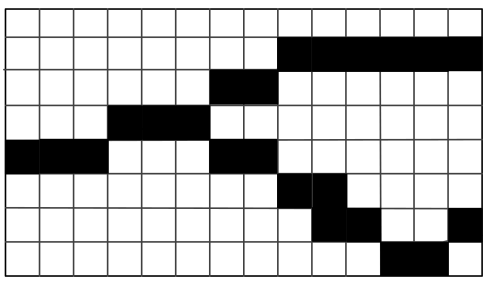
We conclude with an exploration of evaluation methods for developed maps in section 2.6.

2.1. Maps

According to Wing [29, p. 38] map data is commonly stored into one of two formats: raster or vector. The differences between both formats are shown in fig. 2.1 Maps stored in raster format make use of a grid where each of the positions on the grid gives information on a location in the world. The vector format uses a graph representation of the world where points and lines between them represent roads and areas. Labels are added in both representations to indicate what the stored locations represent.



(a) Vector representation



(b) Raster representation.

Figure 2.1: The same world represented as vector (fig. 2.1a) and raster (fig. 2.1b).

Wing [29, pp. 32-43] suggests that vectors are best used to represent the traces and map. Vectors provide information on connectivity between locations and the precision of these locations is not limited by the grid size as in raster maps. The represented locations can be exact latitude and longitude values related to a real-world location.

A vector map used for representation of a road network is conceptually related to a mathematical graph representation. A graph is essentially a collection of objects or nodes ($v \in V$) of which some are connected through so called edges ($e \in E$). A graphical representation of such a collection is shown in fig. 2.2 and makes the relation with vector maps clear. Roads in vector maps are translated to edges in a graph and intersections translate to nodes. Since graphs are well studied concepts in mathematics and offer benefits over raster representations for their direct representation of connectivity. The main difference between vector maps and graphs is that vector maps contain information on the position of nodes whereas nodes in graphs do not. The distance between two nodes can be known in a graph by adding a weight value representing the distance to the edge connecting the nodes.

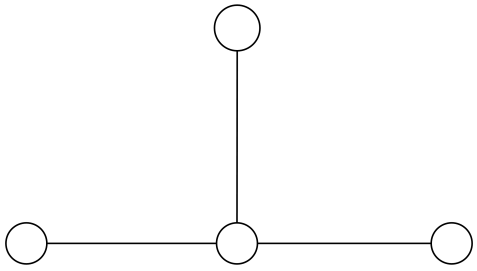


Figure 2.2: A graph with 4 nodes and 3 edges.

Graph type	node (v)	edge (e)
Regular	○	———
World	●	– – – –
Map	■	———
Measurement	▲	· · · ·

Table 2.1: Icons for graphs used in this thesis.

A vector representation is used in this thesis because of the unlimited location specificity and spatial resolution as well as the ease of topology representation. Table 2.1 shows the icons for different graph types used in this thesis: regular (without geographical position), world, map, and measurement.

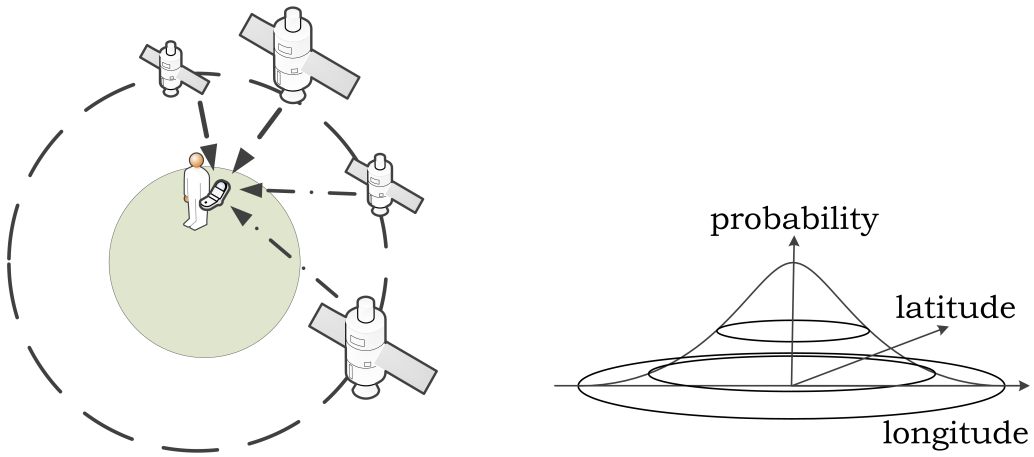
2.2. Pre-processing

The research in this thesis focuses on the use of GPS measurements as input for the algorithm described in chapter 4. GPS measurements are obtained with a GPS receiver which listens to signals from multiple geostationary satellites as shown in fig. 2.3a. Knowledge of the satellite locations and the difference in time of arrival of received signals enables the construction of a position measurement on earth. Constructed measurements contain information on the position of the receiver in 3 dimensions, the error of the measurement, and the moment of measurement. The error on GPS position measurements can be assumed to be circular Gaussian distributed on the horizontal plane [27] as shown in fig. 2.3b. This means that the value of a measurement is 95.45% certain to be within 2 standard deviations of the measured position. Derived information available after calculations on multiple measurements are speed and direction of the receiver.

Based upon the accuracy value of the estimated positions in GPS measurements, Lima and Ferreira [18] define two filters. One filter is based upon the degradation level of the horizontal positioning accuracy of the GPS and removes measurements which are too inaccurate. Another filter removes measurements obtained from less then five satellites, since no decent error estimate can be obtained for those measurements.

Given a subsequent measurements, it is possible to define filters based upon velocity information. Lima and Ferreira [18] throw away all data points with a speed of less then 6 km/h, considering these measurements to be insufficiently accurate. Bruntrup et al. [5] use a threshold on absolute velocity and acceleration to detect and eliminate unrealistic outliers. This last method does rely heavily on the characteristics of the data provider; a pedestrian moving at a speed ≥ 50 km h^{-1} is not realistic, but for a car in an urban area it is a regular speed.

When other information on the environment is available, this can be included for increased accuracy using a Kalman filter. Kalman filtering is often applied in mobile robotics and used in pre-processing by Mayr [19]. Their approach combines a model for GPS accuracy, a model for the error induced by the travelled terrain, and one model for the vehicle characteristics. Kalman



(a) Acquiring signals of GPS satellites.

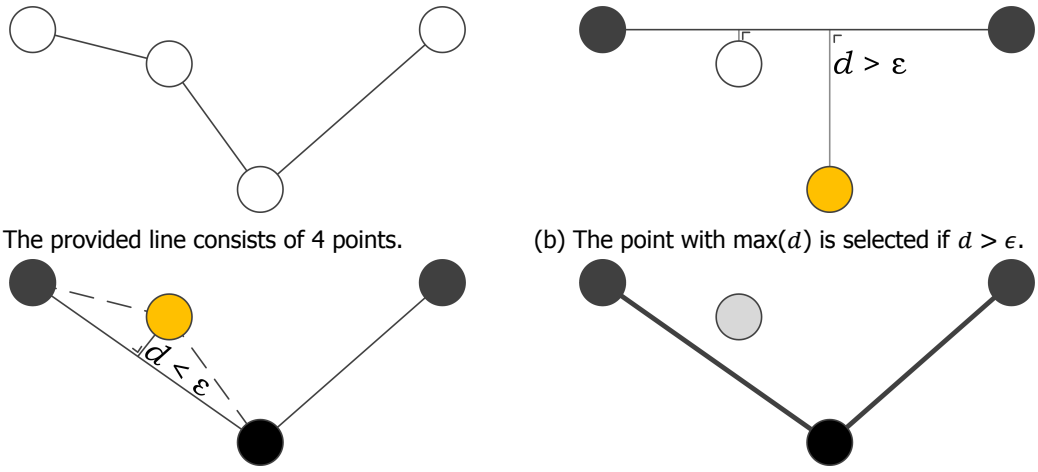
(b) GPS error on the horizontal plane.

Figure 2.3: Signals from GPS satellites (fig. 2.3a) provide an estimated position on earth with a Gaussian error on the horizontal plane (fig. 2.3b).

filters use these models to predict the next position that will be measured and use this prediction to reduce error on the actually measured position. Thrun et al. [26] provide a decent coverage on the use of Kalman filters for localization and mapping.

After removal of inconsistent data points, many methods split traces with irregularities between measurements. This can be done based upon a large time interval [7, 18], a large distance interval [7, 31], and also when there is a significant change in direction [20, 31].

As a final step in pre-processing, traces can be smoothed by the reduction of points to represent the trace. Lima and Ferreira [18] make use of the well known off-line Douglas-Peucker algorithm [12] with a static distance threshold. An example of the Douglas-Peucker algorithm for line simplification is given in fig. 2.4. Cao and Crumm [7] use a variable distance threshold based upon the angle between measurements so that straight lines are represented with less points than bends. Although found in pre-processing steps, we believe that smoothing can also be applied to the resulting map.



(a) The provided line consists of 4 points.

(b) The point with max(d) is selected if $d > \epsilon$.

(c) The method is called recursively until $d < \epsilon$.

(d) The resulting line excludes the grey point.

Figure 2.4: Example of the Douglas-Peucker algorithm Douglas and Peucker [12] for line simplification.

For use in this work we require filters which are non-dependent on vehicle characteristics since we simulate these vehicles. The filters we describe in section 4.1 are based upon measurement accuracy as done by Lima and Ferreira [18]. We also split traces based upon a distance interval as done by Cao and Crumm [7]. Line simplification is not done in pre-processing, but applied after

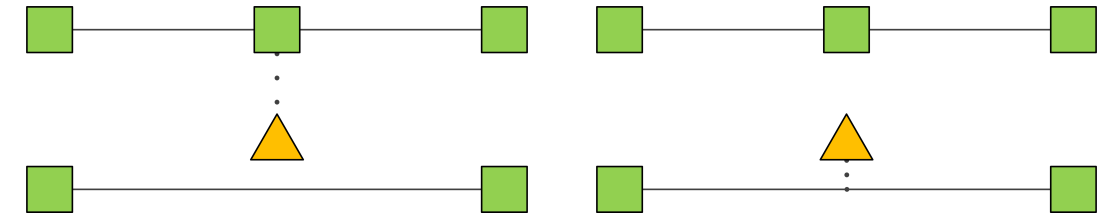
adjustment of the map as can be found in section 4.4. We do this to be able to add as much measurements as possible to the map instead of throwing them away beforehand.

2.3. Map matching

A match with the existing map is tried using the filtered trace of GPS measurements. Each of these measurements is attempted to match to the existing map in this step.

For limiting the search for possible routes on which a vehicle might have driven, Pfoser and Jensen [21] propose a method for defining the area within which a vehicle could have driven in between measurements. Using the accuracy of two measurements and the maximum driven speed between those measurements they define an error-ellipse as that area. This area limits the search but does not cover the map matching problem itself.

In the overview of White et al. [28] several approaches to the problem of map matching are discussed. The most simple method in terms of implementation and execution is point to point matching shown in fig. 2.5a. This method matches a GPS measurement to the nearest node in the map, making it highly dependent on spread of shape points over the map. Point to curve matching shown in fig. 2.5b does not have this dependency since it matches the measurement to the nearest edge instead. This solves the dependency stated for point to point matching, but some other problems are still unsolved.



(a) Point to point matching. (b) Point to edge matching.

Figure 2.5: Examples of different point matching methods.

White et al. [28] suggest improving the map matching procedure by using similarity measures such as difference in heading next to only the distance. An example of a similarity measure based on heading is given in fig. 2.6. This kind of information is used by Brakatsoulas et al. [4] in their incremental map matching algorithm and combined into one similarity measure. Topological information can be used to make sure that a trace is matched in a traversable route over the map as applied by Bruntrup et al. [5] using a depth-first search over possible routes to find the best match as shown in fig. 2.7. This look-ahead method comes at the cost of a more extensive search for the best route to match, but does not affect computational complexity since it is bound by an absolute amount of extra measurements to be explored.

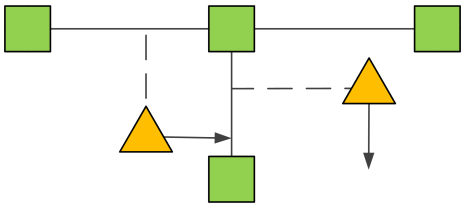
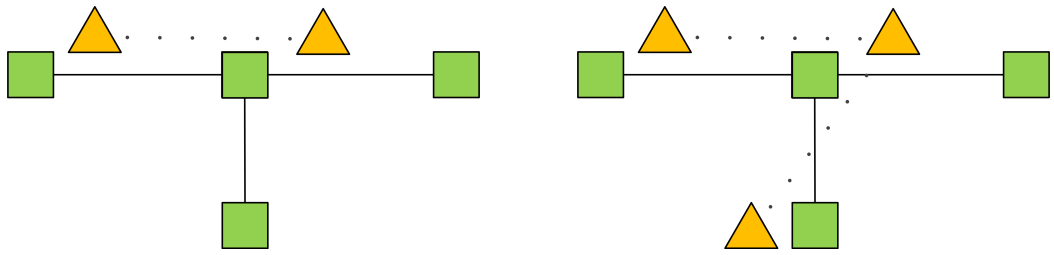


Figure 2.6: The effects of a match based on heading for two measurements.

Most map matching methods deal with maps that they assume are correct, but Haunert and Budig [17] have devised a method which also handles incomplete road data. Their off-line approach allows for the map matching to jump to another piece of the map when no feasible route is present. They also include a set of parameters allowing the algorithm to balance between matching to the map and going off-road.

The path estimation described in section 4.2 uses point-to-edge matching augmented with similarity based upon trace heading and look-ahead as done by Brakatsoulas et al. [4]. Furthermore we restrict the search for fitting edges by application of the method proposed by Pfoser and



(a) Two measurements indicate a straight line.

(b) Looking ahead shows a change in direction.

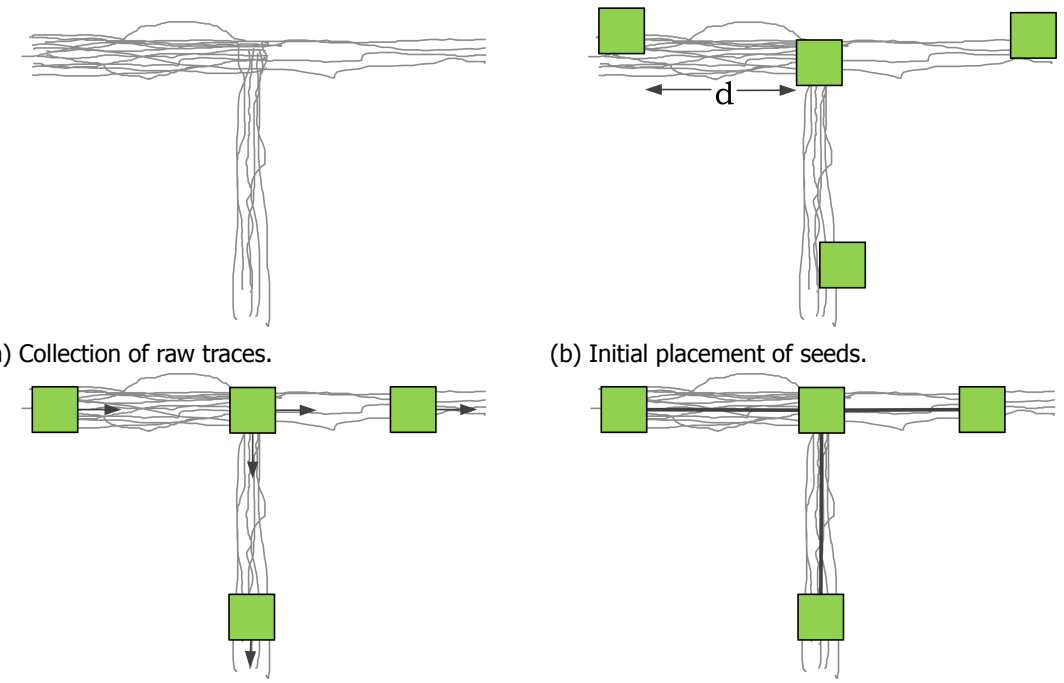
Figure 2.7: A local look-ahead provides more certainty as to which turn was taken.

Jensen [21] to increase algorithmic speed. The method for dealing with incomplete maps is used as inspiration for our map generation method as described in section 4.3.

2.4. Map merging

As defined by Biagioni and Eriksson [3], map merging can be categorized by their algorithmic foundations: k-means, trace merging, or kernel density estimation. In this section we describe all three of these categories.

Edelkamp and Schrödl [13] apply the *k-means* approach as illustrated in fig. 2.8 to map merging by distributing a set of cluster seeds at locations found in the trace data. The coverage of these seeds is chosen in such a way that every measurement found in the trace data is within a fixed distance d and bearing difference δ of a cluster seed which is seen as the de facto standard in this approach [3]. The location and heading of each cluster is then determined by averaging over the traces belonging to that cluster and connections between clusters are determined using these same traces as shown in fig. 2.8c. K-means can also be applied in an on-line manner in which cluster locations can change or new clusters can be added after processing new traces [13]. Schroedl et al. [24] later extended this methodology by extracting lane positions by evaluating the cross-section of traces perpendicular to the cluster heading.



(a) Collection of raw traces.

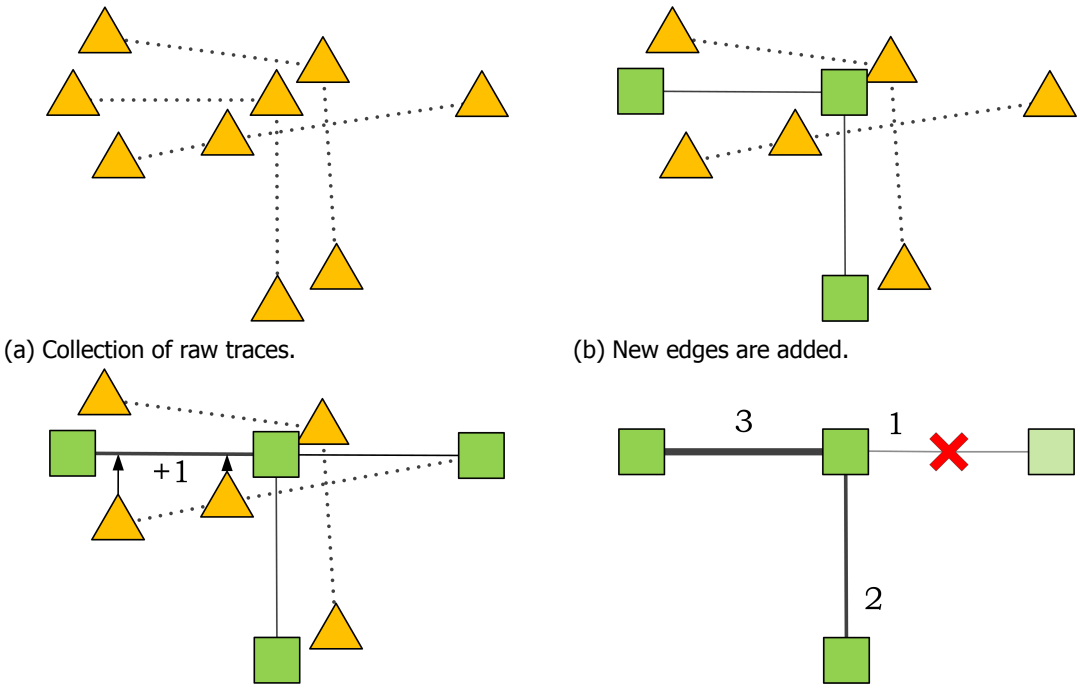
(b) Initial placement of seeds.

(c) Updated means with bearing.

(d) Final graph after means are linked.

Figure 2.8: Example of kernel density estimation (KDE) for map merging as explained in [3]

In the iterative *trace merge* approach, edges from collected traces are added to the map unless a similar edge is already present at that location as illustrated in fig. 2.9. The weight of each edge represents the amount of traces passing over that edge and edges with a weight below a certain threshold are removed at the end of the trace merging approach [3]. Bruntrup et al. [5] present a on-line applicable method for trace merging, where several position indicators move along with the trace. These mark the position until where the trace is explored, where map merging can be applied, and where map merging has been applied. The off-line approach by Cao and Crumm [7] reduces the effect of GPS noise by pulling together similar traces which they expect to be representative of the same road.



(a) Collection of raw traces.

(b) New edges are added.

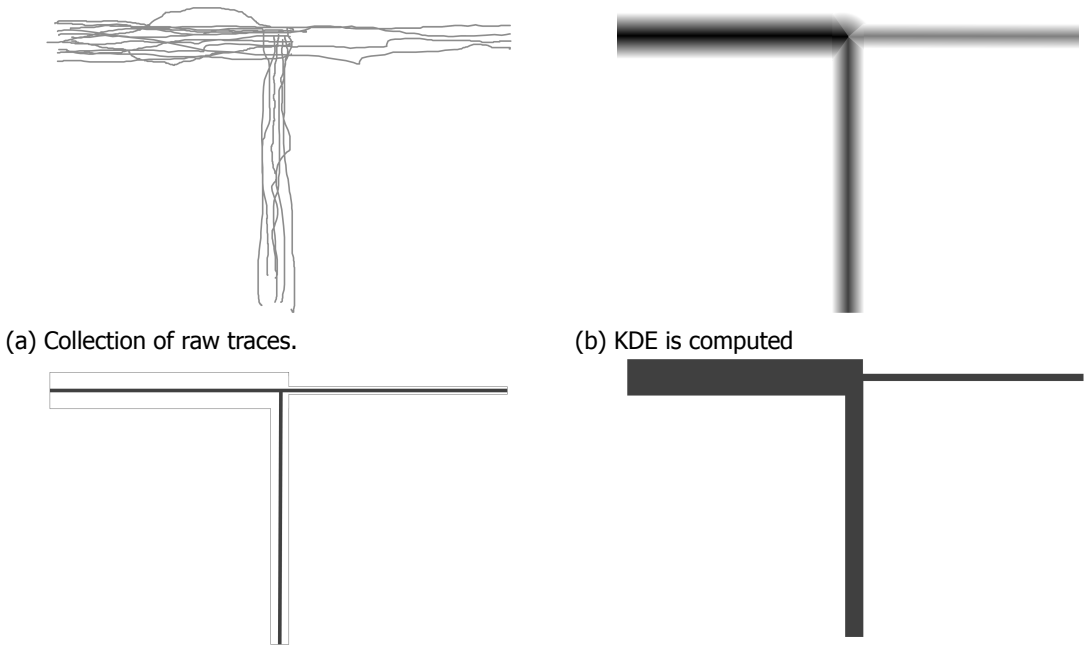
(c) Weight of existing edges increases.

(d) Edges with insufficient weight are removed.

Figure 2.9: Example of trace merging as explained in [3]

In *kernel density estimation* (KDE) as illustrated in fig. 2.10, collected traces are first transferred onto a raster which is then smoothed before the centrelines of driven roads are extracted [3]. The two-dimensional raster contains cell values representing the amount of traces passing through that cell. This raster representation is then smoothed by application of KDE which is a method that assumes the Gaussian distribution of measurements. We know from section 2.2 that GPS measurements show a Gaussian error distribution. Davies et al. [11] apply a threshold function over the smoothed image which results in a a binary image and ensures that erroneous measurements are left out of the map. The Voronoi-graph of this binary image then shows the estimation of road centrelines in a graph representation. Lima and Ferreira [18] take a different approach and apply a method similar to the generation of clusters found in the k-means approach after the rasterization of traces. They place so called centroids at specific distances related to the amount of traffic passing that point; when there is more traffic, centroids may be placed closer together. Lima and Ferreira continue by connecting these centroids with edges and hereby construct a vector representation of the road map. The KDE is not applicable in a dynamic or incremental fashion because all traces on the raster are needed to perform the kernel density estimation.

We believe that k-means and trace merging are suitable options to apply in an on-line and dynamic approach to map inference. Since trace merging can even directly merge sections of a found trace to the available map we use that approach in section 4.3. The centreline extraction method found in the k-means approach by Edelkamp and Schrödl [13] is used to determine node locations.



(a) Collection of raw traces.

(b) KDE is computed

(c) An absolute threshold is applied.

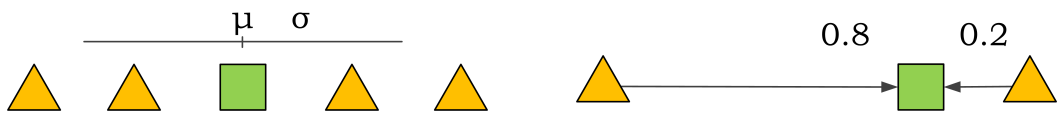
(d) Road outlines and centrelines are computed.

Figure 2.10: Example of kernel density estimation (KDE) for map merging as explained in [3]

2.5. Map dynamics

The measurements being added to the map can influence the map in a variety of ways. In this section we discuss how the aggregation of measurements can contribute to map dynamics. We start with a description of the various methods of storing measurements in the generated graph related to the estimation of centrelines and continue with methods for increasing map accuracy and decreasing the influence of measurement outliers.

The *k-means* method described in section 2.4 averages the position on the plane perpendicular to the cluster direction of all related traces to determine the centreline of the road described by it as shown in fig. 2.11a. The requirements of traces belonging to a cluster are only described in terms of spatial characteristics (distance or directionality), but leave out how to handle changes over time. In *trace merging* it is possible to accumulate knowledge on multiple measurements in one map node as if it were one measurement. Cao and Crumm [7] align similar traces in an elaborate trace refinement step before actual map merging which decreases the need for actual merger of measurements. They therefore only use extra measurements to increase traffic volume over edges and to create new nodes for previously unexplored roads. Rogers [23] adds new traces to the already developed map by weighing the positions of map and trace based upon accuracy of both positions as shown in fig. 2.11b. The resulting new position for the map is then adjusted and its accuracy value improves. Rogers states that this improvement to map accuracy comes at the cost of increased inertia of the map. Edelkamp et al. [14] furthermore state that the improvements make the estimate biased towards recent measurements and suggest to apply a batch-based reconstruction of the entire map periodically. Although a bias towards recent measurements is welcome given the need to adjust to changing environments, our method should deal with the described map inertia.



(a) Averaging over all measurements.

(b) Weighted addition of a new measurement.

Figure 2.11: Options for determining the position of a node on the map.

Given that measurement data is stored in the graph, various authors have shown methods to determine characteristics other than general topology and road geography. Some authors have developed methods to detect the amount of lanes on a road [10, 13, 24], when lanes split and merge [7, 13, 24], or even the precise layout of and legal turns on intersections [13, 18, 24]. Others focused on finding average and maximum speeds allowed on a road [20, 25] resulting in the possibility to estimate the road type (e.g. highways, streets, or walkways) [20]. Since vertical accuracy of GPS signals is quite bad [27], Niehöfer et al. [20] developed a method of detecting bridges and tunnels based upon the signal strength of received GPS signals. In our work, we will let these advanced characteristics untouched and focus on determining the changes of topology and geometry over time.

Although it is known that maps change over time [5] and for some domains (e.g. mining) significant changes can occur within a week [1], no methods of dealing with temporal change have been found other than adding new traces to a map. Deletion of removed roads is often overlooked. Niehöfer et al. [20] state that aged map segments will be deleted eventually in their method, but do not explain their decision method. Davies et al. [11] give the most detailed description and state that the level of trust in certain parts of the map decreases over time. The trust value is increased by adding traces to the map where older traces provide less trust compared to more recent measurements. Roads are removed from the map as soon as the level of trust comes below a certain threshold. Davies, Beresford, and Hopper finally indicate that detecting removal of roads depends much on the amount of vehicles providing GPS traces.

Floreano and Mattiussi [15] describe the use of pheromones by ants to define paths and apply it in algorithms. Ants apply pheromones which fade over time to mark paths travelled between food sources and the nest as shown in fig. 2.12. Each passing ant leaves a pheromone marker which decays in strength over time. Ants prefer the route with most pheromones and eventually find the shortest path as emergent behaviour. For our research, measurements merged to the map can confirm existence of the matched road using a similar method. We will have to adjust this method to work with different traffic intensities since not every road is travelled similarly often.

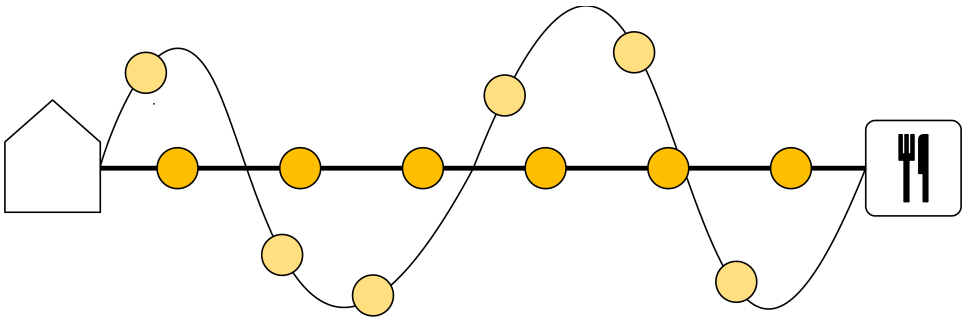


Figure 2.12: Pheromone trails from hill to food. The path will converge to the shortest path due to decay of pheromones.

Many authors [7, 13, 18] apply clean up steps to prevent outliers through single measurements. Some apply a threshold on traffic volume, but the values used differs per author. Cao and Crumm [7] remove edges with a traffic volume < 3 whereas Lima and Ferreira [18] need > 20 traces before they assume a road is valid. Another clean up method is applied by Edelkamp and Schrödl [13] who remove initial and final trace sections since these can be individual driveways or parking lots.

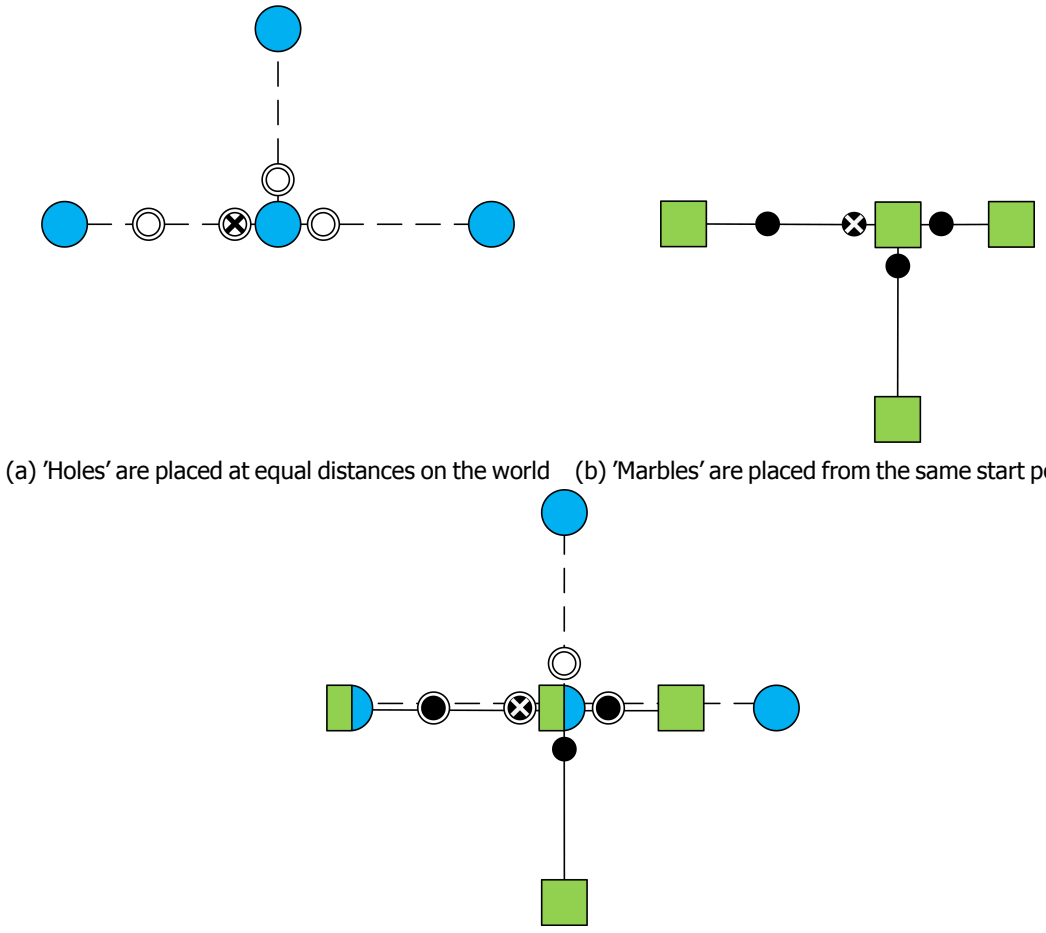
In our approach to map dynamics described in section 4.4, we store a limited set of most recent measurements in each element in the graph. This enables us to benefit from the possibilities of Gaussian estimation found in k-means while keeping only to the most recent measurements to facilitate adjustments to road geometry and prevent map inertia. We model fading of roads by estimating a time before we expect that a new car will pass based upon the recorded traffic characteristics which is inspired by the concept of pheromones described in this section. As a clean up step we disregard edges with a low traffic volume and also dead end streets below a certain length.

2.6. Map validation

Biagioni and Eriksson [3] note in their survey of existing literature that evaluation is almost exclusively done visually. This is often done by laying the produced map onto a ground truth map. Such maps are often obtained from arbitrarily chosen commercially available maps [7, 18, 20] and in some cases constructed using a high-precision Differential GPS [13, 24]. No ground truth was available in cases were data from open pit mining was used [1, 30]. Agamennoni et al. [1] solved this lack of ground truth by comparing their algorithm to other well known methods [5, 11, 24].

Quantitative measures for measuring the quality of a match of a GPS trace to the ground truth map exist (see Brakatsoulas et al. [4]), but for comparison of a generated map with the available ground truth this is still uncommon [3]. Biagioni and Eriksson [3] claim to be the first authors offering both a quantitative and qualitative comparison of existing map generation methods. We doubt if they are the first to apply quantitative comparison since Lima and Ferreira [18] already provide information on general positives, false positives, and false negatives. However, they do not provide a clear description on their method. Biagioni and Eriksson provide a method which automatically calculates these measures and we therefore describe their method in this section 2.6. They also compare three well-known algorithms from each of the three approaches: k-means [13], trace merging [7], and KDE [11].

Biagioni and Eriksson [3] measure both geometric and topological similarity of maps by taking samples of both the map and ground truth. They explore the map up until a certain distance from a random chosen starting point and take samples at a fixed interval over all outgoing edges as shown in fig. 2.13. This operation is performed for both the map and the ground truth after which it is determined which samples from both sets match within a specified matching distance.



(a) 'Holes' are placed at equal distances on the world (b) 'Marbles' are placed from the same start position.

(c) True positives where 'marbles' match 'holes'.

Figure 2.13: Example of the method for map comparison by Biagioni and Eriksson [3].

Biagioni and Eriksson [3] translate these matches into terms found in information retrieval so that one accuracy value can be calculated. Within table 2.2 we summarize the terms used in information retrieval and how they relate to the observations (map) and ground truth (world). All matched samples are considered as true positives (TP), unmatched samples present on the map represent false positives (FP), and unmatched samples on the ground truth are seen as false negatives (FN). True negatives (TN) can not be calculated in this context as there are no bounds to the total search area.

		ground truth	
		available	not available
observations	available	true positive	false positive
	not available	false negative	true negative

Table 2.2: Relations between correctness of observations and ground truth

These terms used for classification methods are then used to produce precision and recall, of which the standard formulae are given in eq. (2.1a) and eq. (2.1b). Precision represents the part of selected items which are relevant and translates to the part of the map which is actually a representation of the world. Recall represents the part of relevant items which are selected and translates to the part of the world which is actually represented on the map.

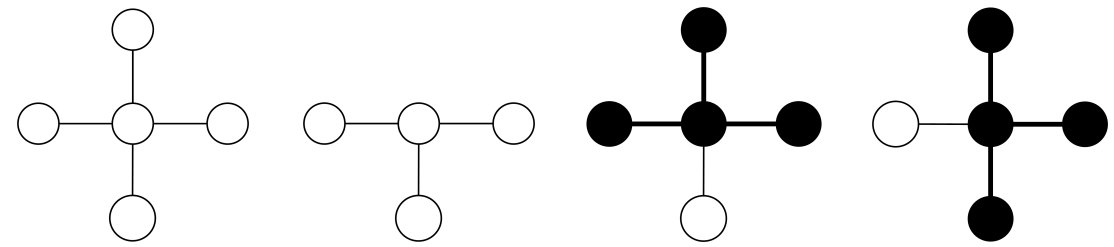
$$\text{precision} = \frac{TP}{TP + FP} \tag{2.1a}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.1b}$$

As both precision and recall indicate fractions of a whole, their values are on the interval $[0, 1]$. The harmonic mean of precision and recall then combines these values into the balanced F-score as shown in eq. (2.2), and results in values on the interval $[0, 1]$.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.2}$$

Biagioni and Eriksson [3] state that a graph similarity algorithms are able to measure the degree of topological similarity but believe that these methods are not suited for measuring any geographical qualities. In this thesis, we propose a method using graph similarity which is able to measure both topological and geographical qualities of a developed map. We shall therefore describe the maximum common edge subgraph (MCES) isomorphism and the limitations which need to be overcome. Two graphs are isomorphic when there is a one-to-one correspondence between nodes and edges only exist between two nodes in one graph when an edge is also present between the two corresponding nodes in the other graph [22]. Raymond and Willett [22] further describe the MCES isomorphism as a subgraph consisting of the largest number of edges between two graphs. An example of a MCES is given in fig. 2.14.



(a) Graph G^1 (b) Graph G^2 (c) MCES of $G^1 \cap G^2$ (d) MCES of $G^1 \cap G^2$

Figure 2.14: Graphs G^1 (2.14a) and G^2 (2.14b) can have many maximum common edge subgraphs. There are already $4 \cdot 3 \cdot 2 = 24$ possible maximum common edge subgraphs of $G^1 \cap G^2$ of which 2.14c and 2.14d are only two examples.

The problem of determining a MCES is categorized as NP-complete in terms of computational complexity given the amount of comparisons needed to check if there exists no subgraph with

more edges. Biagioni and Eriksson [3] furthermore argue that graphs do not possess geographical locations and therefore deem graph-based methods unfit for comparison of maps. We believe that the measurements stored in nodes in a road graph can be exploited to provided a distance measure used to match only nodes from the ground truth to the map which are geographically near to one another.

We agree with Biagioni and Eriksson [3] that quantitative evaluation is necessary to obtain insight in the quality of a mapping algorithm and follow their use of precision, recall, and the combined f-score. However, we see a possibility to apply the MCES concept to obtain a estimate of map quality without the necessity for the random selection of positions within the developed map. In chapter 3 we describe a method for application of the MCES concept which uses the geographical information stored in a vector map representation to enhance node mapping.

3

Evaluation criteria and metrics

In this chapter we describe the metrics used to determine map quality and how these are used to determine both spatial and temporal accuracy of the developed maps. The metrics are used in chapter 6 to quantitatively measure the effectiveness of the algorithm described in chapter 4. Contrary to the metric described by Biagioni and Eriksson [3], the metrics in the current chapter makes use of topological properties and graph-related methods. Section 3.1 describes the method inspired by the concept of the Maximum Common Edge Subgraph (MCES, section 2.6) isomorphism and determines which edges of the map are matched to edges in the world. In section 3.2 we use these matches we calculate true positives (TP), false positives (FP), and false negatives (FN) in order to calculate the spatial accuracy. In section 3.3, these metrics are used to define the temporal accuracy.

3.1. Common edge subgraph

This section describes the process for finding the true positive matchings map and world based upon the concept of the Maximum Common Edge Subgraph (MCES). After a brief recapitulation of the difference between regular graph comparison and the comparison of vector maps we continue to describe the process itself. The first part describes the method for application of geographical information found in vector maps so that the extracted subgraph contains edges which are also matched. The second part then describes how conflicts in node matching are solved using topological and geographical differences. This process results in an approximation of the MCES which takes into account the geographical information stored in vector-maps.

An important difference between regular graph matching and the comparison of vector maps is that nodes in maps have a geographical position. Next to this, the edges within a map have weights which represent the geographical length of the roads they represent. As stated in section 2.6, it is difficult to apply regular MCES algorithms for quantitative evaluation, since graphs do not have notion of their geographical location. We solve this problem by first matching the nodes in map and world based upon geographical distance.

Algorithm 3.1 Determining the matches between nodes in world and map

Require: map graph $G^m = (V^m, E^m)$, world graph $G^w = (V^w, E^w)$
1: $a \leftarrow \text{getNearestMatches}(V^m, V^w)$
2: $b \leftarrow \text{getNearestMatches}(V^w, V^m)$
3: $f \leftarrow a \cap b$
4: $t \leftarrow (a \cup b) \cap \neg f$
5: **while** $t \neq \emptyset$ **do**
6: $\quad c \leftarrow (v^m, v^w) : d(v^m, v^w)$ is minimal in t
7: $\quad$ remove all matches with v^m or v^w from t
8: $\quad f \leftarrow c$
9: **end whilereturn** f

The graph comparison method starts with alignment and matching of the nodes found in map and world. Each of the nodes in map and world contain position information: latitude, longitude, and altitude. Given that the measurements are taken on the surface of the earth, the orthodromic or great-circle distance (d) is used in this work to determine the distance between two node coordinates. The pseudocode overview for deriving suitable node matchings for the common edge subgraph is given in algorithm 3.1 and shows a computational complexity of O($|V|^2$). Matches for nodes (intersection or endpoint) from map (v^m) to world (v^w) and vice versa are made such that $d(v^m, v^w)$ is minimal. These two lists of matches are then searched and combined into t. All mappings between v^m and v^w existing in both lists are automatically stored into f. As long as there are matches remaining in t, the match with the smallest $d(v^m, v^w)$ is taken and stored into f followed by the removal from t of all matches containing either v^m or v^w equal the stored match. Exemplary matching for the graphs in fig. 3.1 is shown in fig. 3.2.

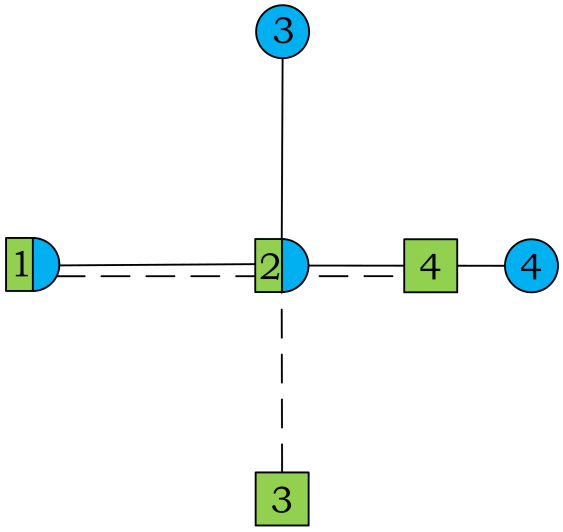


Figure 3.1: Example map (squares) and world (circles) graphs positioned on top of one another.

●	v_1^w	v_2^w	v_3^w	v_4^w
v_1^w	0	2	0	0
v_2^w	2	0	2	2
v_3^w	0	2	0	0
v_4^w	0	2	0	0

(a) Adjacency matrix for nodes in the world

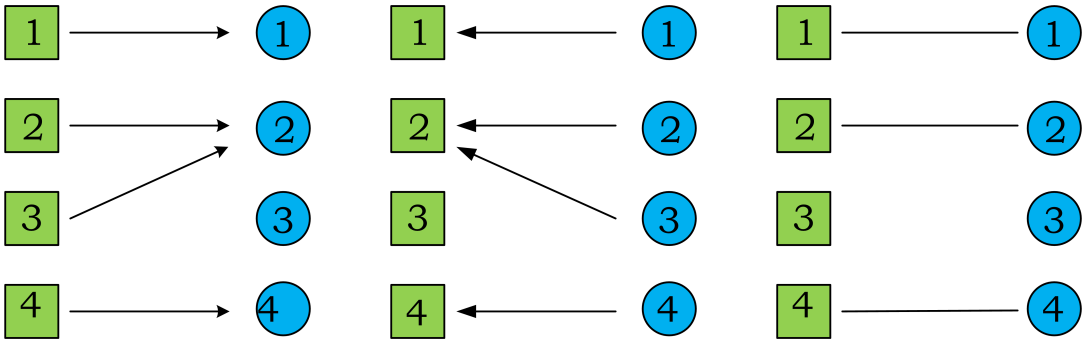
■	v_1^m	v_2^m	v_3^m	v_4^m
v_1^m	0	2	0	0
v_2^m	2	0	2	1
v_3^m	0	2	0	0
v_4^m	0	1	0	0

(b) Adjacency matrix for nodes on the map

Table 3.1: Adjacency matrices the world and map shown in fig. 3.1.

We can not assume that roads between matched nodes in the world are equal to those roads on the map. Therefore two weighted adjacency matrices are built for both world W and map M to be used for calculations of precision and recall. Within these two matrices the edges of the common subgraph are defined including their weight. The weights of edges for directly connected nodes are calculated using an all-pair shortest path algorithm of O($|V|^3$). The shortest paths are restricted to contain only direct paths with no matched nodes within the path itself. A similarity matrix S is defined to take geometric properties into account. The values for this matrix are calculated using a simple comparison of road lengths as shown in eq. (3.1). As required for a similarity measure, this equation is bound on the interval $[0, 1]$ since the exponent of the equation is bound on the interval $(-\infty, 0]$, $\lim_{x \to -\infty} e^{-x} = 0$, and $e^0 = 1$. The exemplary matrices in table 3.2 show the values for M, W, and S for the graphs in fig. 3.1 matched as shown in fig. 3.2.

$$S_{i,j} = e^{-\left(\mathrm{abs}(|e_{ij}^m| - |e_{ij}^w|) \cdot 2\right) \,/\, (|e_{ij}^m| + |e_{ij}^w|)} \tag{3.1}$$



(a) a at line 1. (b) b at line 2. (c) f at line 9.

Figure 3.2: Example of algorithm 3.1 at different moments for map and world in fig. 3.1. Since a match in both directions ($v_2^m \leftrightarrow v_2^w$) exists, the matches ($v_3^w \leftrightarrow v_2^m$) and ($v_3^m \leftrightarrow v_2^w$) are not included in f.

	v_1^w	v_2^w	v_4^w
v_1^w	0	2	0
v_2^w	2	0	2
v_4^w	0	2	0

(a) Weighted adjacency matrix W

	v_1^m	v_2^m	v_4^m
v_1^m	0	2	0
v_2^m	2	0	1
v_4^m	0	1	0

(b) Weighted adjacency matrix M

S	v_1^w	v_2^w	v_4^w
v_1^w	0	1	0
v_2^w	1	0	$e^{-(1/3)}$
v_4^w	0	$e^{-(1/3)}$	0

(c) Similarity matrix S

Table 3.2: Example matrices for the map and world shown in fig. 3.1 matched as described in fig. 3.2c

The computational complexity of the implementation of this approximation of the MCES is of cubic time $O(v^3)$ due to the all-pair shortest path approach to building the adjacency matrix. This metric as used in the simulations described in chapter 6 therefore falls within polynomial time.

3.2. Spatial accuracy

Given the matrices calculated in section 3.1 this section defines the quality measure for spatial accuracy. Precision and recall are both first calculated before combining them into the balanced F-score which is used as main quality measure. We adapt these measures to take geometric properties into account using the similarity matrix S.

Traditional precision can be calculated given the amount of true positives and false positives as defined in eq. (3.2a). In a summation over all elements in the matrices, the precision for map comparison is determined. M is used to measure precision since false positives can only be expressed in terms of size on the map and geographical similarity is included using S. Multiplication of these terms results in a scored length for each of the edges in the common subgraph. The summation of these lengths is divided by the total length of edges on the map to obtain the map's precision as shown in eq. (3.2b).

$$\text{traditional precision} = \frac{TP}{TP + FP} \tag{3.2a}$$

$$\text{vector-map precision} = \frac{\sum_{i=0}^{|S|} \sum_{j=0}^{|S|} S_{i,j} \cdot M_{i,j}}{\sum |e^m|} \tag{3.2b}$$

Traditional recall can be calculated given the amount of true positives and false negatives as defined in eq. (3.3a). Since false negatives are expressed in terms of size in the world, the recall value for map comparison is calculated using W. The traditional formula is transformed into one

for vector-map recall as given in eq. (3.3b) in a similar fashion as done for precision.

$$\text{traditional recall} = \frac{TP}{TP + FN} \tag{3.3a}$$

$$\text{vector-map recall} = \frac{\sum_{i=0}^{|S|} \sum_{j=0}^{|S|} S_{i,j} \cdot W_{i,j}}{\sum |e^w|} \tag{3.3b}$$

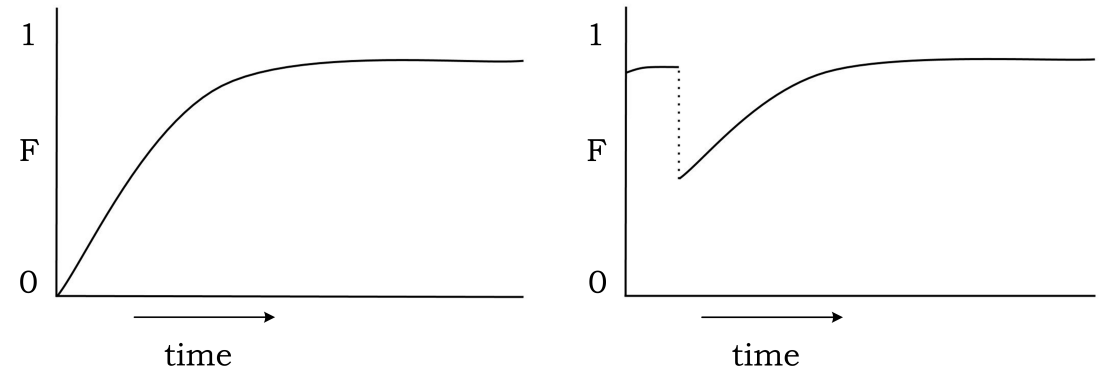
Given the precision and recall values adjusted for geographical similarity, the formula for determining their harmonic mean stays intact. The balanced F-score combines both precision and recall into one quality measure for spatial accuracy as defined in eq. (3.4). Application of the balanced F-score for spatial accuracy implicitly states that false positives and false negatives are equally unwanted.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.4}$$

3.3. Temporal accuracy

Precision, recall, and the F-score are all used in chapter 6 to examine the change of map quality after roads are added or removed. A map with optimal temporal accuracy would represent the world at any time exactly as it is. Because changes to the world are not always directly observed, the speed of recovery from these changes is used as an indicator for the temporal accuracy of the map. The speed of recovery is defined as the interval between introduction of changes in the world and the moment these changes are visible in the map.

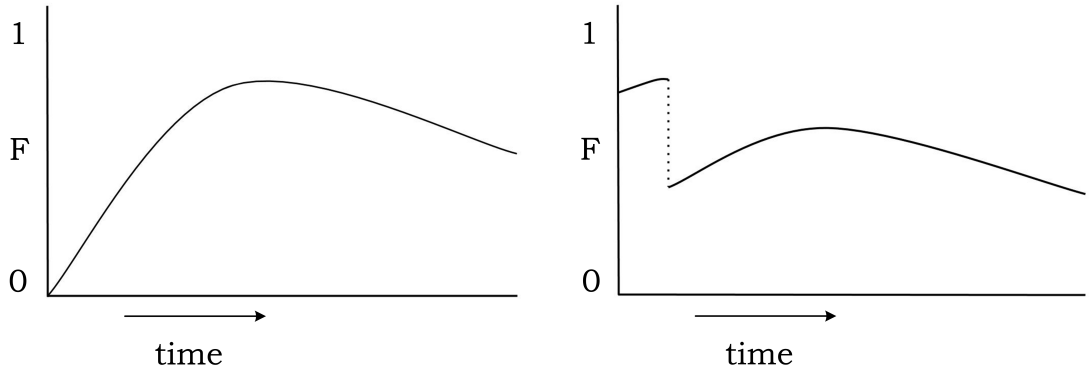
For an algorithm that produces maps without much added noise, the changes for experiments adding or removing roads would show a course in the F-score as shown in fig. 3.3. After introduction of new roads in an empty world, the map quality would increase up to at most a quality value of 1. Given an existing map and the removal of roads from the world, map quality would first fall directly due to misrepresentation of the world. The quality would then increase because the removal of these roads is detected by the algorithm and can ideally increase up to a quality value of 1.



(a) After road introduction (b) After road removal

Figure 3.3: Expected course of map quality in an ideal setting following road introduction and road removal

Noise has a significant effect on map quality, changing the expected course of the map quality as shown in fig. 3.4. For the introduction of new roads, the course changes to show a pattern similar to the effects of overfitting in machine learning. After removal of roads from an existing map, noise hinders the recovery of the map. Because of these effects we rely on comparison of map qualities for different settings in the experiments.



(a) After road introduction (b) After road removal

Figure 3.4: Expected course of map quality with increasing noise following road introduction and road removal.

4

Algorithm design

In this chapter we describe the robust online map-generation algorithm (ROMA). ROMA is a trace merging algorithm in the categorization by Biagioni and Eriksson [3]. It is executed in an on-line fashion using the processTrace-method every time that a GPS-trace of sufficient length becomes available. The outline of this method is given in algorithm 4.1 and shows four sequential steps: pre-processing, path estimation, path generation, and path adjustment. The pre-processing step filters the incoming GPS-trace based upon measurement accuracy and distance between measurements as described in section 4.1. Path estimation described in section 4.2 matches the GPS-trace to the existing graph by application of point-to-edge matching and uses a look-ahead method to improve the best match. The path generation step described in section 4.3 ensures extraction of the best possible path and closes gaps in the existing graph where needed. How the measurements in the GPS-trace influence the travelled path and how the dynamic nature of ROMA is ensured is described in section 4.4. The entire processTrace-method representing ROMA has a computational upper bound of O($|V| \cdot |E|$) defined by the getSpanningForest-method called in line 5 and described in section 4.2. This method generates a set of spanning trees, or a spanning forest, with limitations for which a subset E' of all edges in the graph has to be compared with the generated spanning forest which consists of a subset of nodes V in the graph resulting in the computational complexity of O($|V| \cdot |E|$).

Several variables for the algorithm are defined throughout the chapter as tuning parameters for the algorithm. Each of these variables has been determined using several simulator runs with 50 simulated cars using the simulator described in chapter 5. The ideal values obtained through these simulations are given where the variable is defined in the algorithm. Visual examples are provided throughout the chapter to accompany the procedural descriptions of steps in the processTrace-method. The initial graph $G = (V, E)$ used in the examples is shown in fig. 4.1 and consists of a set of nodes V connected by a set of edges E. Figure 4.2 shows the measurement stream $S = m_0, \ldots, m_5$ used in the examples. Each measurement m is essentially a vector containing the values defined in eq. (4.1).

$$m \in S = \begin{bmatrix} \tau_m \\ \text{pos}_m \\ \sigma_m \\ \angle_m \\ \text{speed}_m \end{bmatrix} \tag{4.1}$$

4.1. Pre-processing

In this section we describe the pre-processing step based upon the related work selected in section 2.2. The filter methods used in this work are non-dependent on vehicle characteristics for a more general applicability. A filter on accuracy and distance interval are applied in the filter-function on line 2 in algorithm 4.1. The filter on accuracy ensures that measurements with a low accuracy, high standard deviation ($\sigma < \alpha$), are removed as shown in fig. 4.3. Although the

Algorithm 4.1 General overview of the trace processor for ROMA

Require: map graph $G = (V, E)$
Require: a measurement stream S
1: **function** processTrace(S)
pre-processing — section 4.1
2: $M \leftarrow$ filter(S)
3: **if** $|M| \geq n + 2 \vee$ end of stream **then**
path estimation — section 4.2
4: $E' \leftarrow$ getFitEdges(M)
5: $T \leftarrow$ getSpanningForest(E', M)
path generation — section 4.3
6: $P \leftarrow$ getBestPath(T, E'^0)
7: **if** $P = \emptyset$ **then**
8: $P \leftarrow$ getBestPath(T, E'^1)
9: $T' \leftarrow$ getSpanningForest($p_0p_1, [m_1, m_0]$)
10: $P \leftarrow$ bridgeSmallestGap($T, T', [m_0, m_1]$)
11: **end if**
path adjustment — section 4.4
12: influencePath($P, [m_0, m_1]$)
13: **end if**
14: **end function**

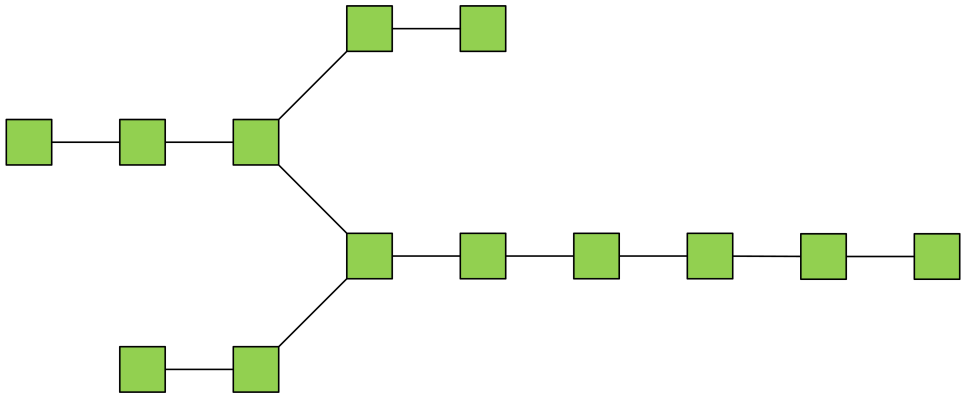


Figure 4.1: The graph G as used in visual examples throughout this chapter. The nodes $v \in V$ are shown as green squares with connecting edges $e \in E$ as lines.

functionality is implemented in ROMA, the simulations provide measurements with a constant σ. The value of α is therefore set to ∞. Trace splitting ensures that no gaps of a distance above β meters exist in the final set of measurements by splitting the trace as shown in fig. 4.4. The value of β was set to 300 meters following the advise in [7, 31]. The function ends with returning the set of filtered measurements in an array M. The length of M is limited by the value n representing the look-ahead size. Table 4.1 shows the size of M resulting from different choices of n in the exemplary trace shown in fig. 4.4. The value of n is set to 4 following the advice in [4, 5].

n	0	1	2	3
$\lvert M \rvert$	2	3	4	4

Table 4.1: The size of M for different look-ahead sizes n for the sequence of measurements shown in fig. 4.4.

4.2. Path estimation

In this section we describe the path estimation step based upon the related work in section 2.3. The functions as listed in algorithm 4.1 for point-to-edge matching and local look-ahead are first

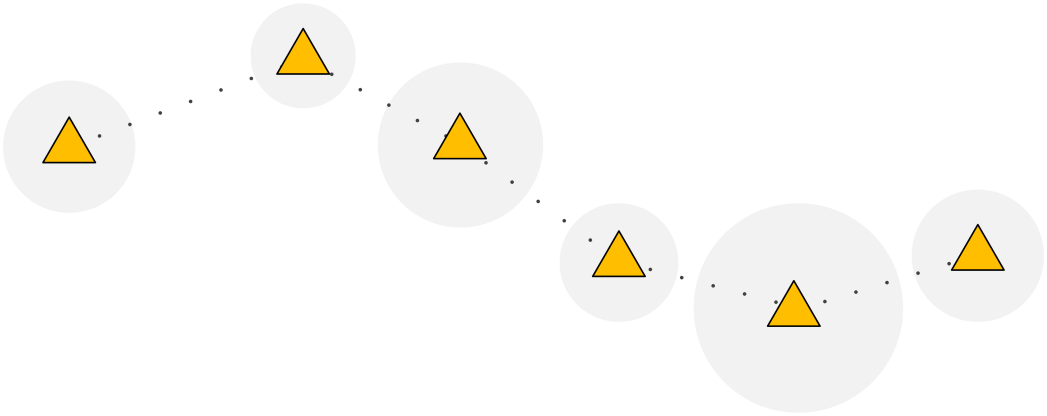


Figure 4.2: Example of a set of measurements S with six measurements used in this chapter. For each measurement m, a position (pos, centre of the circle), and accuracy (σ, shown as the grey circle) is shown. Time of measurement τ, direction $\angle$, and speed are not shown in the visualisations of this chapter.

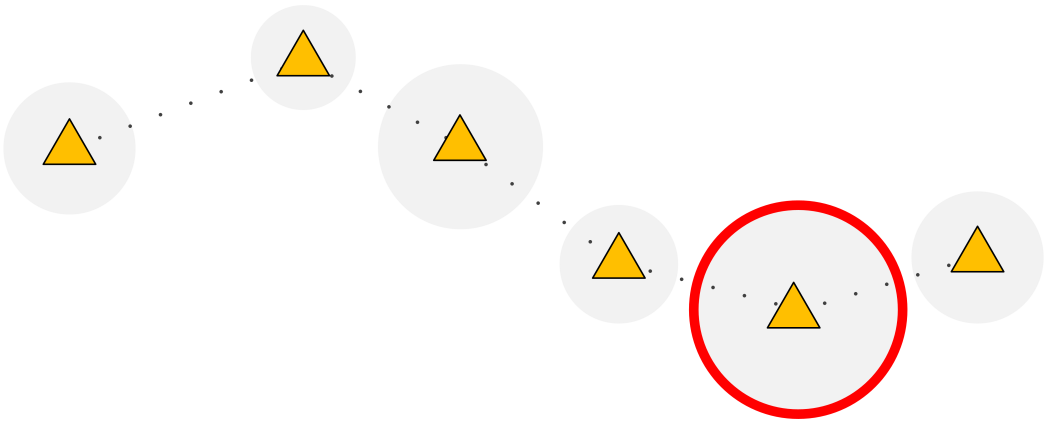


Figure 4.3: Example of a measurement with $\sigma > \alpha$. The highlighted measurement is therefore removed from the trace.

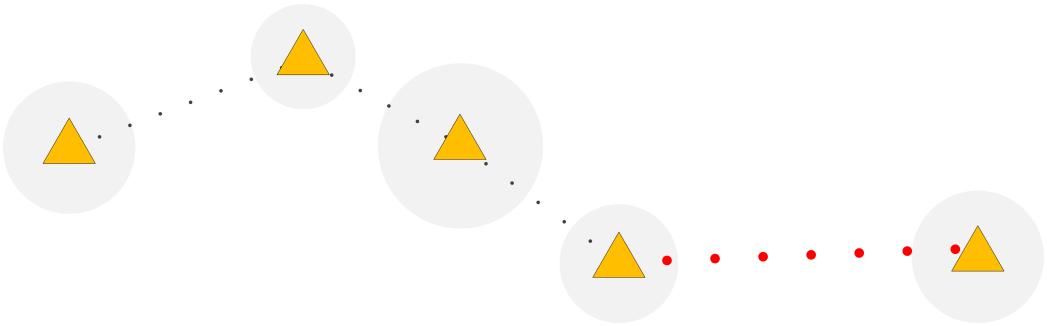


Figure 4.4: Example of a gap between measurements > β which is reason for a split as shown in table 4.1.

explained followed by examples to clarify their working.

The getFitEdges-function called in line 4 applies point-to-edge matching based on Brakatsoulas et al. [4]. Within the function each of the edges in the map constructed thus far is compared to the filtered measurements found in M. Each edge is matched to the measurement m to which it is most similar. The resulting set E' is therefore subdivided into $E'^0, \ldots, E'^{|M|-1}$ such that all edges most similar to m_i are contained in E'^i as shown in fig. 4.5. The similarity measure used for comparison is based on distance and heading and applies the formula given in eq. (4.2). The distance between measurement and edge is calculated in the function d using the orthodromic or great-circle distance. This distance is normalized by division through the error of measurement for m. In the calculation for angle difference between heading of the measurement directionality of

the edge is left out of consideration. The method returns only those edges with a similarity of at least γ to the matched measurement m. The value for γ is determined during tuning-experiments as 0.01.

$$\text{similarity}(m, e) = \frac{\text{d}(m, e)}{\sigma_m} \cdot \left(\frac{1}{2} + \frac{\cos(\text{abs}(\angle m - \angle e) * \pi)}{180} \right) \tag{4.2}$$

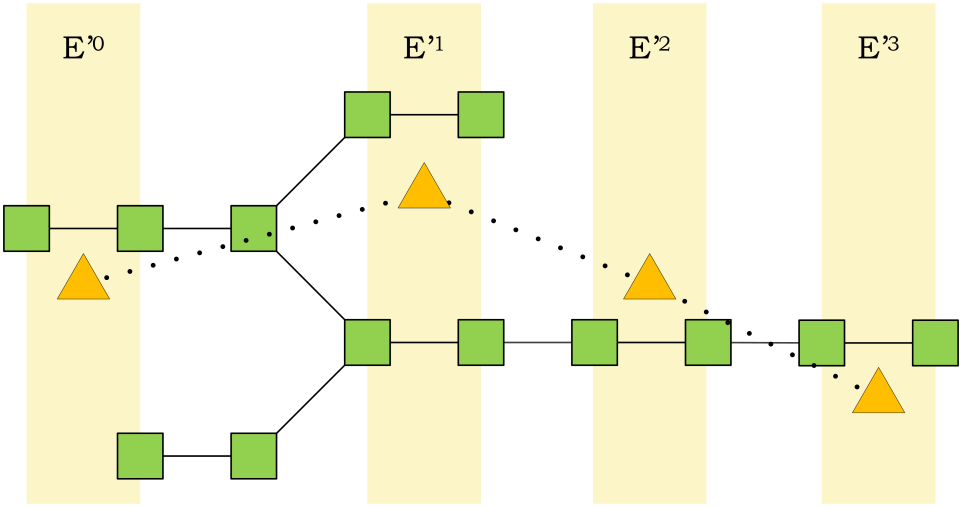


Figure 4.5: Example of how E' is divided into $E'^0, \ldots, E'^3$.

The getSpanningForest-method called in line 5 generates a spanning forest as shown in fig. 4.6 in order to provide a local look-ahead. A spanning forest provides a shortest path to all nodes reachable from a certain start node and a spanning forest has multiple start nodes and thus contains multiple spanning trees. An introduction to algorithms for the creation of spanning trees can be found in [16, pp. 638 - 642]. The developed spanning forest contains spanning trees originating from the nodes contained in all edges in E'. Based upon the measurement m_i matched to the originating edge e, each tree is pruned by the distance function defined by Pfoser and Jensen [21] as written in eq. (4.3). The pruning distance used in the construction of the minimal spanning forest is implemented with a multiplication of 1.1 to prevent round-off errors.

$$\text{d}(m_i, m_{i+1}) = \max(\text{speed}_{m_i}, \text{speed}_{m_{i+1}}) \cdot (t_{m_{i+1}} - t_{v_i}) \tag{4.3}$$

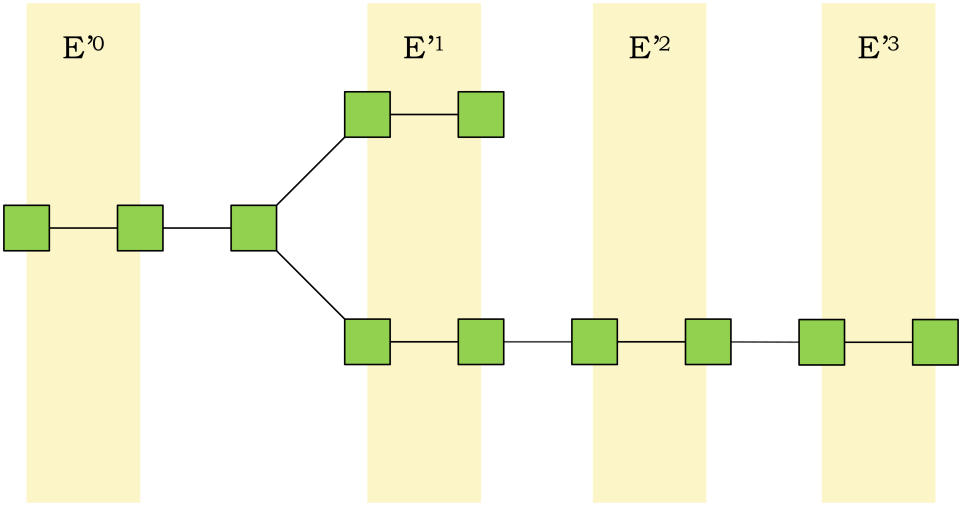


Figure 4.6: The spanning forest based on M in shown fig. 4.5. Only trees originating from E'_0 or E'_1 are kept.

4.3. Path generation

In this section we describe the path generation step based upon the related work in section 2.4. The methods for path extraction and closing of gaps as listed in algorithm 4.1 are first described followed by examples to clarify their working.

The getBestPath-function called in line 6 returns the best path P from the spanning forest T which starts at an edge in the provided set of edges E'^0 as shown in fig. 4.7. The cumulative fitness for a path is determined by accumulating the fitness-values of the fittest edge in $P \cap E'^i \cap T$ for each of the subsets of E'. The best path has the highest cumulative fitness of all traversable paths, regardless of whether or not they have a match in each of the subsets of E'. An empty path $P = \emptyset$ is returned when no valid path starts at the given set of edges E'^i.

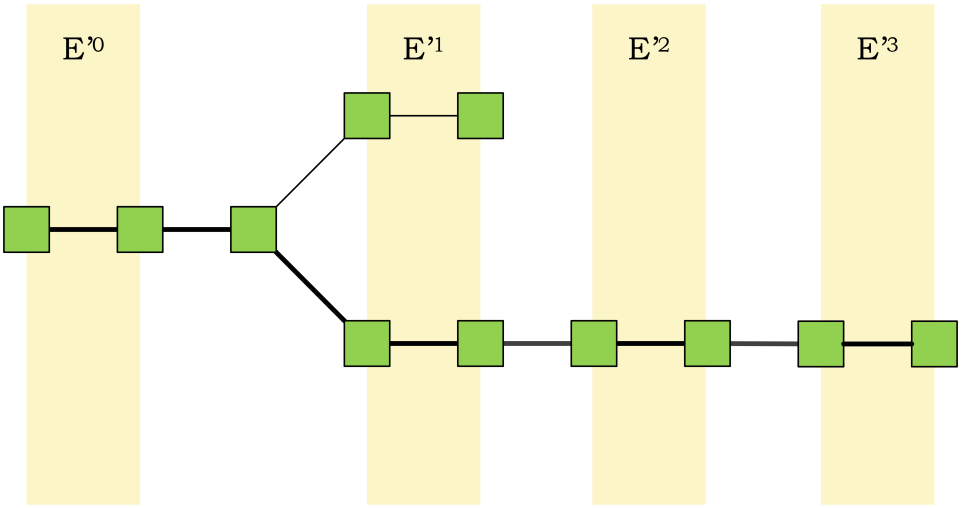


Figure 4.7: Example of the best path found in the spanning forest in fig. 4.6.

If an empty path was returned from the previous function, it is concluded that no path exists between E'^0 and E'^1 such as would be done for fig. 4.8. The conditional section from line 8 to line 10 creates a path by adding a single edge to the graph.

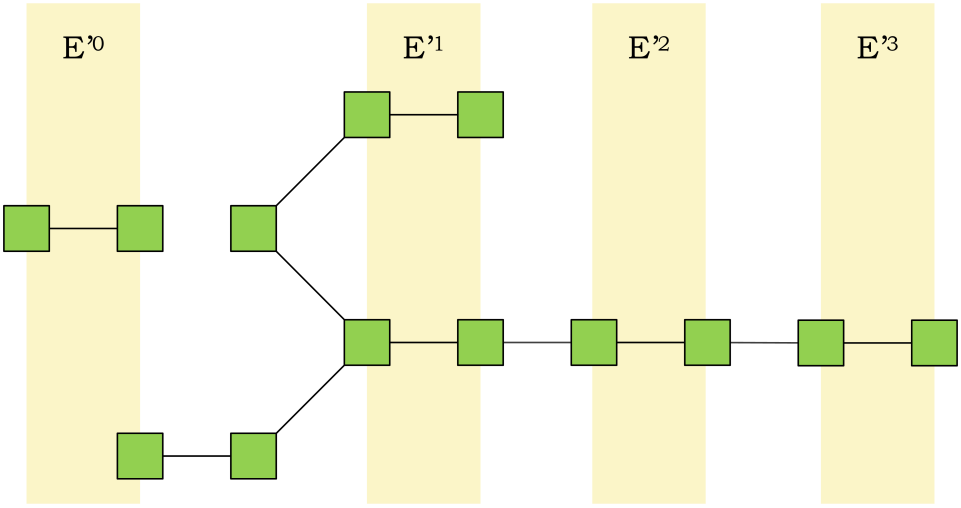


Figure 4.8: With one edge removed from fig. 4.1, no direct path from E'^0 to E'^1 is possible.

This search starts with a call of the getBestPath-function for E'^1 resulting in the best path P starting at E'^1. With the best path starting at E'^1 the path starts at the best edge p_0p_1 in E'^1 based upon cumulative fitness. A spanning forest from p_0p_1 in backward search towards m^0 is created to find a suitable location for the new edge as shown in fig. 4.9. This is done in the

getBestPath-function with p_0p_1 as start location, and $[m_1, m_0]$ in reverse order as indicators for search direction.

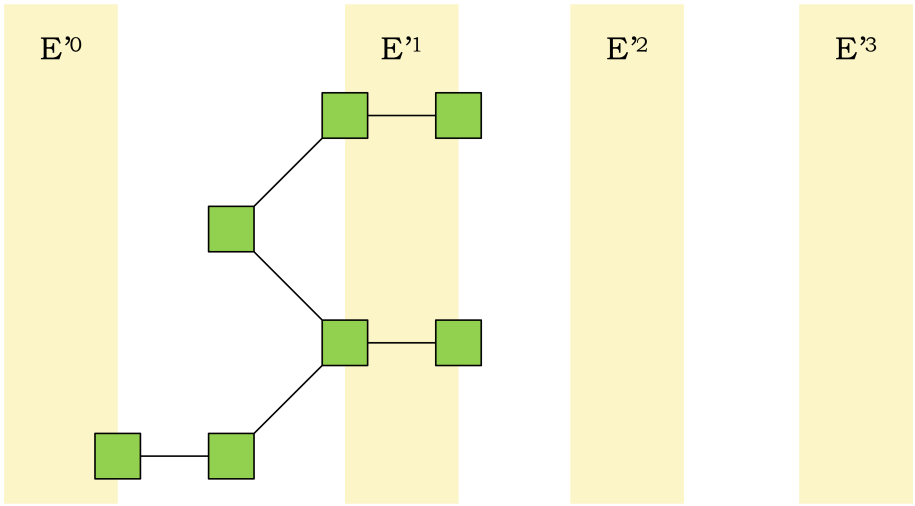


Figure 4.9: Example of a backward forest search from E'^1 to E'^0 on the graph in fig. 4.8.

In the bridgeSmallestGap-function an edge is then added to enable a traversable path from m_0 to m_1 as shown in fig. 4.10. An iteration over all possible combinations of nodes from the forward T and backward T' spanning forests is made to search for the best location for the connecting edge. The best combination ensures that the distance travelled from $e^0 \in E'^0$ to $e^1 \in E'^1$ over the nodes in the combination keeps within the limits set in eq. (4.3) and adds the shortest edge e^{01} in doing so. After adding the new edge e^{01} to the set of edges E in the graph G, a path P is returned which includes e^{01}.

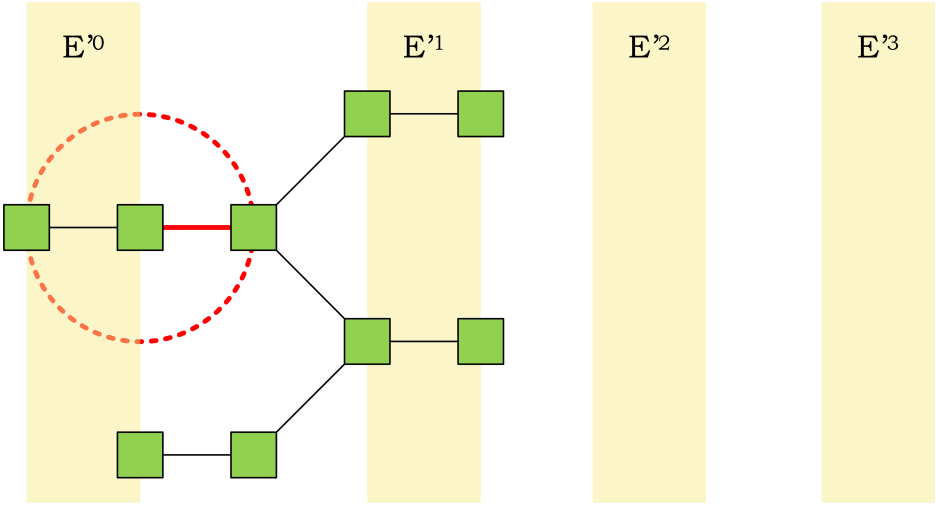


Figure 4.10: Example of the shortest jump possible between the forward and backward forests. Adding this edge to the map results in fig. 4.1.

In the initial situation when the map has not been initialized, it can be the case that either T or T' is empty. In case $T = \emptyset$, measurement m_0 is added to V as a new node and to T thereafter. Measurement m_1 is added to V as a new node and to T' as only node whenever $T' = \emptyset$. These initializations make sure that both T and T' are non-empty ensuring that an edge can be added to E to create a traversable path.

4.4. Path adjustment

In this section we describe the path adjustment step based upon the related work in section 2.5. The influencePath-method for path adjustment listed in algorithm 4.1 is first described followed by an example to clarify its working. This section first describes the addition of interpolated measurements to nodes and edges, followed by a description of position adjustment for nodes, and changes in map dynamics as managed in the edges of the map.

The influencePath-method called on line 12 in algorithm 4.1 adds interpolations of the provided measurements m_0 and m_1 to the nodes and edges on the path P. For each node v on the path a projection v' onto the line m_0m_1 is made as shown in fig. 4.11. An interpolation m' of the measurements m_0 and m_1 is constructed based on the relative position of v' on the line m_0m_1 using eq. (4.4) The midpoint v'_e of each edge e on the path is projected onto m_0m_1 so that an interpolated measurement can also be constructed for e using eq. (4.4). For each node or edge in the path, the interpolated measurement is added to the set of measurements M' contained in that node or edge as shown in fig. 4.12.

$$m' = \begin{bmatrix} \tau_{m'} \\ \text{pos}_{m'} \\ \sigma_{m'} \\ \angle_{m'} \\ \text{speed}_{m'} \end{bmatrix} = \frac{\text{d}(m_0, v')}{\text{d}(m_0, m_1)} \cdot \begin{bmatrix} \tau_{m_0} \\ \text{pos}_{m_0} \\ \sigma_{m_0} \\ \angle_{m_0} \\ \text{speed}_{m_0} \end{bmatrix} + \frac{\text{d}(v', m_1)}{\text{d}(m_0, m_1)} \cdot \begin{bmatrix} \tau_{m_1} \\ \text{pos}_{m_1} \\ \sigma_{m_1} \\ \angle_{m_1} \\ \text{speed}_{m_1} \end{bmatrix} \tag{4.4}$$

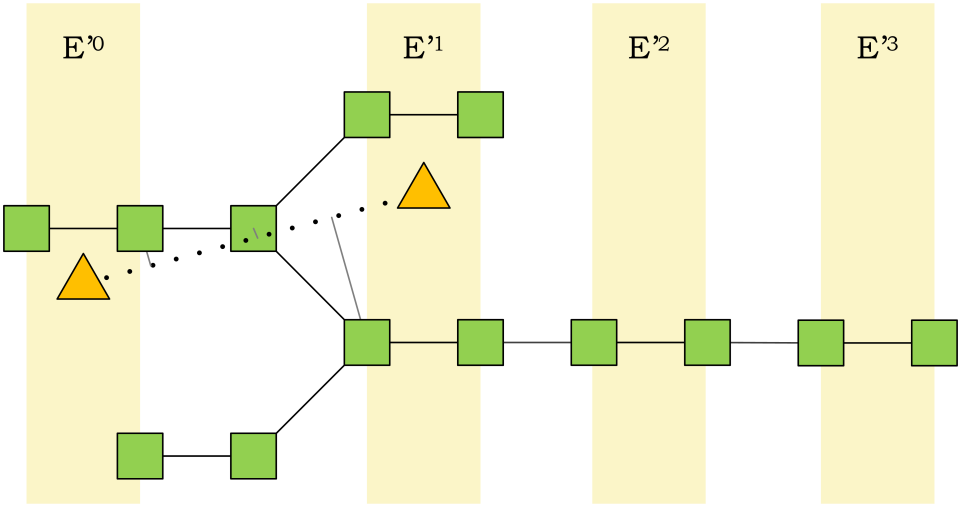


Figure 4.11: Example of projection of nodes in the travelled path.

The amount of measurements is limited by the parameter ζ to prevent map inertia as described in section 2.5. Only the ζ most recent measurements are stored in nodes and edges to calculate respectively node position and map dynamics. Based on tuning experiments the maximum number of measurements ζ is set to 125. The node position is determined by calculating an average position μ and standard deviation σ of measurements for a node as described in eq. (4.5).

$$\text{pos}_v = \frac{1}{|M'|} \sum_{m' \in M'} \text{pos}_{m'} \tag{4.5a}$$

$$\sigma_v^2 = \frac{1}{|M'|} \sum_{m' \in M'} \text{d}(m', v)^2 \tag{4.5b}$$

Line simplification is performed using t-test to see if two neighbouring nodes should be merged after the calculation of a node position. Given that the amount of measurements per node and position error are not known to be equal, an Welch's adaption of Student's t-test is used. Two nodes should be merged if they are believed to represent the same position by comparison using

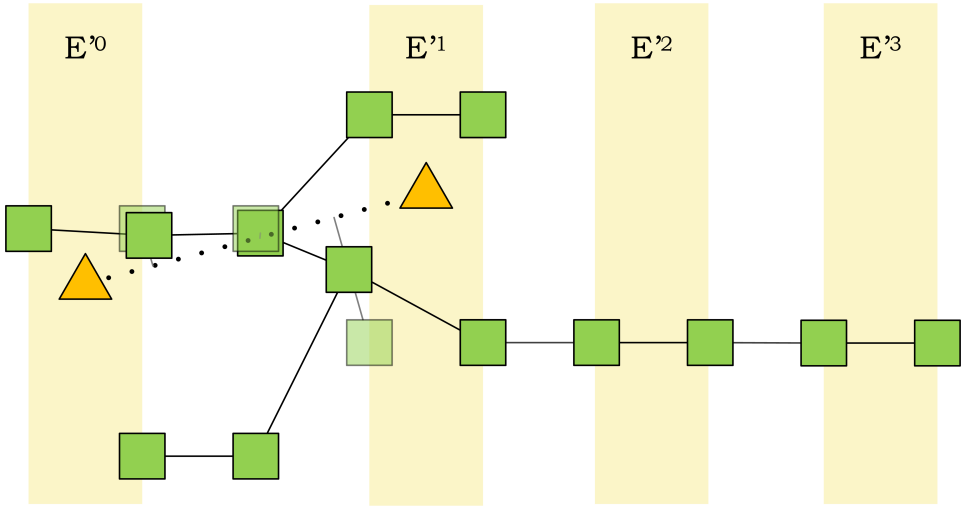


Figure 4.12: Influenced nodes move towards the line between measurements.

Welch's t-test. Given the sample mean (pos), standard deviation (σ), and amount of samples (|M|) for each of two nodes v_1 and v_2, the formulae for the values t and ν are given in eq. (4.6). The degrees of freedom ν in eq. (4.6) are approximated using the Welch-Satterthwaite equation shown ineq. (4.6b). These two values are used in a t-distribution to test for the hypothesis that both nodes are actually at the same position. The used threshold for Welch's t-test in the experiments is 0.01 based on tuning experiments. This form of line simplification ensures that nodes moving towards one another are eventually combined.

$$t = \frac{\text{pos}_{v_1} - \text{pos}_{v_1}}{\sqrt{\frac{\sigma_{v_1}^2}{|M|_{v_1}} + \frac{\sigma_{v_2}^2}{|M|_{v_2}}}} \tag{4.6a}$$

$$\nu \approx \frac{\left(\frac{\sigma_{v_1}^2}{|M|_{v_1}} + \frac{\sigma_{v_2}^2}{|M|_{v_2}}\right)^2}{\frac{\sigma_{v_1}^4}{|M|_{v_1}^2 \cdot (|M|_{v_1})} + \frac{\sigma_{v_2}^4}{|M|_{v_2}^2 \cdot (|M|_{v_2} - 1)}} \tag{4.6b}$$

Changes in map dynamics are calculated for each edge since movement of traffic is a change of location over certain path. An estimation of these dynamics is calculated using the set of measurements stored for each edge. Using eq. (4.7) the average interval between measurements μ_{τ_e} and variance thereof $\sigma_{\tau_e}^2$ are derived.

given $|M| \geq 2$:

$$\mu_{\tau_e} = \frac{1}{|M| - 1} \sum_{i=1}^{|M|-1} (t_{m_i} - t_{m_{i-1}}) \tag{4.7a}$$

$$\sigma_{\tau_e}^2 = \frac{1}{|M| - 1} \sum_{i=1}^{|M|-1} \text{abs}((t_{m_i} - t_{m_{i-1}}) - \mu_{\tau_e})^2 \tag{4.7b}$$

An expectation of the arrival of a new measurement is made using the estimation of the measurement interval. An edge is removed when the expectancy of a measurement arriving drops below a threshold value θ. The time within which a measurement should be updated before it is removed is called the time to live (TTL). The TTL is calculated using a combination of a dynamic value based upon the measurement interval statistics and an element linearly decreasing over the amount of measurements as shown in eq. (4.8). The dynamic time to live dTTL is determined to be μ_{τ_e}, plus the inverse cumulative distribution function CDF at x for a t-distribution with $\nu = |M|$ multiplied by σ_{τ_e}. Value x is determined as the moment when the inverse cumulative distribution

function for a t-distribution with $\nu = |M|$ drops below θ as stated in eq. (4.8b). The static time to live sTTL-component decreases to 0 as the amount of measurements reaches ζ. The time to live is then determined to be the maximum of dTTL or sTTL. Based on tuning experiments the static TTL value is set to 30 minutes and the dynamic TTL threshold θ to 0.01.

$$\mathrm{TTL} = \max(\mu_{\tau_e} + 1 - \mathrm{CDF}_{|M|}(x) \cdot \sigma_{\tau_e}, \left(1 - \frac{|M|}{\zeta}\right) \cdot \mathrm{sTTL}) \tag{4.8a}$$

$$x: \quad \mathrm{CDF}_{|M|}(x) = 1 - \theta \tag{4.8b}$$

The map G is cleaned up before being presented. Inspired by Cao and Crumm [7] the resulting map only shows edges with $|M|_e > \psi$ and dead-end streets with a length below ω are removed from the final representation. During tuning experiments the minimum amount of measurements ψ is set to 2 and the minimal dead-end street length ω to 100 meters.

5

Simulator

As described in section 1.4 a simulator is used for the experiments described in chapter 6 so as to provide an accurate ground truth and possible reproduction of experiments. An overview of the simulator is given in fig. 5.1 and shows how the world, map, and graph comparison are linked to the simulator. Simulation control and creation of the ground truth world representation is performed by the Road Block Simulation tool (RBS) as described in section 5.1. From measurements of cars in the world a map is developed using ROMA as described in section 5.2. Comparison of the world and the developed map is described in section 5.3. The entire software package is implemented in Java. The source code for the algorithm and graph comparison can be found in appendix C.

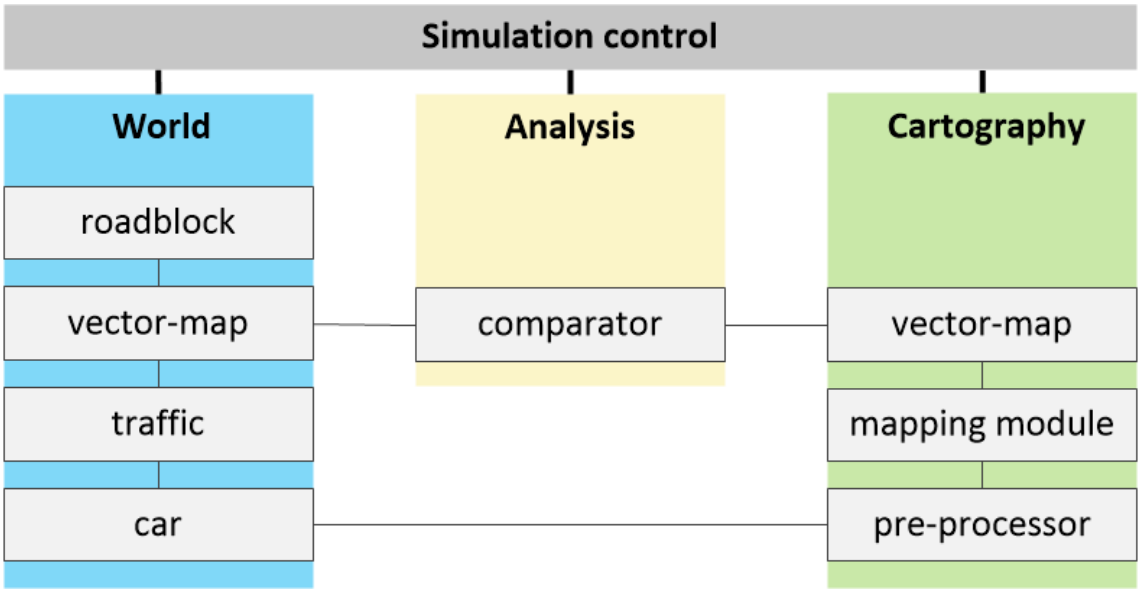


Figure 5.1: Overview of the Road Block Simulation tool (RBS) with modules for ROMA and graph comparison added.

5.1. World (RBS)

The Road Block Simulation tool (RBS) is developed by Joris Scharpff at the Dutch research company Almende. This tool was developed to demonstrate the effects of road maintenance blockades on traffic flow of a road network using discrete time steps of 1 second. At the start of a simulation, a map is loaded in XML-format as world on which a stable amount of traffic is simulated and roadblocks are placed. An agent based approach is applied in the chosen traffic model to simulate individual cars which drive the shortest path between two randomly chosen points on the map. Roadblocks can be placed on specific segments of the map for a given period of time. A visualization module provides a view of the blockades and where they are placed, as well as the

effects on traffic as it traverses the map. RBS is suitable for the experiments in this research for simulating traffic on a ground truth map and removal of roads on that map at specific moments.

5.2. Cartography (ROMA)

The cartography module shown in fig. 5.1 contains the mapping functionality of ROMA as described in chapter 4 and also the derived map. The pre-processing step from ROMA is disconnected from the main mapping module to allow for parallel pre-processing. A pre-processor is instantiated per simulated car which also adds noise to the actual position to simulate GPS measurements. Based on the statement by Bruntrup et al. [5] that GPS errors can be less than 15 meters as of the year 2000, the simulated GPS error σ is set to 15 meters. To prevent measurements to jump back and forth along the actually driven path because of this artificial GPS error, a GPS measurement interval of 8 seconds (distance of 4.44σ at a speed of 30 km h^{-1}) is taken. These measurements are then pre-processed and the pre-processor calls the mapping module when a set of measurements of the required size is available. The mapping section then performs the other three steps found in ROMA: path estimation, path generation, and path adjustment. These steps influence and use the map stored in the cartography module within which the derived measurements are finally stored. The visualization module of ROMA provides a view of the developed map.

5.3. Graph comparison

The module for graph comparison executes the common edge subgraph comparison described in chapter 3. Graph comparison is not executed every time step since the method of comparison has a computational complexity of $O(|V|^3)$. A comparison interval of 30 seconds is chosen to provide insight in the development of map quality over time. Before actual comparison the world graph is simplified to represent a similar type of graph as the map graph. The world graph also contains edges with shape-points in between so that nodes only represent intersections and road ends. The edges from the world graph are therefore split so that individual edges are present between all shape-points. The world graph is also filtered using the same post-processing filter applied to the map graph: dead-end streets with a length of < 100 meters are removed as are roads travelled only once. With the two graphs being equal in representation the graph comparison as described in chapter 3 is then applied. The resulting statistics in terms of precision, recall, and F-score are written to a separate comma-separated file such that a time-stamped progress of map quality is stored.

6

Experimental set-up and results

This chapter describes the experiments testing the dynamic behaviour of the algorithm, ROMA, as described in chapter 4. The dynamic behaviour is investigated in two experiments where traffic intensity and road topology changes are independent variables as shown in fig. 6.1. The values for these independent variables and other parameters are described in section 6.1. Experiment 1 investigates the dynamic behaviour of ROMA after introduction of new roads as described in section 6.2 whereas experiment 2 investigates the dynamic behaviour of ROMA after removal of roads as described in section 6.3. Measurements of dependent variables follow the methodology described in chapter 3.

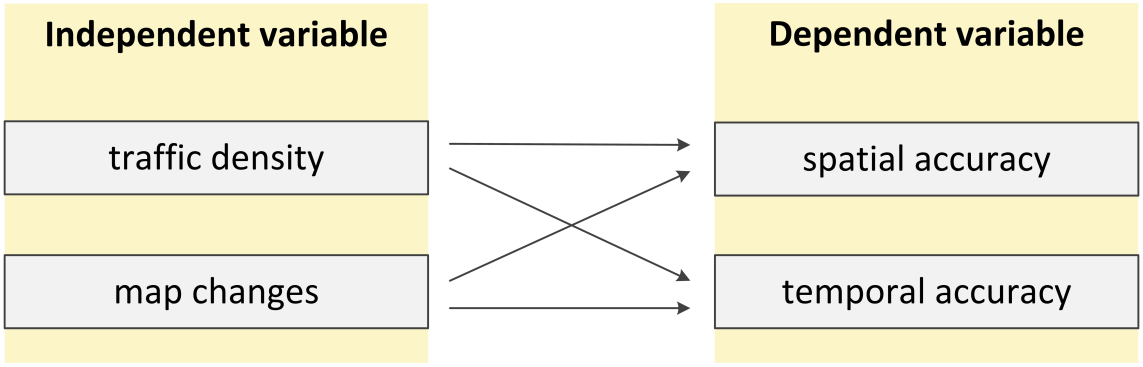


Figure 6.1: Structural diagram for the experimental design. The predictive relationships from independent to dependent variables are explored in the experiments as well as the correlation between spatial and temporal accuracy.

6.1. Experimental set-up

This section describes the motivation and values used for the independent variables followed by a brief summary of algorithm and simulator settings unchanged during the experiments provided for completeness.

The simulator described in chapter 5 provides position measurements with added noise from simulated traffic on a known ground truth. From these measurements a map is generated with ROMA which is regularly compared to the world resulting in a series of spatial accuracy values. The ground truth world graph chosen for the simulations is the town of Zoeterwoude-dorp, located in the urbanised western part of the Netherlands. This world is based upon data obtained from Open Street Map and filtered so it only contains the roads shown in fig. 6.2. Traffic simulated on the world graph adheres to the speed limits for the different types of roads present in the map.

The independent variables for the experiments are the traffic density and topological changes in the world. Traffic intensity is chosen so as to give a reasonable representation of traffic expected in Zoeterwoude-dorp. Topological changes are the introduction of new roads in experiment 1 and

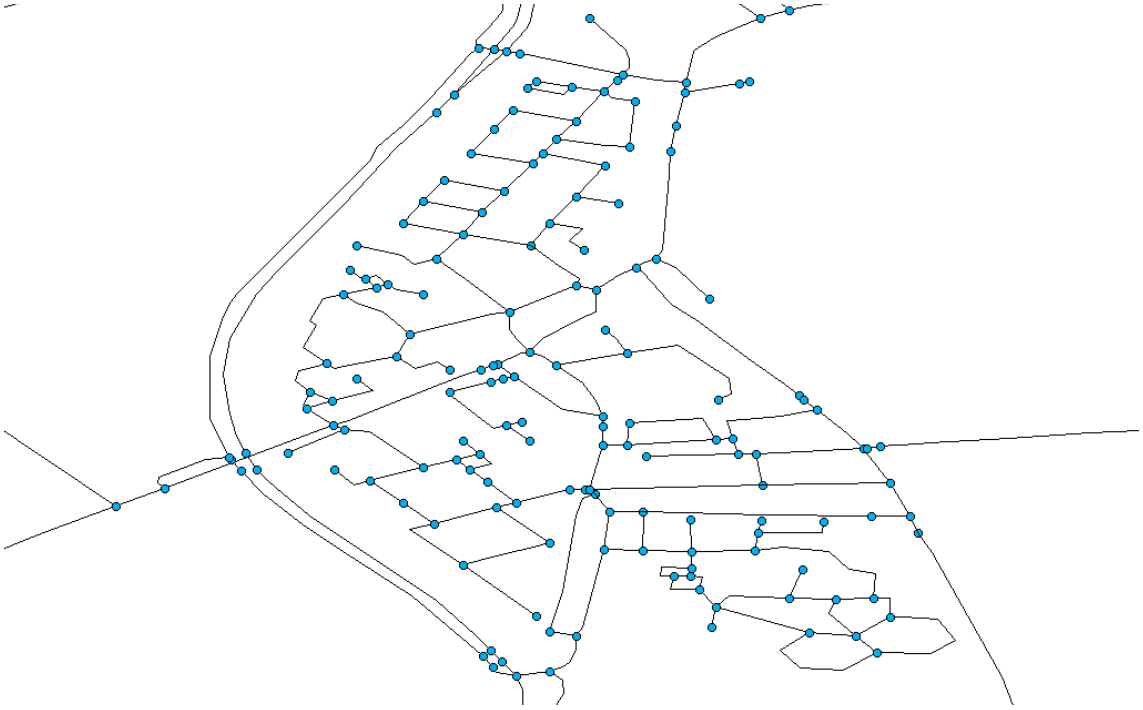


Figure 6.2: The undirected world graph representation of Zoeterwoude-Dorp as used in the experiments. Intersections and dead-ends in the dataset are indicated with a blue dot.

the removal of roads in experiment 2. The changes are introduced at the start of each experiment and not altered during map recovery.

A reasonable amount of simulated traffic is estimated using the number of inhabitants of Zoeterwoude-dorp multiplied by the average car ownership in the region. Zoeterwoude-Dorp had 4.514 inhabitants in 2012 [9] and 418 cars were owned per 1000 in the province of Zuid-Holland in 2014 [8]. Multiplication of these numbers gives an estimated total of 1887 cars present in Zoeterwoude-Dorp. For experiment 1, a broad spectrum of traffic intensities is chosen so as to provide insight in algorithm performance. For the simulated numbers of cars in experiment 1, the following values are chosen: 50, 100, 200, 500, 1000, 2000, and 4000. Following the results of experiment 1, traffic intensities for experiment 2 are set to: 1000, 2000, and 4000. For experiment 2 the number of roads blocked is set to 5, 10, and 50 percent of the total length of roads. We expect that the lower values of 5 or 10 percent of roads blocked cover regular scenarios where roads are blocked for construction works. The value of 50 percent is chosen to provide insight in algorithm performance under more extreme changes in topology.

The simulator and ROMA contain variables that are not changed during the experiments. The values for these variables for the simulator are defined in chapter 5 and in chapter 4 for ROMA. A summary of these settings is provided in table 6.1 for the completeness of this chapter.

6.2. Experiment 1: road introduction

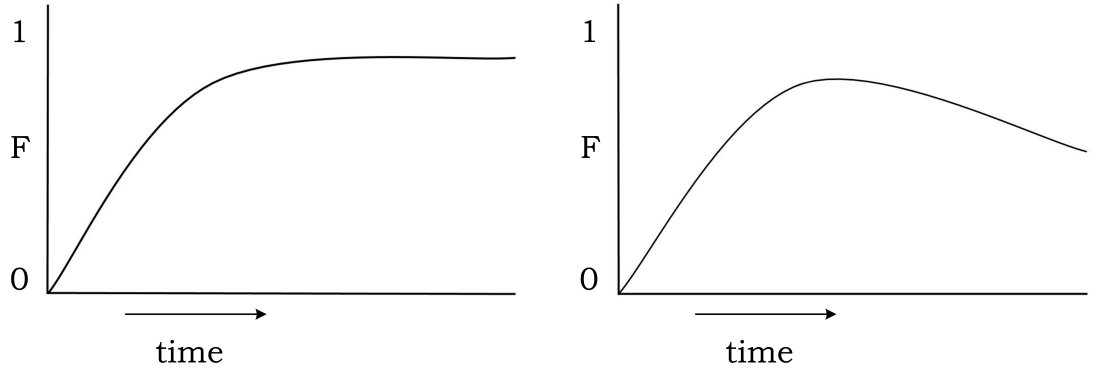
Experiment 1 investigates the dynamic behaviour of ROMA after introduction of new roads. Traffic density for different runs of the experiments is set to 50, 100, 200, 500, 1000, 2000, and 4000 simultaneously simulated cars over a maximum period of 240 minutes (4 hours). The initial vector-map is an empty graph and no roadblocks are added in this experiment. This section first describes the expectations of the experiment followed by the quantitative results and a qualitative analysis of the development of the map over time. The experiment with a traffic density of 2000 cars is used as example. An overview of other experimental data is available in appendix A.

symbol	variable	value
	Time step size	1 sec
	Traffic model	random
	Max. simulated time	4 hours
σ	simulated σ	15 meters
α	σ threshold	∞
	GPS interval	8 sec
β	max. measurement distance	300 meters
n	look-ahead size	4
γ	min. fitness	0.01
	max. distance multiplier	1.1
ζ	max. # measurements	125
	node equality threshold	0.01
sTTL	static TTL	30 min
θ	TTL threshold	0.01
ψ	min. # measurements	2
ω	min. dead-end length	100
	compare interval	30 s

Table 6.1: Settings for ROMA and RBS as described in chapter 4 and chapter 5 respectively.

Expectations

The expected quantitative results for experiment 1 as discussed in section 1.4 are shown in fig. 6.3. Given that ROMA learns about the world we expect the behaviour of the quality function to be similar to training a statistical model. In the ideal setting shown in fig. 6.3a would have the map quality increase towards an asymptote at a 100 % match of the world. Figure 6.3b shows how incremental noise influences the map quality over time by decreasing the map quality.



(a) Ideal course without noise

(b) Expected course with increasing noise

Figure 6.3: Expected course of map quality following road introduction in an ideal setting fig. 6.3a and with noise fig. 6.3b

Quantitative results

The expected maximum value or start of map decrease due to noise did not clearly influence the simulations with low amounts of traffic (50, 100, 200) in a period of 240 minutes of simulated traffic. A decline in map quality is visible before the end of the simulation for traffic amounts above 200. These simulations were stopped at a moment before 240 minutes of simulated traffic. Overfitting is clearly observed in plots for traffic amounts of ≥ 1000 such as in fig. 6.4 for 2000 simulated cars. A climb in the balanced F-score is visible before the value of 0.55 at 5.5 minutes into the simulation with 2000 cars. The balanced F-score declines after 5.5 minutes influenced mostly by the decline in precision. The recall value keeps relatively stable with a value above 0.55 after the moment of overfitting.

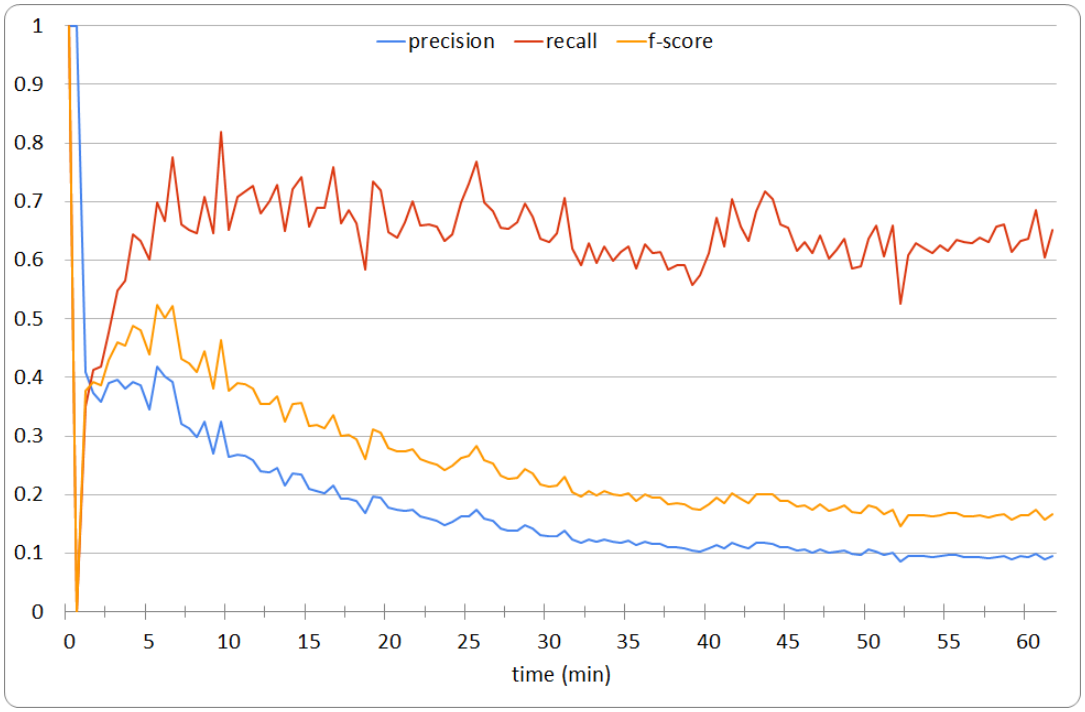


Figure 6.4: Course of map quality over time for 2000 simulated cars in experiment 1. Note that precision decreases after 5.5 minutes while recall stays rather stable.

The moment of overfitting and maximum map quality are determined for traffic amounts of 1000, 2000, and 4000 and shown in table 6.2. From the data we know that for lower traffic amounts the moment of overfitting occurs at a later time and a higher map quality is seen for maps with a later moment of overfitting.

traffic	τ (min)	prec.	rec.	F
1000	6.5	0.515	0.791	0.624
2000	5.5	0.463	0.750	0.573
4000	3.5	0.350	0.730	0.473

Table 6.2: Estimated moment of overfitting for different amounts of traffic.

Qualitative evaluation

The qualitative evaluation of multiple simulations showed the development of a map representing the world. An example of the development over time is shown in the simulation with 2000 simultaneous cars in fig. 6.5. Web-like structures are seen to be developing which seem more present around the intersections on the map. After the moment of overfitting, the development of web-like structures around the position of roads in the ground truth map is clearly visible in fig. 6.5d.

6.3. Experiment 2: road removal

Experiment 2 investigates the dynamic behaviour of ROMA after removal of roads from the world. Traffic density for different runs of the experiments is set to 1000, 2000, and 4000 simultaneously simulated cars given the clear peak in measurement quality observed in experiment 1. Different amounts of road are removed from the world at the moment in the simulation at which the peak in measurement was observed in experiment 1 as summarized in table 6.2. The length of roads removed is set to 5, 10, or 50 percent of the total road length as motivated in section 6.1. The simulation from experiment 1 without road removal is used to find the moment of recovery in a qualitative comparison of map qualities. This section first describes the expectations of the

(a) t = 1 min, prec. = 0.377, rec. = 0.196, F = 0.258

(b) t = 3 min, prec. = 0.318, rec. = 0.388, F = 0.350

(c) t = 5.5 min, prec. = 0.458, rec. = 0.703, F = 0.554

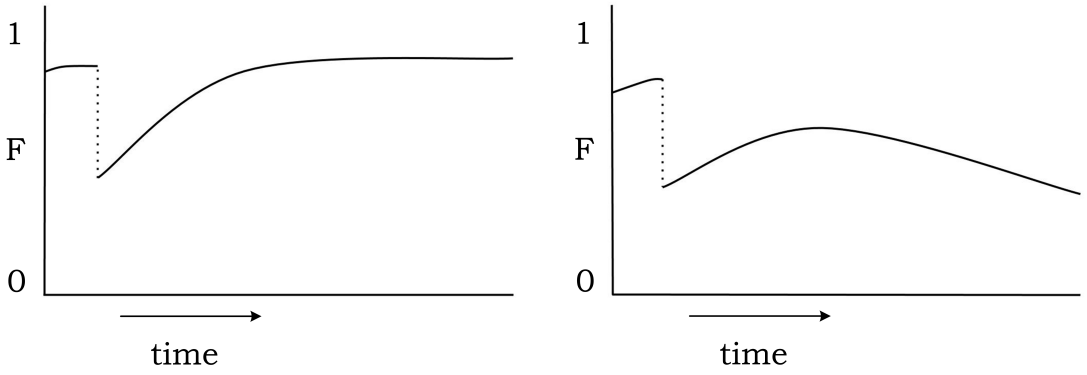
(d) t = 20 min, prec. = 0.180, rec. = 0.645, F = 0.281

Figure 6.5: Visualisations of the map at several moments during simulation with 2000 cars. Figure 6.5a and fig. 6.5b show the map before the peak map quality. Figure 6.5c shows the map at maximum quality and fig. 6.5d shows the formation of webbing after overfitting.

experiment followed by the quantitative results and a qualitative analysis of the development of the map over time. In this section 6.3 only precision is used as the inspected measure for map quality as described in the expectations. The experiment with a traffic density of 2000 cars is used as example. An overview of other experimental data is available in appendix B.

Expectations

The expected quantitative results for experiment 2 as discussed in section 1.4 are shown in fig. 6.6. Map quality is expected to instantly decrease following the removal of roads in the world. In the ideal setting shown in fig. 6.6a map quality would then increase towards an asymptote at a 100 % match of the world. Figure 6.6b shows how incremental noise influences the map quality over time by decreasing the map quality. Precision is used as the inspected measure for map quality since the expected drop is in the number of roads on the map representing the world.



(a) Ideal course without noise

(b) Expected course with increasing noise

Figure 6.6: Expected course of map quality following road removal in an ideal setting fig. 6.3a and with noise fig. 6.3b

Quantitative results

The simulation results for all three simulated traffic densities show similar behaviour and generally follow the pattern shown for 2000 simulated cars in fig. 6.7. After the introduction of roadblocks a steep drop in precision is visible compared to the simulation without roadblocks. Recovery of the precision value up to the values of the simulations without roadblocks is also visible. In the simulation with 50 percent of roads blocked this recovery is visible as a near-horizontal line in the graph. This simulation also reaches the baseline precision values without blockades later compared to the simulations with only 10 and 20 percent of the roads blocked. Table 6.3 shows that the times of recovery differ per amount of roadblocks but show a decrease in relation to the simulated traffic density.

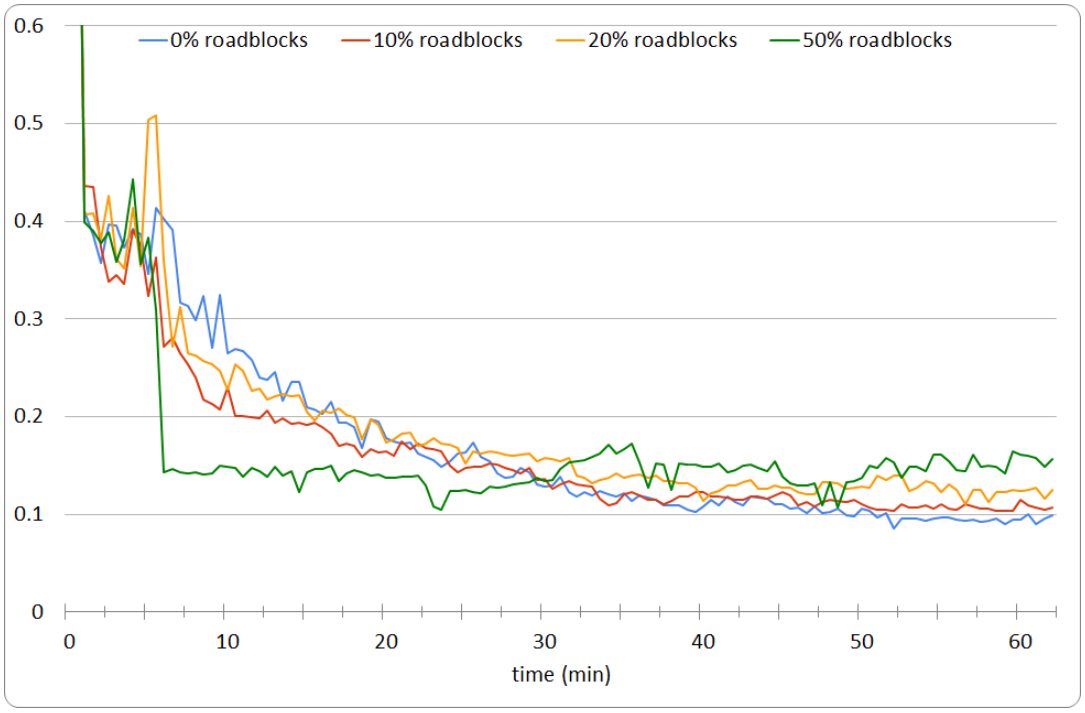


Figure 6.7: Course of precision over time for different percentages of roads blocked with 2000 simulated cars in experiment 1. Roadblocks were added at 5.5 minutes.

	τ (min)		
block (%)	1000 cars	2000 cars	4000 cars
10	28.5	15.5	4.5
20	17.0	8	6
50	28.5	24	out of range

Table 6.3: The estimated time of recovery for different amounts of roadblocks.

Qualitative evaluation

The qualitative evaluation of multiple simulations showed the removal of several roads from the map. An example of the development over time is shown in the simulation with 20% roadblocks and 2000 simultaneous cars in fig. 6.8. The removal of roads is visible after the start of experiment 2 and is clearly visible in the lower left portions of the map. Road removal continues after the determined moment of recovery. The development of web-like structures also observed in experiment 1 also visibly contributes to the overall map as shown in fig. 6.8d.

(a) t = 5.5 min, prec. = 0.508

(b) t = 13.5 min, prec. = 0.223

(c) t = 21.5 min, prec. = 0.184

(d) t = 36.5 min, prec. = 0.137

Figure 6.8: Visualisations of the map at several moments during a simulation with 2000 cars and 20 % roadblocks. Figure 6.8a shows the map at maximum quality. After fig. 6.8c shows the map at the moment of recovery. Figure 6.8d shows that even more roads have been removed after another 15 minutes. Road removal is most clearly visible in the lower left area off the map.

7

Discussion

In this chapter we discuss the research in this thesis based on insights from experiments. In section 7.1 we discuss the simulator described in chapter 5. The metrics and evaluation criteria described in chapter 3 are discussed in section 7.2. In section 7.3 we discuss ROMA as described in chapter 4 using the experimental results from chapter 6. After scrutinizing the developed algorithm and metrics we conclude this chapter with a discussion of general applicability of ROMA in section 7.4.

7.1. Simulator

A simulator as described in chapter 5 was chosen for the experiments in this thesis to obtain a reliable ground truth to compare the performance of ROMA against. Although it is known that map geometry changes over time [5], many authors [7, 18, 20] do compare their work against commercially available maps. We believe that it can not be assumed that these maps and the maps created from GPS-traces represent the same ground truth because of the mentioned geometry changes. To ensure a comparison of the developed map against a ground truth we therefore simulated the measurements directly from an absolute and known ground truth.

The approach taken to traffic and measurement simulation was straightforward and simple. Traffic was simulated by selecting the shortest available path between two randomly chosen locations on the ground truth for each simulated car. Although a smooth adjustment function to maximum speeds at different streets was used, no interaction between cars due to congestion was modelled. The speed of simulated cars was also not adjusted when turning a corner. The simulated GPS-accuracy in the simulator was set to a constant σ which was the maximum value suggested by [5]. With this rather straightforward approach to simulation of traffic and GPS signals we were able to provide a set of measurements for ROMA. With traffic densities heavier on larger roads we believe that even with a simple traffic model a decent variety of characteristics was visible. Especially after the removal of roads, the simulation indicated that traffic density on minor roads increased after a major road was closed. We do advise some adjustments to the traffic model since ROMA could benefit from a higher resolution of measurements in corners and intersections. We therefore encourage the use of a more realistic traffic model in future research using an absolute ground truth especially with regard to traffic speed adjustment in corners and on intersections. Values for GPS-accuracy also have to be adapted to more recent values given that [5] is 10 years old. We advise to use the accuracy values from existing datasets to provide realistic accuracies in simulations.

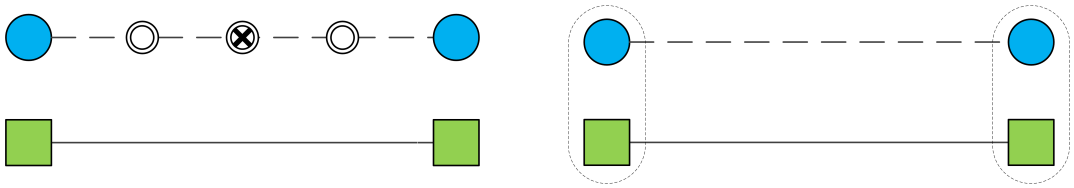
7.2. Metrics

Graph comparison as described in chapter 3 was used to describe the quality of the map developed by ROMA. The approximation of true positives using the common edge subgraph of the map and world graphs is an attempt to compare both topology and geometry of the two graphs. We managed to apply a graph-theoretic approach to map comparison where Biagioni and Eriksson [3]

deemed this impossible. We have dealt with the problem of the lack of geographical information in graph theoretical approaches. By first matching nodes in both vector-maps based upon their geographical distance we obtained promising results. From the course of map quality over time we were able to extract the moment of recovery which is used as indicator for temporal accuracy of ROMA.

The results of the experiments in chapter 6 show that the map quality does not have a smooth course over time in either precision or recall. The observed simultaneous and sporadic jumps in both precision and recall can be explained by the fact that node matching is performed for each quality sample taken. If the node matching from map to world is changed between two quality samples, the resulting edges in the set of true positives is also influenced. This influences both these quality measures because true positives are present in the formulae for both precision and recall. We propose to apply an approximation of the Maximum Common Edge Subgraph (MCES) to prevent these jumps in map quality. An MCES-algorithm can apply the distance between map and world as a heuristic for an appropriate match instead of comparing only the nearest possible nodes to be matched. Polynomial-time approximations of algorithms for the MCES problem exist [22] and can be used to provide an improved approximation. An improvement to geographic comparison can be made by comparing the shape of matched edges for the used similarity matrix S. The currently applied easy method of comparison of edge lengths does not penalize if two edges are the same length but not the same location.

The method of Biagioni and Eriksson [3] based upon holes and marbles as published in 2012 compares a map on different grounds. The marbles-method of Biagioni and Eriksson as described in section 2.6 is briefly compared in this section to our method described in chapter 3. In the holes and marbles approach of Biagioni and Eriksson matches marbles and holes only if geometry is exactly the same making it rather vulnerable to displacements and geometry differences between map and world. The example in fig. 7.1a shows how holes are placed on the world, but not on the map which is displaced in regard to the world. Since our method does work with displaced maps as shown in fig. 7.1b, we have provided a method with a more forgiving attitude towards geometrical differences. We believe that because of this forgiving attitude, our method for vector-map comparison is better suited for dynamic map generation where both geometry and topology can change.

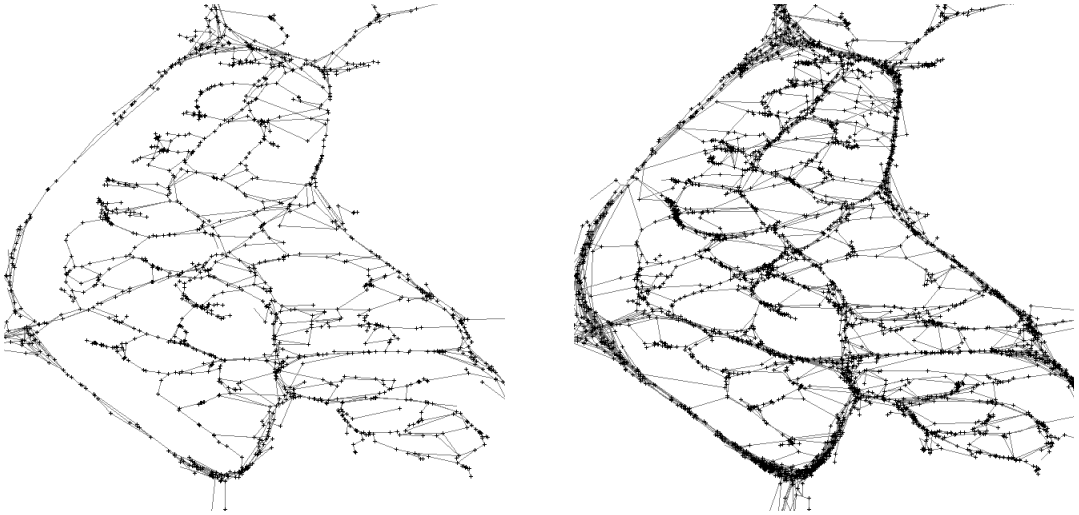


(a) Measuring using Biagioni and Eriksson [3]. (b) Measuring with the method from chapter 3.

Figure 7.1: Comparison of displaced roads. The method by Biagioni and Eriksson [3] lacks overlap and therefore obtains no matches between placed 'holes' and 'marbles'. The method from chapter 3 matches the nearest nodes from map and world resulting in a common edge.

7.3. Algorithm

The robust online map-generation algorithm (ROMA) described in chapter 4 defines an on-line approach to trace merging. It is an innovation on related work by addressing the issue of map dynamics where other authors assume a never-changing world in their articles. The maximum observed recall in experiment 1 was generally above 0.8 as shown in table 7.1. This indicates that most of the roads on the world were represented in the map, creating a reasonably navigable map. Recall was below precision in all configurations of experiment 1, showed maximum values below 0.6, and also a decline over time during the simulations. In the visual inspections of map development, web-like structures were visible and increasing in size over time as shown in fig. 7.2. We believe these structures have made a great attribution to noise visible through the precision values. The presence of noise furthermore seems inversely correlated with the amount of simulated traffic as can be seen in table 7.1.



(a) t = 5.5 min, prec. = 0.458, rec. = 0.703, F = 0.554

(b) t = 20 min, prec. = 0.180, rec. = 0.645, F = 0.281

Figure 7.2: Visualisations of the map at several moments during experiment 1 with 2000 cars. Note the increasing presence of web-like structures around intersections.

traffic	max. precision	max. recall
50	0.558	0.928
100	0.524	0.676
200	0.578	0.834
500	0.558	0.928
1000	0.515	0.864
2000	0.476	0.819
4000	0.387	0.781

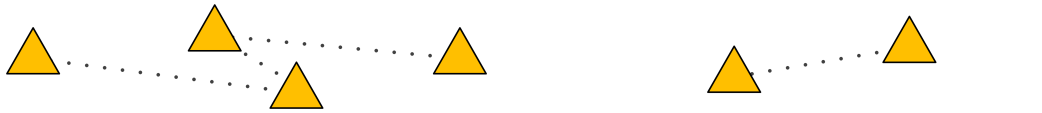
Table 7.1: Maximum precision and recall in experiment 1 for different amounts of traffic.

The observed noise does not greatly influence ROMA as a proof of concept for dynamic mapping. Based upon the observed noise we will discuss possible improvements for ROMA in this section. We start by discussing how better filters can improve the performance of ROMA in section 7.3.1. We then discuss how more conservative map generation can decrease noise in section 7.3.2. We conclude by a theory on the effects of conservative edge removal on noise section 7.3.3.

7.3.1. Pre-processing

The shape of the web-like structures shown in the experimental results suggest that the applied filter does not smooth the trace well. Due to the lack of a decent model for traffic or GPS we used a GPS measurement interval of 8 seconds to prevent measurements to jump back and forth along the driven path as seen in fig. 7.3a. Measurements which stray far from the driven path are not filtered out by the current methods resulting in the creation of improbable road shapes. Based on the results we conclude that these outliers negatively influence map precision. In the current implementation, we only sample GPS measurements every 8 seconds and throw away all location data in between. To improve on measurements jumping back and forth along the driven path, we suggest to sample more often and average over multiple samples to smooth the trace as shown in fig. 7.3b. Given that GPS error follows a 2-dimensional Gaussian error [27], we believe that such a combination results in a better approximation of the actual location.

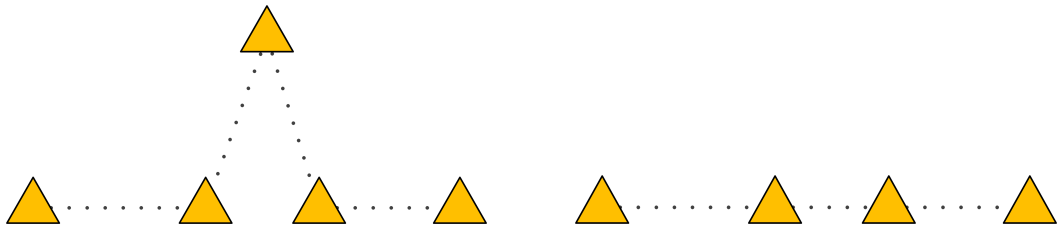
More extreme outliers such as shown in fig. 7.4 might be thrown away by removing measurements which make a large change in direction. Such a filter based on direction change is applied in related work [20, 31], removes extreme outliers and splits the trace by doing so. This direction filter should be applied before measurement averaging to better recognize extreme outliers.



(a) Jumpy behaviour GPS measurements. (b) Measurement averaging smooths the trace.

Figure 7.3: Jumpy behaviour of GPS measurements along a trace might be solved by averaging.

We believe that the GPS position trace can be smoothed before map matching and merging by application of a direction filter and measurement averaging in that order.



(a) Example of extreme outliers due to GPS error. (b) Removal of an extreme outlier benefits the trace.

Figure 7.4: Removal of extreme outliers smooths a trace of GPS measurements.

7.3.2. Map Expansion

The speed of map expansion depends on the availability of a valid path between two measurements. If path estimation does not yield a valid path a new edge is generated which expands the map. This can either be because no edges were matched to the measurements or no path within the maximal travelled distance between matched edges exists. Edge matching is influenced by the minimal-fitness-setting ($\gamma = 0.01$) of which higher values result in less edges matched and more new edges added.

The maximum travelled distance in ROMA is based on the work of Pfoser and Jensen [21] who assume the measured position to be the absolute truth. We applied this method to uncertain GPS measurements by only calculating travelled distance on the map, without counting the distance between measurement and the matched edge. We also multiplied the maximum travelled distance by 1.1 to prevent round-off errors due to conversion from the float to double type in JAVA. A higher multiplier would result in more valid paths between matched edges.

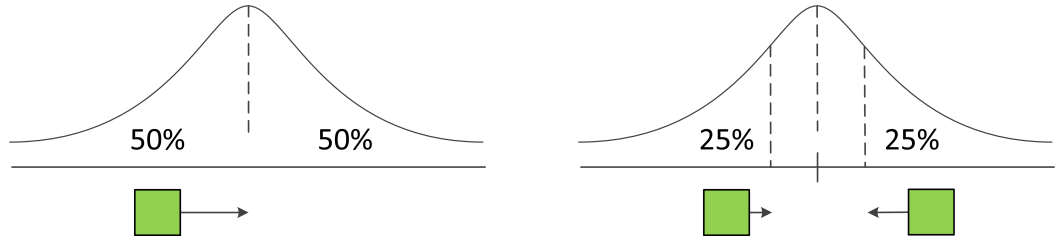
In a more progressive setting where new edges are quickly created, recall will probably increase since all roads in the world have a decent change to be added. However more new edges also increase the amount of false positives which negatively influences the precision of the map. In a conservative setting the opposite is true and recall suffers at the cost of increased precision. Depending on the requirements of the end-user the value of γ and the multiplier for maximum travelled distance can be adjusted to select a setting on the spectrum between conservative and progressive map-generation.

7.3.3. Edge removal

The speed of edge removal depends on both the dynamic and static time to live (TTL) settings. The dynamic TTL would ideally present an estimate of road availability based on the average and standard deviation of collected measurement intervals. Since an edge is created with only one measurement, we added an static TTL value with decreasing influence when more measurements are added. The used value of 30 minutes was appropriate for a traffic density of 50 cars where the web-like noise was not as present as with higher traffic densities. In the example of a traffic density of 2000 cars, the moment of overfitting was 5.5 and the moment of recovery occurred within 30 minutes for all roadblock sizes. Since these times of adjustment to new situations are all well below the static TTL value, we believe that static TTL can contribute greatly to the observed clutter on the map.

With this insight we have developed a theory on how the web-like structures form on the map

and contribute to decreasing precision values. Because of the Gaussian distribution of GPS error [27], the general believe is that road centrelines obtained from such signals will converge to the actual location on the world [5, 11, 23, 24]. The example in fig. 7.5 shows how newly added edges can contribute to the observed web-like structures in 1-dimension representing the cross section of a road. The location of a first edge on the cross section will converge towards the centre of the road because of the Gaussian distribution of GPS error as shown in fig. 7.5a. This changes as soon as a measurement of the same road causes the creation of an extra edge when it is not matched to the original edge. Two edges representing the road have to share the incoming measurements and will probably converge towards symmetrical positions around the actual centreline as shown in fig. 7.5b. In this situation, all measurements between the two edges are divided between the two edges, and outside measurements mostly contribute to the nearest edge. Here again, outliers can create new edges which make the cross section more crowded with edges and measurements are distributed amongst these edges. With this decreasing amount of measurements per individual edge, the static TTL value contributes more to the estimation of time to live for each edge. With a relatively high static TTL edges remain on the map for a longer time compared to when dynamic TTL would determine the time to live. These remaining edges are observed as web-like structures on the map and are more present when more measurements per time are added such as is the case with higher traffic density. Since only one edge per road on the world will be counted as true positive, all others contributed to a decrease in precision which was observed for higher traffic densities.



(a) One edge converges towards the road's centreline.

(b) Two edges converge on different positions.

Figure 7.5: Example of points of convergence for 1 and 2 edges representing the same road.

The web-like structures can possibly be merged during algorithm execution in a similar style as the merging of nodes defined in section 4.4. Because of differences in connectivity of nodes in edges a solution is less trivial than the simple merger of shape points applied within ROMA. We suggest to further research the best method of edge merging to also improve the dynamic nature of ROMA. Regarding the static TTL value we advise to apply a value related to the overall traffic density as initial setting.

7.4. Generalisation

This section discusses the general applicability of ROMA as well as the developed metric. ROMA was developed to be develop a map (1) based on simulated GPS-measurements (2) without a priori map, (3) using direct communication, and (4) centralized processing. In this section we explore how ROMA can be applied to settings with (1) real world measurements, (2) existing maps, (3) indirect communication, or (4) decentralized processing. We start this section with the general applicability of the developed metric.

The metric as described in chapter 3 was applied to describe the quality of the map developed by ROMA. We believe that we can even use the metric as generic similarity measure for two vector-maps. The balanced F-score provides a harmonic mean which indicates that both precision and recall are equally important. Precision and recall are both calculated as the amount of true positives relative to the total length of respectively map and world. When neither of the vector-maps is an absolute ground truth, the balanced F-score is thus essentially a similarity measure between two vector-maps.

The simulator described in chapter 5 was used to simulate GPS measurements from car traffic. With the adjustments proposed in section 7.3 we believe that ROMA can also be used on real-world

data. With only the proposed measurement averaging and outlier removal as filter we furthermore believe that the dependency on GPS or the use of cars is no longer essential. Even data without information on the accuracy can be used with a few small adjustments to parts of the filter and the fitness function where measurement accuracy is used. Without timestamps it is possible to apply a general trace-merging method, but the temporal dynamics of ROMA will be lost.

The experiments described in chapter 6 used an empty map as starting point for ROMA. We have already shown that ROMA operates with a map built using ROMA in experiment 2 as described in section 6.3. Using the static time-to-live (TTL) we believe that ROMA can also be used with non-empty maps as starting point. By converting an existing map to ROMA the static TTL provides an initial estimate of traffic for the map. The value for static TTL then has to be set based on the most specific traffic intensity data available as concluded in section 7.3.

The simulator described in chapter 5 provided direct communication with the cars sending GPS measurements to ROMA. In a real-life situation direct communication can not be assumed and buffers are often used to store messages until they have been properly transferred. The ActMAP - FeedMAP framework described by Thomas et al. [25] provides a client-server architecture required for such buffering. We do not foresee difficulties with such a method since ActMAP - FeedMAP processes sets of map changes similar to the incremental trace-merging approach of ROMA.

The research in this thesis focused on a centralized map to be kept up to date. We believe that the basic principle of trace-merging can also be applied in a distributed approach without central control. We do think that the current implementation of ROMA is far from distributed and shall therefore describe this option as future work in section 8.2.3 of the next chapter.

8

Conclusion and future work

This chapter concludes the research in this thesis on the development and evaluation of a robust online map-generation algorithm (ROMA). Section 8.1 describes the answers to the research questions set in section 1.3 based on the results and discussion thereof. Section 8.2 concludes the chapter with a look into future work based upon reflection on the research performed in this work.

8.1. Conclusion

The research in this thesis was used to find answers to the research questions in section 1.3. This section first lists the subquestions and concludes with a conclusion on the main research question.

Subquestion 1

How to measure spatial and temporal accuracy of a map?

The first subquestion is addressed in chapter 3 where we described a novel graph-based evaluation of a vector-map against a known ground truth. This method, which has a non-optimized computational complexity of $O(|V|^3)$, defines the common edge subgraph for two graphs which respects geographical node locations. The common edge subgraph enabled us to define true positive matches between map and ground truth and derive precision and recall for the developed map. We defined the balanced F-score of these two measures as a single value metric for spatial accuracy of a map. We defined the time to recover from changes in the world as the temporal accuracy of a map. The developed metric is an advantage to the method of Biagioni and Eriksson [3] for its ability to deal with displaced maps which occur in dynamic map generation.

Subquestion 2

How to create a dynamic map?

The second subquestion is addressed in chapter 4 where we described the Robust Online Map-generation Algorithm (ROMA). This method, which has a non-optimized computational complexity of $O(|V|^2)$, creates a dynamic map by processing streams of GPS measurements. ROMA falls in the class of trace-merging algorithms [3] and applies artificial pheromones for topological dynamics.

Subquestion 3

How spatially and temporally accurate can a dynamic map be?

The third subquestion is addressed in chapter 6 where we describe the application of ROMA in the simulated environment described in chapter 5. From the results of the road introduction experiment in section 6.2 we conclude that a navigable map can be created with a recall value of 0.8 and above. Roma suffers from outliers in measurements, as do all trace merging algorithms [3]. The precision values for ROMA in the road introduction experiment decline over time as the formation of web-like structures on the map get the upper hand. Maximum values for precision

of traffic densities up to 1000 cars do show maximum precision values of 0.5 and above as shown in table 8.1. In section 7.3 we discussed how the effect of outliers can be minimized and in section 8.2.2 we propose future research to contribute to this goal. The obtained insight in the process of development of web-like structures is an addition to the known literature on trace merging methods.

traffic	max. precision	max. recall
50	0.558	0.928
100	0.524	0.676
200	0.578	0.834
500	0.558	0.928
1000	0.515	0.864
2000	0.476	0.819
4000	0.387	0.781

Table 8.1: Maximum precision and recall in experiment 1 for different amounts of traffic.

The moment when precision declined was defined as proxy for temporal accuracy and showed how fast ROMA responds to changes in the world. These moments of overfitting were defined for traffic intensities of 1000, 2000, and 4000 cars in experiment 1 as shown in table 8.2. ROMA responds to road introduction in 3.5 to 6.5 minutes with speedier recovery for more cars. The speed of detection for road removal differs greatly per amount of roadblocks as shown in table 8.3. With the percentage of roadblocks equal, the moment of recovery seems earlier for higher traffic densities. ROMA generally responds to road removal within 30 minutes for the simulated traffic densities. The obtained insight in the temporal aspect of incrementally generated maps is an addition to research in map generation algorithms.

traffic	τ (min)	prec.	rec.	F
1000	6.5	0.515	0.791	0.624
2000	5.5	0.463	0.750	0.573
4000	3.5	0.350	0.730	0.473

Table 8.2: Estimated moment of overfitting for different amounts of traffic.

	τ (min)		
block (%)	1000 cars	2000 cars	4000 cars
10	28.5	15.5	4.5
20	17.0	8	6
50	28.5	24	out of range

Table 8.3: The estimated time of recovery for different amounts of roadblocks.

Subquestion 4

How do spatial and temporal accuracy correlate ?

The fourth subquestion is also addressed in chapter 6. From the results of experiment 1 we expect that better maps take more time to be build using ROMA. This implies a negative correlation between spatial and temporal accuracy since temporal accuracy is the opposite of time needed to recover. As single experimental runs were used for each of the configurations we can not indicate specific values of correlation.

Subquestion 5

Which domains can benefit from the developed method for dynamic mapping?

The fifth subquestion is addressed in section 7.4 where the generalisability of ROMA and the developed metric is explored. We believe that environments where no a priori map is available or

where the topology and geometry are changing can benefit from the dynamic aspect of ROMA. Examples of such environment are found in recovery scenarios for large scale (natural) disasters and the constantly available setting of open pit mining.

Besides ROMA this research has also provided methods and insights useful for the Geographic Information System (GIS) community. To our knowledge this is the first research project looking in to the effects over time of incremental map generation. We have therefore developed new insight in the outliers generally observed in trace merging algorithms. The developed map comparison method also offers a novel and graph-based approach suitable for the evaluation of dynamic maps.

Main research question

How to create and maintain an accurate and dynamic map based on position traces?

In conclusion we believe that the Real-time Online Map-generation Algorithm (ROMA) described and evaluated in this thesis provides a proof of concept for online generation of dynamic vector-maps. The focus on quantitative evaluation of map quality furthermore makes a valuable contribution to the GIS community.

8.2. Future work

In this section we provide three recommendations for future work following the discussion in chapter 7. Section 8.2.1 defines a research focus for future work on the developed metric. Section 8.2.2 explores the improvements to ROMA regarding its dynamic nature and dealing with outliers in trace merging algorithms. Section 8.2.3 describes the possibilities for ROMA in a distributed setting and the resulting positive effects on end-user privacy.

8.2.1. Improvements to the metric

The metric described in chapter 3 provides a similarity measure for two vector-maps on the interval [0,1]. In this section we propose improvements to better approximate the true positives.

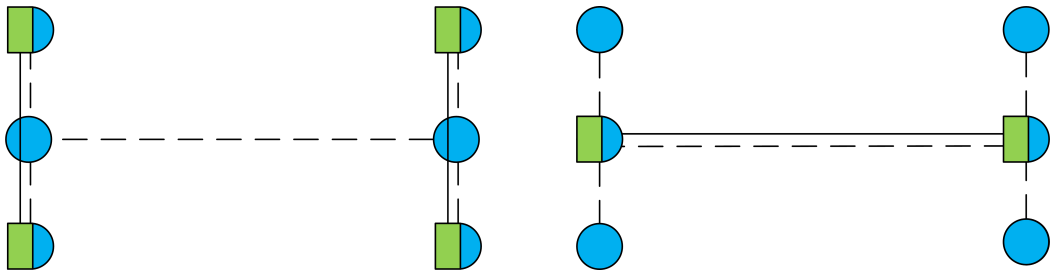
We propose to investigate a combination of Maximum Common Edge Subgraph (MCES) algorithms with the use of edge-similarity measure usable for geographical maps. Similarity between roads is now calculated using a simple comparison of edge lengths which does not penalize the true positives for roads of equal length at unequal locations. To emphasize geographical similarity we would replace this method with one which directly compares the shape and position of two edges. In terms of topological similarity we suggest to look into approximation algorithm for the Maximum Common Edge Subgraph (MCES). Polynomial-time approximations of algorithms for the MCES problem exist [22] and can be used to provide an improved approximation of topology. Since MCES algorithms normally work with graphs of fixed edge weight, including an edge similarity measure makes the problem less trivial.

We furthermore propose to further research the definition of map quality with regard to different application domains. In our research we concluded that a vector-map representation is usable for disaster relief settings as long as it has a decent recall value. For situations where a map is used for automatic driving it might be more valuable to have a high precision value. Connectivity may also play a role in determination of map quality. Our method does not differentiate between maps with many disjoint clusters and maps with coverage and connection on a large area as shown in fig. 8.1. Preference of connectivity and the balance between precision an recall might differ per application domain. We therefore propose to investigate what qualities in terms of topology and geometry are most valued for maps in different domains.

8.2.2. Topological and geographical map dynamics

This section explores the improvements to ROMA regarding its dynamic nature and dealing with outliers in trace merging algorithms. In section 7.3 we discussed how measurement outliers probably result in the web-like structures frequently observed in trace merging methods. In this section we combine these discussed points to propose a different map representation method for dynamic maps. We furthermore propose further research in comparison of edges so they could be combined when representing the same road.

The shape of a road in ROMA is initially determined by the positions of the first measurements creating that road. Projections of measurements are added to adjust these positions but influence



(a) A disjoint common edge subgraph (b) Non-disjoint common edge subgraph

Figure 8.1: Two common edge subgraphs result in identical precision and recall values. Which is the better graph?

the shape mainly perpendicular to the road location. In order to better represent the actual road shape, we propose to store the measurements directly in the edge itself. Methods used in regression analysis can then be applied to represent the shape on the map and estimate a smooth centreline fitting the level of detail required. With this alternative approach we believe that the road shape would be better represented.

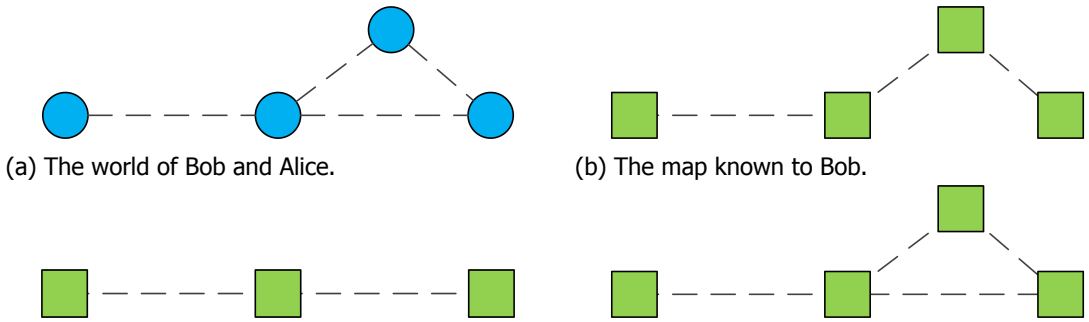
Although the suggested approach better represents the road shape, measurement outliers might still cause the creation of unnecessary duplicate roads. With all original measurements stored in the road itself it becomes possible to compare two sets of measurements with statistical methods. ROMA uses Welch's t-test to compare if two nodes actually represent the same location and such a comparison can also be applied to roads consisting of sets of measurements. If the t-test concludes that measurements in two roads actually stem from the same road in the world, both roads can be merged into one. Where the merger of two adjacent nodes representing the same road does not cause topological changes in the map, the merger of two roads might and is therefore non-trivial. Given this added complexity for merging two roads we propose further research into the conditions for and effects of merging two roads.

8.2.3. Distributed application of ROMA

The research in this thesis focused on centralized processing of traces to generate a map. In this section we propose a method for a decentralized application of ROMA and trace-merging methods in general.

In a decentralized approach to map generation GPS trace data is exchanged directly between trace providers who each generate their own map representation of the world. The car 2 car communication consortium [6] proposes the use of wireless signals in their concept where cars communicate dangers and traffic information directly to one another. We believe that a collaborative roadmap can be created by collectively broadcasting parts of the known road map to other cars. The following example shows how we envision this distributed application for the fictional world shown in fig. 8.2a. Bob intends to travel from left to right on the world and travels towards his destination while Alice travels in the opposite direction. Bob's knowledge of the world is shown in fig. 8.2b and Alice knows what is shown in fig. 8.2c. Bob and Alice encounters one another just after Bob has started his travel at the common part of both their maps. Bob broadcasts his intended route and this message is received by Alice. Alice adds this knowledge to her map representation using ROMA and sees that her map offers a shorter route. She then replies to Bob's message by sending an improved route (subset of her map) to Bob. Bob adds this knowledge to his map using ROMA and sees that he can now travel on a shorter route. Alice also broadcasts her intended path, but receives no alternative since Bob knows no shorter route. In the end Bob and Alice both have the map representation shown in fig. 8.2d.

Since wireless signals have a limited range and cars might not drive within communication of one another range for a long time, research is needed as to what can and should be communicated. Balch and Arkin [2] has shown that little communication is needed for artificial ant colonies to efficiently perform a foraging task together. We suggest to research the decentralization of ROMA in an similar manner to explore what is essential for collaborative map generation. Based on the believe that locals best know the local situation, perhaps only the first part of the intended route can be sent in detail while restricting the rest to topological information only.



(a) The world of Bob and Alice.

(b) The map known to Bob.

(c) The map known to Alice.

(d) The resulting map for Bob and Alice.

Figure 8.2: Evolution of the maps for Bob and Alice after exchanging map information.

We believe that by broadcasting a route known to the sender, user privacy can be preserved. No information about the intended route of the sender himself has to be transferred which makes it difficult to obtain information such as a home address. Broadcasting equal message types for both requests and answers furthermore prevents obvious sending of the intended route. The sent messages can be constructed using measurements from different origins for each edge in the route. If the recorded measurements are labelled with a rotating random number for each edge, the source of measurement can also not be traced back to his or her places of interest.

A

Graphs for experiment 1

A.1. Per quality measure

Precision

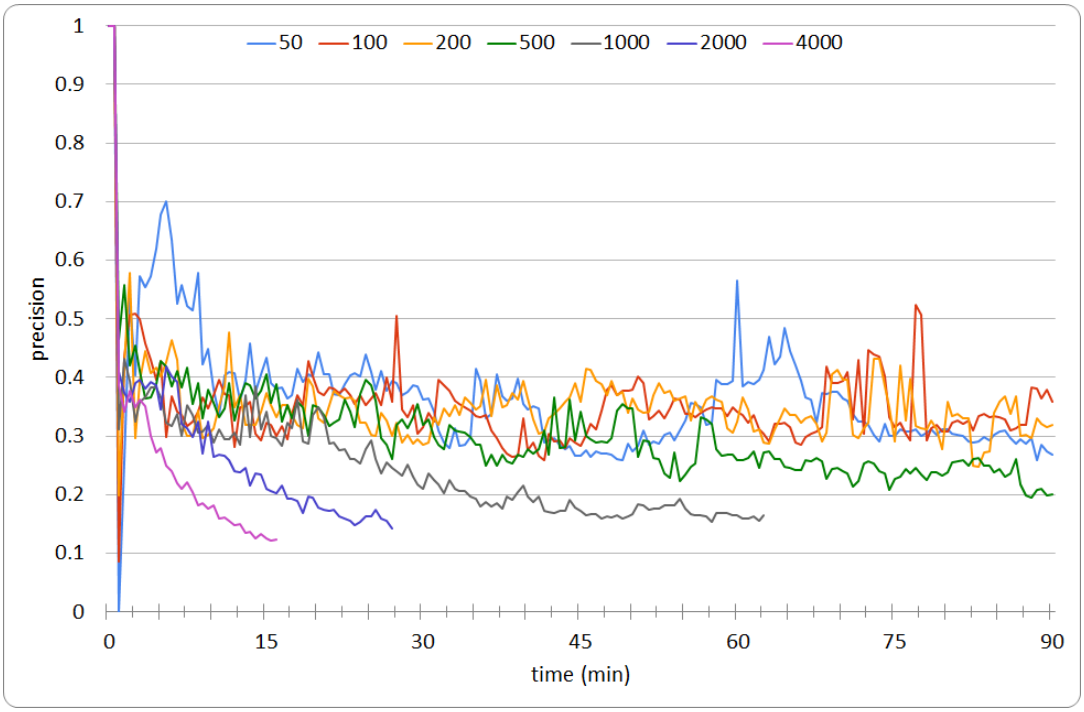


Figure A.1: Course of precision over the first 90 minutes simulated.

Recall

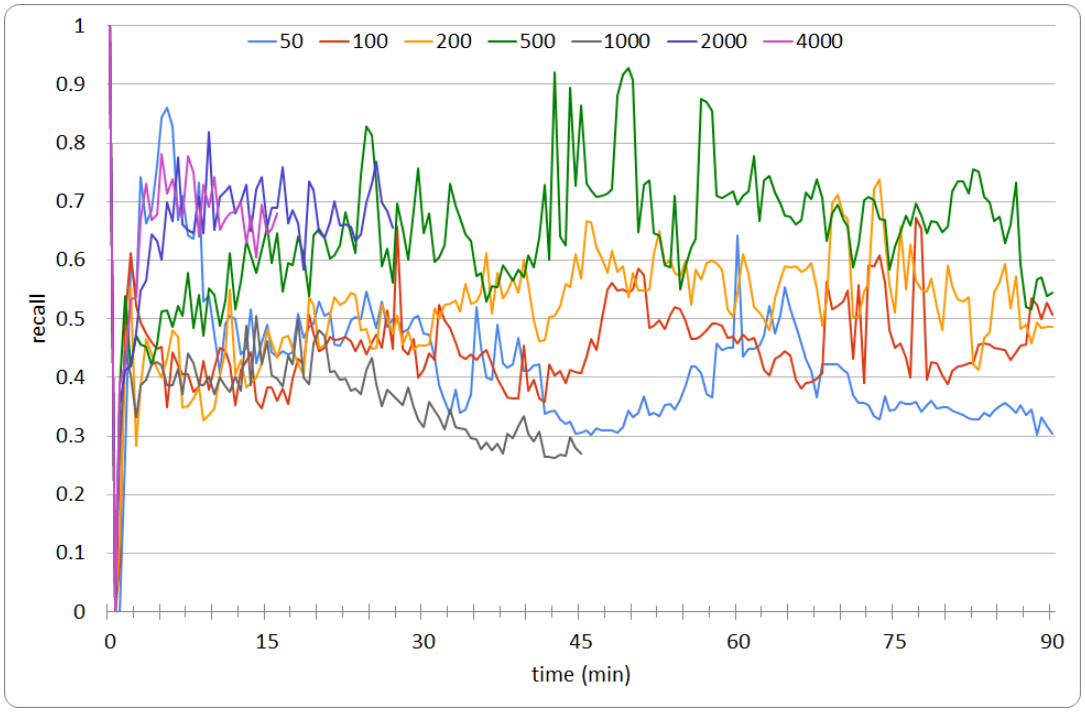


Figure A.2: Course of recall over the first 90 minutes simulated.

F-score

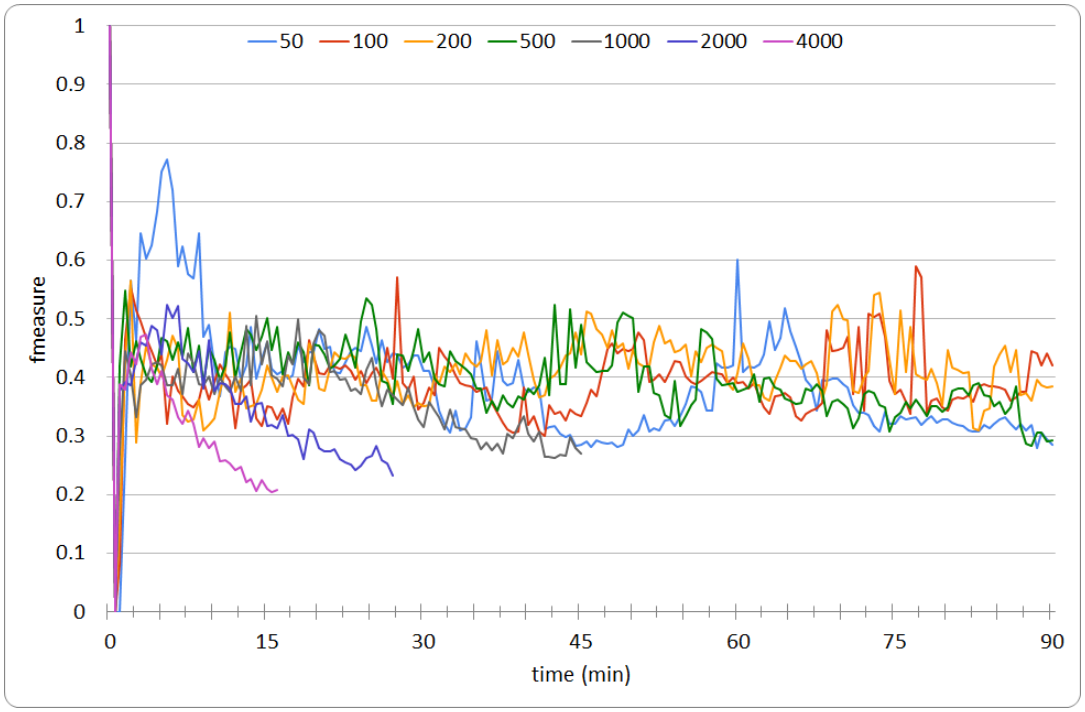


Figure A.3: Course of the balanced f-score over the first 90 minutes simulated.

A.2. Per traffic intensity

50 cars

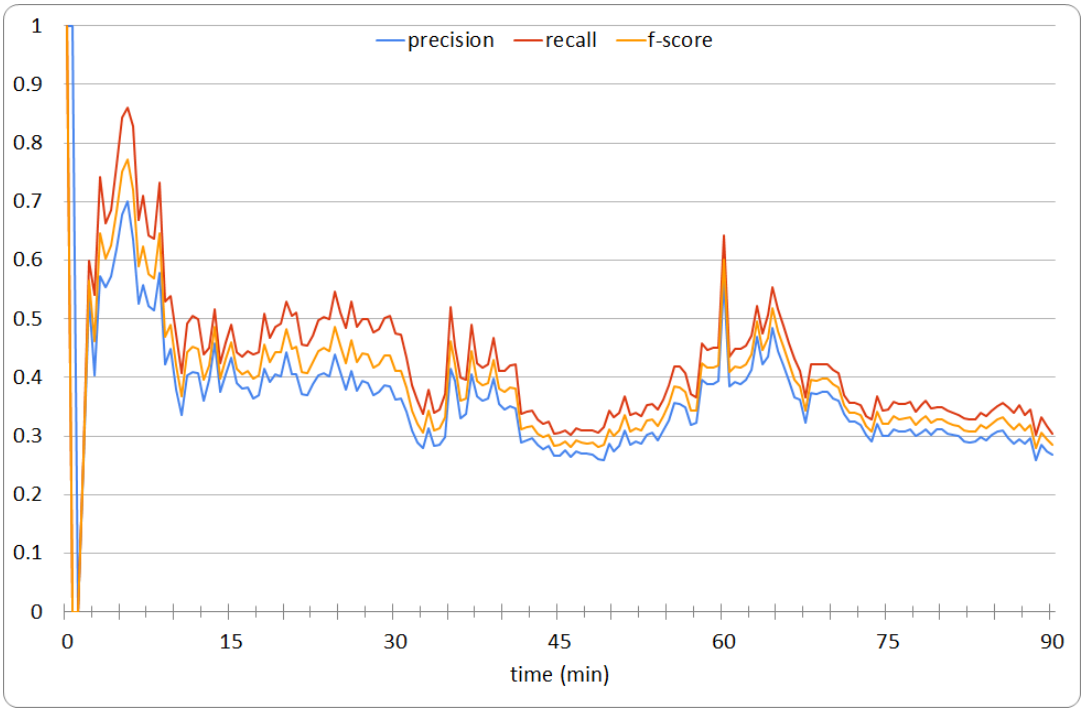


Figure A.4: Course of map quality for 50 cars over the first 90 minutes simulated.

100 cars

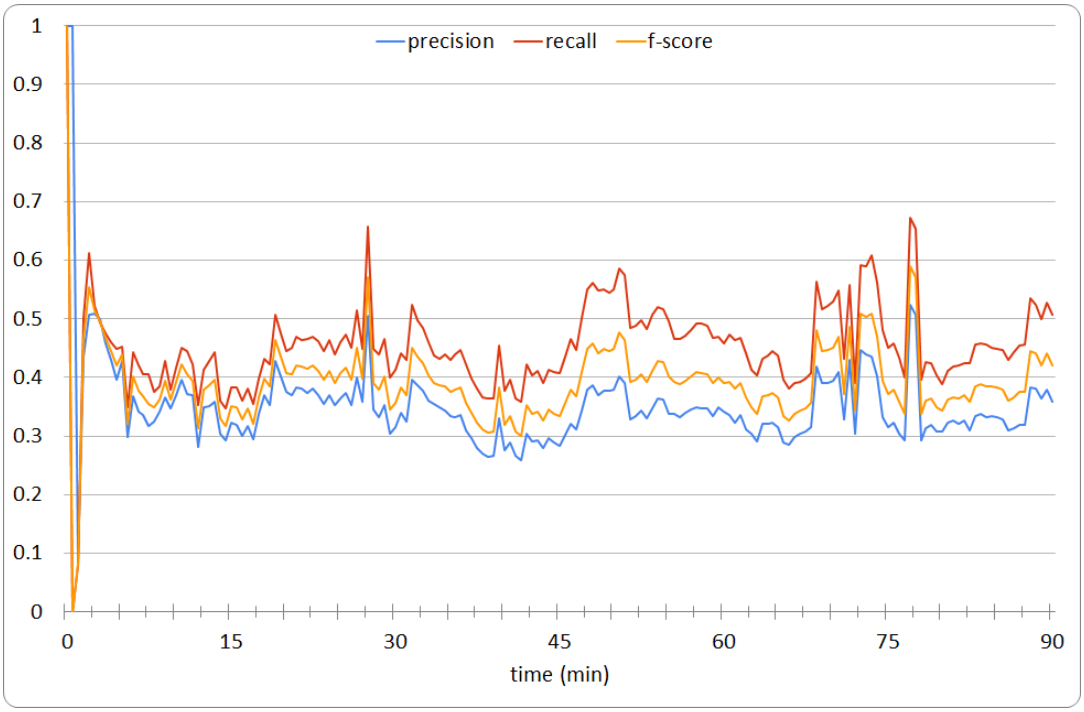


Figure A.5: Course of map quality for 100 cars over the first 90 minutes simulated.

200 cars

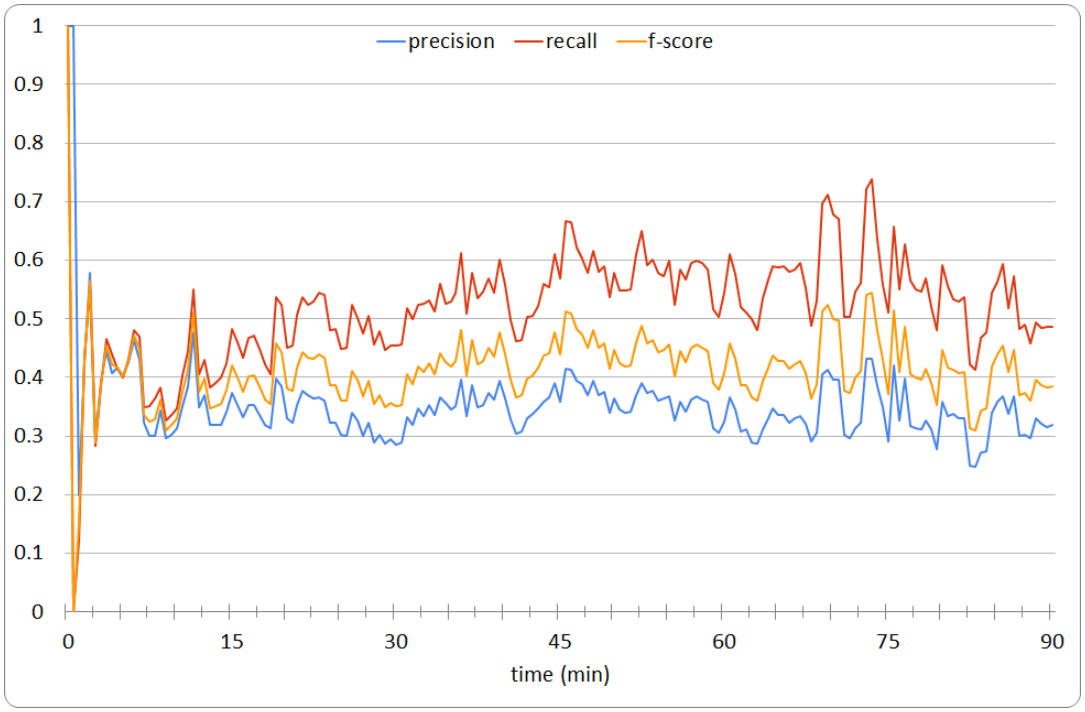


Figure A.6: Course of map quality for 200 cars over the first 90 minutes simulated.

500 cars

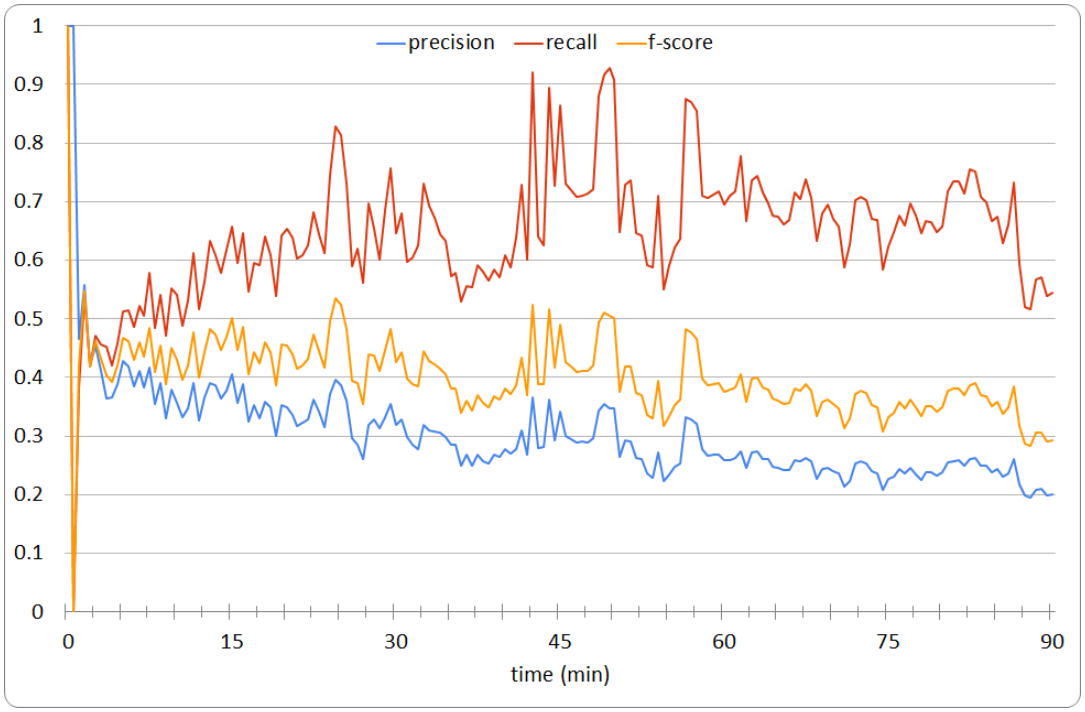


Figure A.7: Course of map quality for 500 cars over the first 90 minutes simulated.

1000 cars

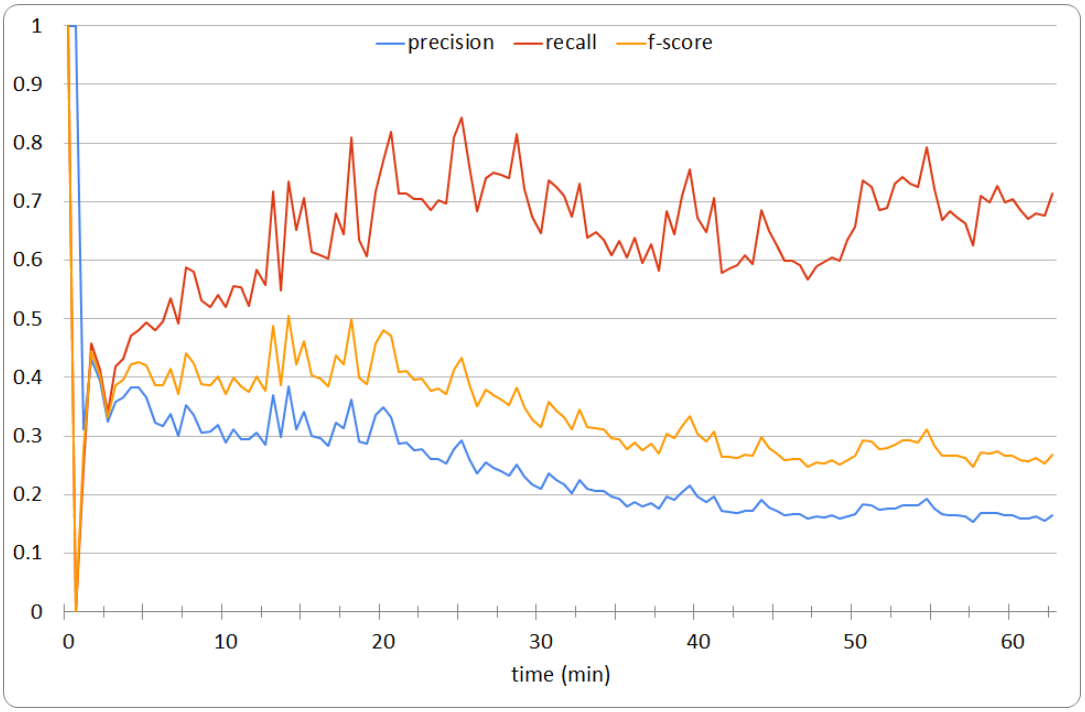


Figure A.8: Course of map quality for 1000 cars over 62.5 minutes simulated. Overfitting was observed at 6.5 minutes.

2000 cars

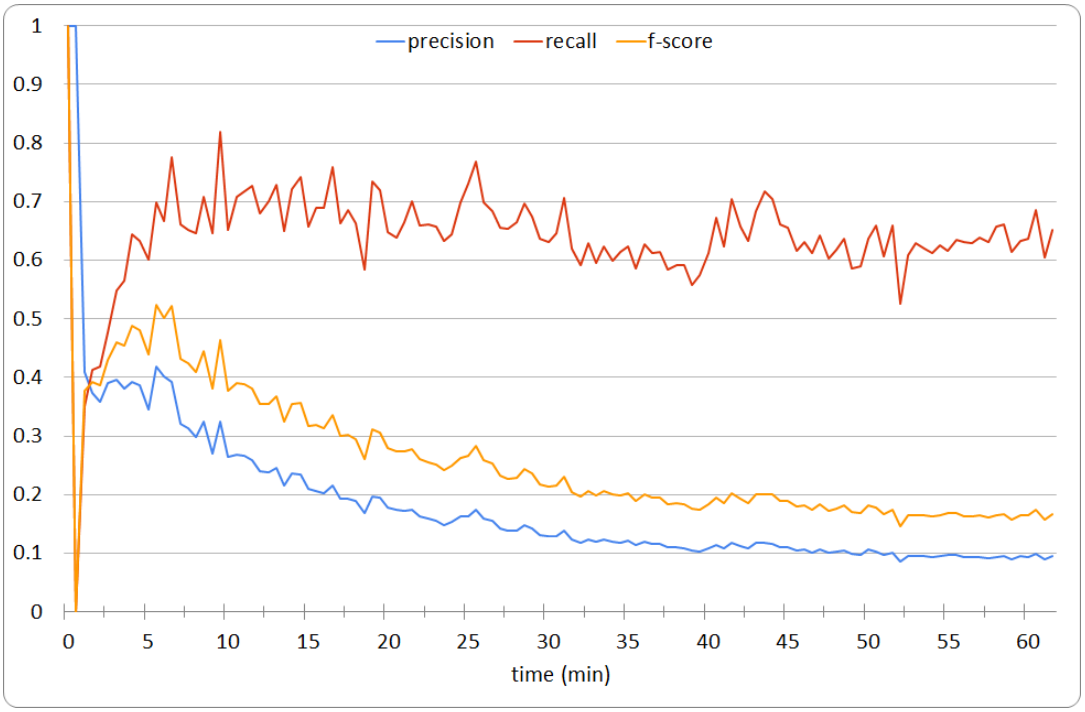


Figure A.9: Course of map quality for 2000 cars over 61.5 minutes simulated. Overfitting was observed at 5.5 minutes.

4000 cars

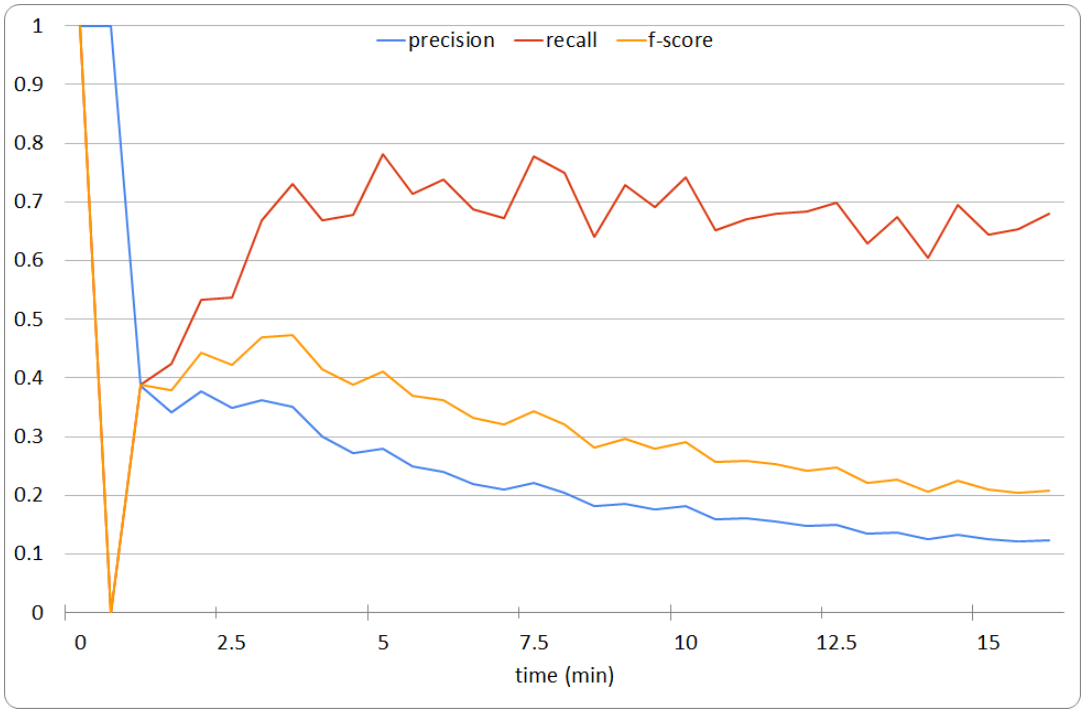


Figure A.10: Course of map quality for 4000 cars over 16 minutes simulated. Overfitting was observed at 3.5 minutes.

B

Graphs for experiment 2

B.1. Per traffic intensity

1000 cars

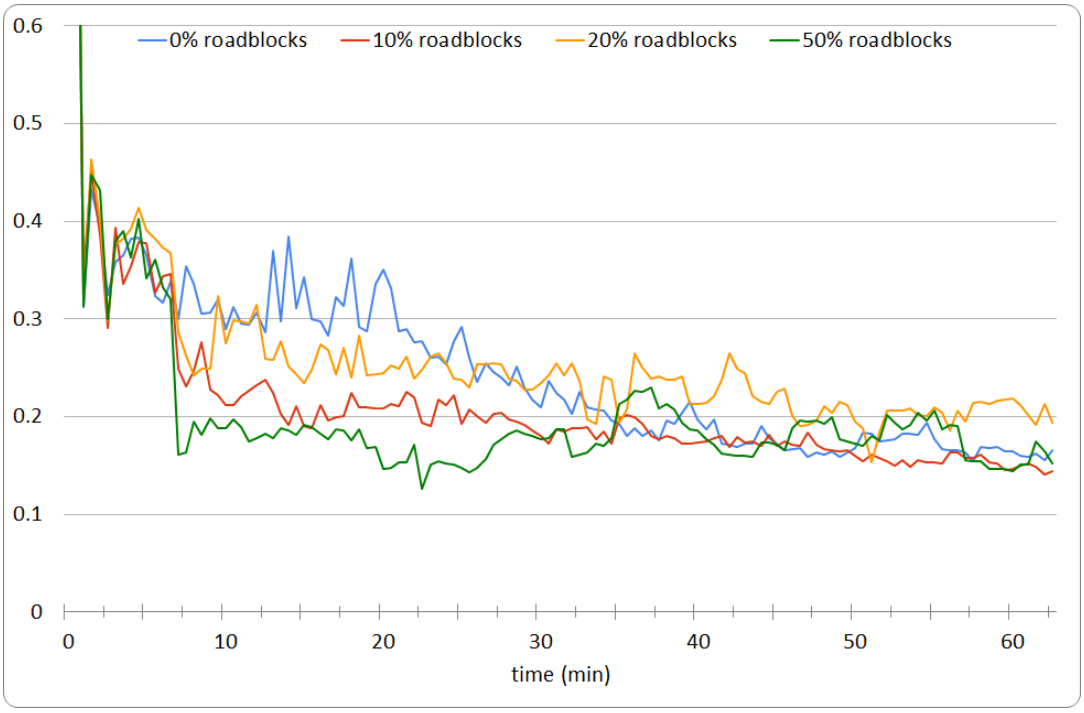


Figure B.1: Course of precision over the first 62.5 minutes simulated. Roadblocks were added at 6.5 minutes and recovery was observed 28.5 minutes after for 10%, 17 minutes after for 20%, and 28.5 minutes after for 50% roadblocks.

2000 cars

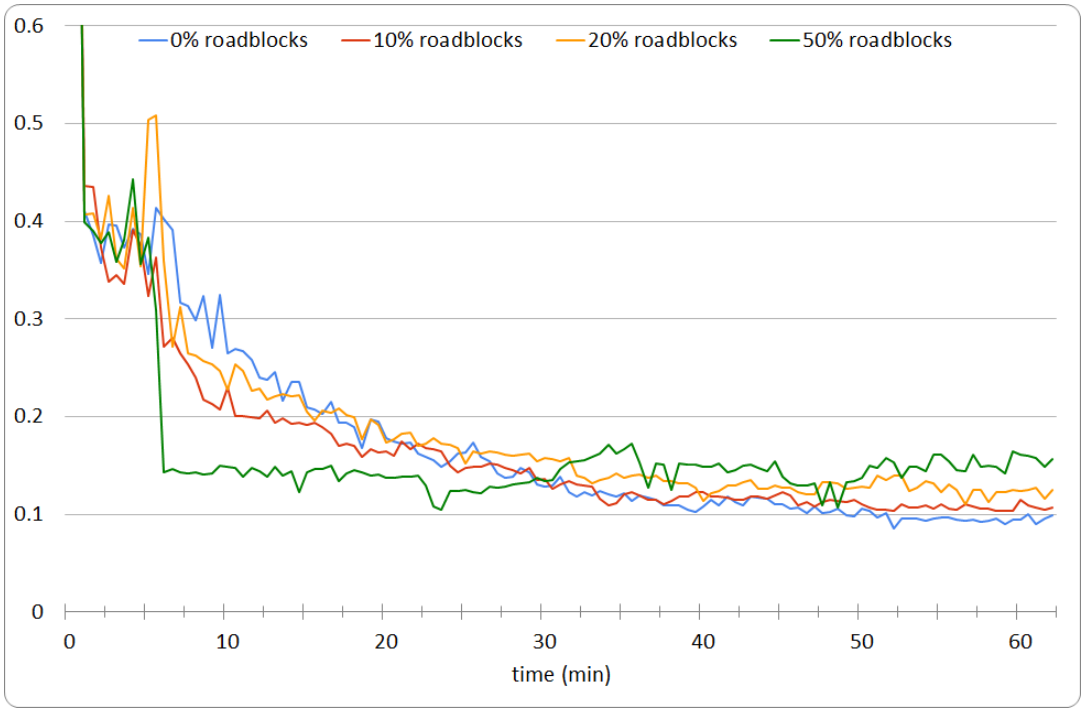


Figure B.2: Course of precision over the first 61.5 minutes simulated. Roadblocks were added at 5.5 minutes and recovery was observed 15.5 minutes after for 10%, 8 minutes after for 20%, and 24 minutes after for 50% roadblocks.

4000 cars

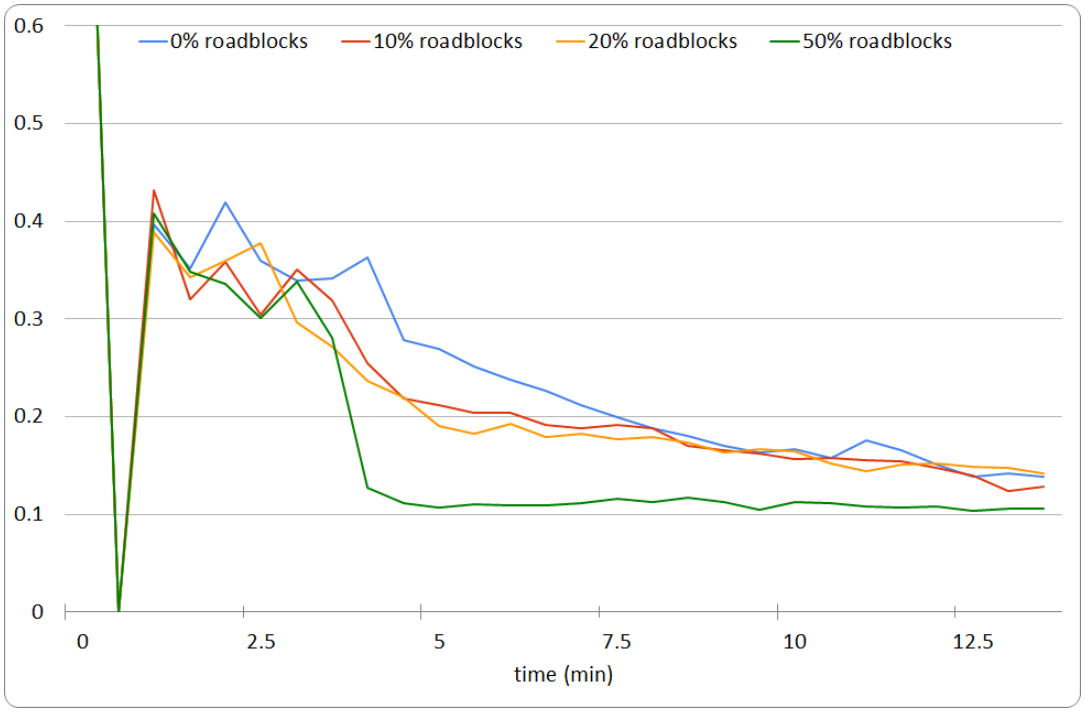


Figure B.3: Course of precision over the first 13.5 minutes simulated. Roadblocks were added at 3.5 minutes and recovery was observed 4.5 minutes after for 10%, 6 minutes after for 20%, and not before the end of simulation for 50% roadblocks.

C

Source code

In this appendix we provide a selection of functions from the cartography and analysis modules in fig. C.1. All code is written in Java. The complete software package is available upon request.

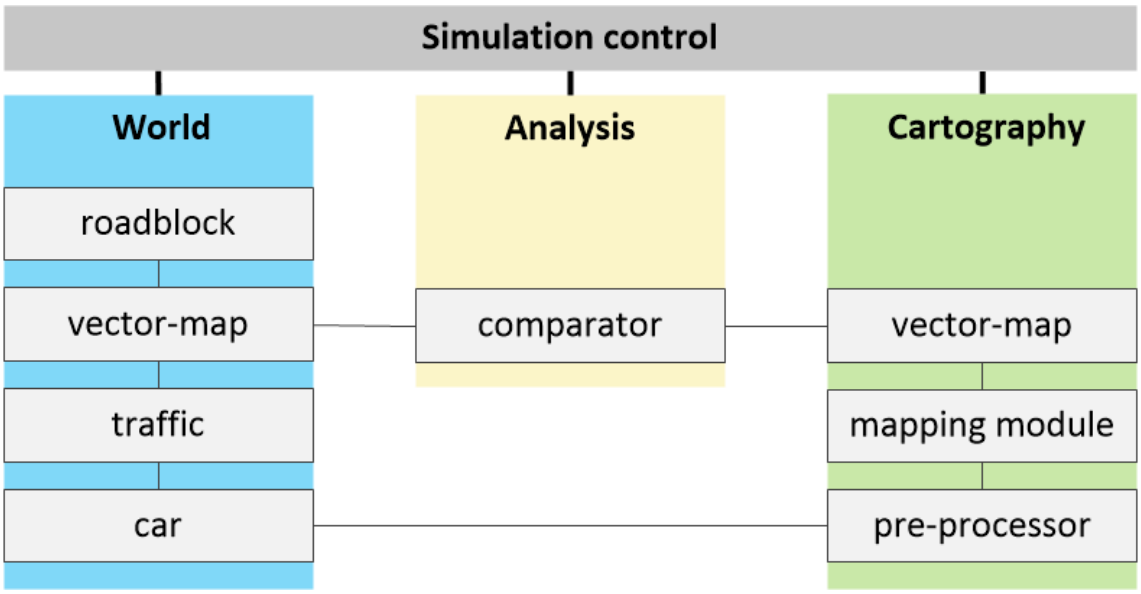


Figure C.1: Overview of the Road Block Simulation tool (RBS) with modules for cartography (chapter 4) and analysis (chapter 3) added.

Appendix C.1 describes the steps used for simulation and preprocessing of measurements. Appendix C.2 describes the core functionalities of ROMA in terms of path matching, merging, and adjustment. Appendix C.3 describes the map dynamics used for calculating the time to live for nodes and edges. We conclude this appendix with a description of the method for graph comparison in Appendix C.4.

C.1. Pre-processing

Both functions described in this section can be found in *rbs.traffic.traceagents.CarTraceAgent*.

update

public void update(TimePoint now)

```
// only add a measurement if the car has not stopped yet (End of stream
    )
if(!carstopped){
```

```
  MeasurementProvided current = getNoisyMeasurement(now);
  if(current.accuracy < minaccuracy){
    double d = measurements.isEmpty()?0:Math.abs(current.getBearing() -
        measurements.getLast().direction(current)) % 360;
    measurements.add(current);
  }
}

//get a valid sequence of measurements up to look-ahead size if
    available
if(carstopped || measurements.size() >= lookahead + 2){
  LinkedList<MeasurementProvided> validlist = new LinkedList<
      MeasurementProvided>();
    while(measurements.getFirst() == null)
      measurements.removeFirst();
  validlist.addLast(measurements.getFirst());
    while(validlist.size() < Math.min(measurements.size(), lookahead +
        2) && measurements.get(validlist.size()) != null && validlist.
        getLast().distance(measurements.get(validlist.size())) <=
        maxdistance)
      validlist.addLast(measurements.get(validlist.size()));

    //continue path estimation, generation, and adjustment in the map-
        module
    MeasurementProvided[] validarray =
        validlist.toArray(new MeasurementProvided[validlist.size()]);
  if(CartographyGlobals.getMap() instanceof Graph){
    Graph map = (Graph)CartographyGlobals.getMap();
    lastlocation = map.updateGraph(lastlocation, validarray);
  }
  else
    ((Measured)(CartographyGlobals.getMap())).addMeasurement(validarray
        );

  //remove the measurement from where this iteration started
    measurements.removeFirst();
}
```

getNoisyMeasurement

private MeasurementProvided getNoisyMeasurement(TimePoint now)

```
Coord realpos = getCar().getPosition();

//determine the direction to which to shift this measurement
int latsign = (Globals.getRandom().nextInt(2)) * 2 - 1;
int lonsign = (Globals.getRandom().nextInt(2)) * 2 - 1;
double angle = Globals.getRandom().nextDouble() * (Math.PI / 2d);

//determine the distance to shift the measurement
double range = Globals.getRandom().nextGaussian() * accuracymean;
double distlon = Math.cos(angle)* range;
double distlat = Math.sin(angle)* range;

//determine how much distance is covered when shifting one unit of
    precision in lat/lon
double precision = 1d/(1000d);
```

```
double distdlat = realpos.distance( new Coord((realpos.getLatitude() +
    precision), realpos.getLongitude()) );
double distdlon = realpos.distance( new Coord((realpos.getLatitude()),
    realpos.getLongitude() + precision) );

//calculate new latitude and longitude
double newlat = realpos.getLatitude() + latsign * ((distlat / distdlat)
     * precision);
double newlon = realpos.getLongitude() + lonsign * ((distlon / distdlon
    ) * precision);

return new MeasurementProvided( now.getTimeInMillis(), this.carID,
    newlat, newlon, 0d, getCar().getBearing(), (float) getCar().
    getVelocity(), accuracymean, ""+ getCar().hashCode() );
```

C.2. Map generation

The functions described in this section can be found in *rbs.world.designed.Graph*.

updateGraph

public FoundPath updateGraph(FoundPath previouspath, MeasurementProvided[] measurements)

```
if(measurements == null || measurements.length <2)
  return null;
if(previouspath == null)
  previouspath = new FoundPath(null, null);

//find the fit and reachable edges
LinkedList<LinkedList<MatchedEdge>> fitedges = getFitEdges(previouspath
    .lastedge, measurements);
LinkedList<HashMap<MatchedEdge, SearchNode2>> reachededges =
    getReachedEdges(measurements, fitedges, previouspath);

//extract the followed path
MatchedEdge bestgoal = getBestGoal(fitedges.get(1), reachededges);
FoundPath foundpath = getInfluencedNodes(bestgoal, measurements,
    fitedges, reachededges);

//if no distance was traveled, because start and goal where direct
    neighbours, influence the connecting node
if(bestgoal != null && bestgoal.edge != previouspath.lastedge && !
    bestgoal.edge.contains(previouspath.getLastInfluenced()))
  foundpath.path.add(bestgoal.edge.getA().distance(measurements[0]) <
      bestgoal.edge.getB().distance(measurements[0])? bestgoal.edge.
      getA(): bestgoal.edge.getB());

//update each of the influenced (traveled) nodes
if(!foundpath.path.isEmpty()){
  //influence the traveled nodes
  for(Node cur: foundpath.path)
    influenceNode(cur, measurements[0], measurements[1]);
  //influence the traveled edges
  for(Edge e:foundpath.getTraveledEdges())
    influenceEdge(e,measurements[0], measurements[1]);
  //determine if nodes need to be merged based upon location equality
  for(Node cur: foundpath.path){
```

```
    boolean hasmerged = true;
    while(hasmerged){
      hasmerged = false;
      for(Node neighbour: cur.getNeighbours()){
        if(cur.hasEqualLocation(neighbour,minimallocationequality)){
          cur = mergeNodes(cur, neighbour);
          hasmerged = true;
          break;
        }
      }
    }
  }
}
return foundpath;
```

getFitEdges

private LinkedList<LinkedList<MatchedEdge» getFitEdges(Edge start, MeasurementProvided[] measurements)

```
LinkedList<LinkedList<MatchedEdge>> result = new LinkedList<LinkedList<
    MatchedEdge>>();
for(int i = 0; i<measurements.length; i++)
  result.addLast(new LinkedList<MatchedEdge>());
for(Edge e: E){
  double[] fitness = new double[measurements.length];
  Boolean[] perpendicular = new Boolean[measurements.length];
  //check if the edge is perpendicular to any of the provided
      measurements
  for(int i = 0; i<measurements.length; i++){
    double currentfitness = fitnesscalc.calculateFitPerpendicular(e.
        getA(), e.getB(), measurements[i]);
    double currentfitness2 = fitnesscalc2.calculateFitPerpendicular(e.
        getA(), e.getB(), measurements[i]);
    fitness[i] = currentfitness;
    double positiononline = Util.getPositionOnLine(e.getA(), e.getB(),
        measurements[i]);
    perpendicular[i] = new Boolean((positiononline >= 0d &&
        positiononline <= 1d) || (start != null && i == 0 && start == e
        ));
  }
  //if the edge is perpendicular to a measurement, match it to such a
      measurement
  if(Arrays.asList(perpendicular).contains(true))
    for(int i = 0; i < measurements.length; i++){
      if(perpendicular[i] && fitness[i] > this.minimaledgefitness)
        result.get(i).add(new MatchedEdge(e, fitness[i]));
    }
  //else, match it to the best (non-perpendicular) option
  else{
    int bestposition = -1;
    for(int i = 0; i < measurements.length; i++){
      if(fitness[i] > this.minimaledgefitness && (bestposition < 0 ||
          fitness[i] > fitness[bestposition]))
        bestposition = i;
    }
    if(bestposition >= 0)
```

```
        result.get(bestposition).add(new MatchedEdge(e, fitness[
            bestposition]));
    }
  }
  //if a previous step matched to an edge, only that edge and its
      neighbours should be in E'^0
  if(start != null){
    LinkedList<MatchedEdge> optionalfit = (LinkedList<MatchedEdge>)
        result.get(0).clone();
    result.get(0).clear();
    result.get(0).add(new MatchedEdge(start, fitnesscalc.
        calculateFitPerpendicular(start.getA(), start.getB(),
        measurements[0])));
    boolean added = true;
    while(added){
      added = false;
      for(MatchedEdge mo: optionalfit){
        for(MatchedEdge mi: result.get(0)){
          if(mo.edge.neighbours(mi.edge)){
            result.get(0).add(mo);
            optionalfit.remove(mo);
            added = true;
            break;
          }
        }
        if(added)
          break;
      }
    }
  }
  for(int i = 0; i<measurements.length; i++)
    Collections.sort(result.get(i));
  return result;
```

getReachedEdges

private LinkedList<HashMap<MatchedEdge, SearchNode2» getReachedEdges(MeasurementProvided[] measurements, LinkedList<LinkedList<MatchedEdge» fitedges, FoundPath previouspath)

```
LinkedList<HashMap<MatchedEdge, SearchNode2>> reachededges = new
    LinkedList<HashMap<MatchedEdge,SearchNode2>>();
LinkedList<MatchedEdge> startedges = fitedges.get(0);
for (int i = 0; i < measurements.length-1; i++){
  LinkedList<MatchedEdge> unreachededges = new LinkedList<MatchedEdge>(
      fitedges.get(i+1));
  reachededges.addLast(new HashMap<Graph.MatchedEdge, Graph.SearchNode2
      >());
  //find all reachable edges from the startedges, starting at the
      fittest edge thusfar
  for(int j = startedges.size()-1; j>= 0; j--){
    LinkedList<SearchNode2> currenttree;
    if(i == 0 && startedges.get(j).edge.contains(previouspath.
        getLastInfluenced()))
      currenttree = getSearchTree2(measurements[i], measurements[i+1],
          new Node[]{startedges.get(j).edge.getOther(previouspath.
          getLastInfluenced())});
    else
```

```
        currenttree = getSearchTree2(measurements[i], measurements[i+1],
            new Edge[]{startedges.get(j).edge});
      HashMap<MatchedEdge, SearchNode2> currentreachededges =
          getReachableEdges2(unreachededges, measurements[i],
          measurements[i+1], currenttree, startedges.get(j).fitness);
      //work around method, because unreachededges.removeAll does not use
          equals method
      for(MatchedEdge e1: currentreachededges.keySet()){
        for(MatchedEdge e2: unreachededges){
          if(e2.equals(e1)){
            unreachededges.remove(e2);
            break;
          }
        }
      }
      //unreachededges.removeAll(currentreachededges.keySet());
      reachededges.get(i).putAll(currentreachededges);
      if(unreachededges.isEmpty())
        break;
    }
    //generate the set of startedges for the next iteration. Special case
        if reachededges is empty and we are at first iteration
    if(i==0 && reachededges.get(i).isEmpty())
      startedges = fitedges.get(i+1);
    else
      startedges = new LinkedList<MatchedEdge>(reachededges.get(i).keySet
          ());
    Collections.sort(startedges);
  }
  return reachededges;
```

getBestGoal

private Graph.MatchedEdge getBestGoal(LinkedList<MatchedEdge> goals, LinkedList<HashMap<MatchedEdge, SearchNode2» reachededges)

```
double bestsummedfitness = 0d;
MatchedEdge bestfounddestination = null;
for(int i = reachededges.size()-1; i>0; i--){
  // set the destination for which the source should be found
  if(bestfounddestination == null){
    if(!reachededges.get(i).isEmpty()){
      LinkedList<MatchedEdge> current = new LinkedList<MatchedEdge>(
          reachededges.get(i).keySet());
      Collections.sort(current);
      bestfounddestination = current.getLast();
      bestsummedfitness = bestfounddestination.fitness;
    }
  }

  // set the list of sources in which the source should be found
  LinkedList<MatchedEdge> sources = new LinkedList<MatchedEdge>(
      reachededges.get(i-1).keySet());
  if(i == 1 && sources.isEmpty() && !goals.isEmpty()){
    Collections.sort(goals);
    return goals.getLast();
  }
```

```
  Collections.sort(sources);

  //find if there is a better source then the best found destination
  boolean betteroption = false;
  for(int s = sources.size()-1; s >= 0; s--){
    if(bestfounddestination == null || sources.get(s).fitness >
        bestsummedfitness){
      bestfounddestination = sources.get(s);
      bestsummedfitness = sources.get(s).fitness;
      betteroption = true;
    }
  }
  if(betteroption)
    continue;

  //Find the edge from where bestfounddestination was reached
  if(reachededges.get(i).containsKey(bestfounddestination)){
    Coord root = reachededges.get(i).get(bestfounddestination).getRoot
        ().current;
    if(root instanceof Node){
      for(int s = sources.size()-1; s >= 0; s--){
        if(sources.get(s).edge.contains((Node)root)){
          bestfounddestination = sources.get(s);
          break;
        }
        else if(s == 0){
          bestsummedfitness = 0d;
          bestfounddestination = null;
        }
      }
    }
    else{
      bestsummedfitness = 0d;
      bestfounddestination = null;
    }
  }
}
return bestfounddestination;
```

getInfluencedNodes

private Graph.FoundPath getInfluencedNodes(Graph.MatchedEdge bestgoal, MeasurementProvided[] measurements, LinkedList<LinkedList<MatchedEdge» fitedges, LinkedList<HashMap<MatchedEdge, SearchNode2» reachededges)

```
Set<MatchedEdge> r = reachededges.getFirst().keySet();
if(bestgoal != null && r.contains(bestgoal))
  return new FoundPath(reachededges.getFirst().get(bestgoal).getPath(),
      bestgoal.edge);
else{
  //set the possible origins for path-search
  Coord[] origins = (fitedges.getFirst().size() == 0)? new Coord[]{
      measurements[0]}: new Coord[]{fitedges.getFirst().getLast().edge.
      getA(), fitedges.getFirst().getLast().edge.getB()};

  //set the possible destinations for path-search
```

```
  Coord[] destinations = (bestgoal == null)? new Coord[]{measurements
      [1]}: new Coord[]{bestgoal.edge.getA(), bestgoal.edge.getB()};

  return generatePath(origins, destinations, measurements[0],
      measurements[1]);
}
```

InfluenceNodes

private void influenceNode(Node v, MeasurementProvided m1, MeasurementProvided m2)

```
double percentonlinesegment = Util.getPositionOnLine(m1, m2, v);
MeasuredPosition m = (m2.combineWithMeasurement(m1,percentonlinesegment
    )).getPositionMeasurement();
v.addMeasurement(m);
```

mergeNodes

private Node mergeNodes(Node v1, Node v2)

```
//sever the connection between v1 and v2
Edge connection = v1.getEdge(v2);
if(connection != null)
    removeEdge(connection);

//remember all neighbours of v2
List<Node> n1 = v1.getNeighbours();
List<Node> n2 = v2.getNeighbours();

//move all neighbours from v2 to v1
for(Node neighbour: n2){
  removeEdge(v2.getEdge(neighbour));
  //only add this neighbour if it not already known to v1
  if(!n1.contains(neighbour))
    addEdge(new Edge(v1,neighbour));
  }

//add all measurements from v2 to v1
List<MeasuredPosition> m = v2.getMeasurements();
v1.addMeasurements(m);

//remove v2 from the graph
V.remove(v2);

//return the node which is left, for ease of pointing
return v1;
```

C.3. map dynamics

The functions described in this section can be found in *rbs.world.designed.Node* and *rbs.world.designed.Edge*.

Node.addMeasurement

public boolean addMeasurement(MeasuredPosition newmeasurement, boolean calculate)

```
boolean merged = false;
for(int i = 0; i<M.size(); i++){
  MeasuredPosition current = M.get(i);
```

```
    //if the instrument has already measured for this node, merge the two
        measurements
    if(M.get(i).getAgentID().equals(newmeasurement.getAgentID())){
      M.set(i, current.combineWithMeasurement(newmeasurement, 0.5));
      merged = true;
      break;
    }
  }
  if(!merged)
    M.addLast(newmeasurement);
  if(calculate)
    this.reCalculate();
  return true;
```

Node.calculateCoordinates

private void calculateCoordinates()

```
  double newlat = 0, newlon = 0;

  //alternative for empty set of measurements
  if(M.isEmpty()){

    //calculate new location of this intersection based upon average
       location of neighbouring nodes
    ArrayList<Node> neighbours = this.getNeighbours();
    for(Node neighbour: neighbours){
      newlat += neighbour.getLatitude();
      newlon += neighbour.getLongitude();
    }
    setCoordinates(newlat/neighbours.size() , newlon/neighbours.size());

    //calculate new spatial variance
    double summedsquarederror = 0d;
    for(Node neighbour: neighbours)
      summedsquarederror += Math.pow(this.distance(neighbour), 2);
    this.spatialvariance = (neighbours.size() > 1)? (summedsquarederror /
        neighbours.size()): Math.pow(neighbours.get(0).getAccuracy(),2);
  }

  //normal mode for non-junctions
  else{

    //calculate new location of this node based upon measurements
    for(MeasuredPosition measurement: M){
      newlat += measurement.getLatitude();
      newlon += measurement.getLongitude();
    }
    setCoordinates(newlat / M.size(), newlon / M.size());

    //calculate new spatial variance
    double summedsquarederror = 0d;
    for(MeasuredPosition measurement: M)
      summedsquarederror += Math.pow(this.distance(measurement), 2);
    this.spatialvariance = (M.size() > 1)? (summedsquarederror / M.size()
        ): Math.pow(M.getFirst().accuracy,2);
  }
```

calculateFrequency

private void calculateFrequency()

```
if(M.size() >= 2){
  dist = new TDistribution(this.getAmountMeasurements()-1);
  temporalintervalmean = ((double)(M.getLast().getTimeInMillis() - M.
      getFirst().getTimeInMillis())) / ((double)(dist.
      getDegreesOfFreedom()));
  double summedpopulationvariance = 0d;
  for(int i = 0; i<M.size()-1; i++)
    summedpopulationvariance += Math.pow(temporalintervalmean - (M.get(
        i+1).getTimeInMillis() - M.get(i).getTimeInMillis()),2);

  //unbiased estimate of interval-variance
  temporalsigma = Math.sqrt(summedpopulationvariance / ((double)(M.size
      ()-1)));
}
```

calculateTimeOfDeath

private void calculateTimeOfDeath()

```
long lasttime = M.getLast().getTimeInMillis();
switch (CartographyGlobals.getDecayMethod()){
  case solid:
    timeofdeath = lasttime + CartographyGlobals.getStaticTimeToLive();
    break;
  case dynamic:
    timeofdeath = Math.round(lasttime + getIntervalMean() + dist.
        inverseCumulativeProbability(1 - CartographyGlobals.
        getMinimalExistanceProbability()) * temporalsigma);
    break;
  case argmax:
    //let the solid value decay linearly over the amount of
        measurements
    long solid = lasttime + (CartographyGlobals.getStaticTimeToLive() *
         (CartographyGlobals.getMaximalNumMeasurements() - M.size())) /
         CartographyGlobals.getMaximalNumMeasurements();

    //dynamic value is determined by TDistribution
    long dynamic = (dist == null)? 0l: Math.round(lasttime +
        getIntervalMean() + dist.inverseCumulativeProbability(1 -
        CartographyGlobals.getMinimalExistanceProbability()) *
        temporalsigma);

    //take the maximum value of both methods
    timeofdeath = Math.max(solid, dynamic);
    break;
  default:
    timeofdeath = Long.MAX_VALUE;
    break;
}
```

C.4. Graph Comparison

The functions described in this section can be found in *rbs.graphcomparison.FastMCESGraphComparator*.

Compare

public StatisticalFrame Compare(Graph world, Graph map)

```
//Step 1: Matching junctions and endnodes
final Node[] worldnodes = getNodes(world, new NodeType[]{NodeType.
    junction, NodeType.endnode});
final Node[] mapnodes = getNodes(map, new NodeType[]{NodeType.junction,
     NodeType.endnode});
final LinkedList<Match> nearest4world = getNearest(worldnodes, mapnodes
    , true);
final LinkedList<Match> nearest4map = getNearest(mapnodes, worldnodes,
    false);
final Match[] bestmatches = getBestMatches(nearest4world, nearest4map);

//Step 2: Finding true positives and geometric similarity between map
    and world
final double[][] worldadjecency = makeUndirected(getAdjecency(
    bestmatches, world.getRoads(), true));
final double[][] mapadjecency = makeUndirected(getAdjecency(bestmatches
    , world.getRoads(), false));
final double[][] scoredmask = getMask(worldadjecency, mapadjecency,
    true);

//Step 3: Calculating precision and recall
double scoredprecision = getScoredPercentage(scoredmask, mapadjecency,
    map);
double scoredrecall = getScoredPercentage(scoredmask, worldadjecency,
    world);

// return the StatisticalFrame
return new StatisticalFrame(Globals.getNow().getTimeInMillis(),
    scoredprecision, scoredrecall);
```

getNodes

private Node[] getNodes(Graph graph, NodeType[] nodeTypes)

```
LinkedList<Node> result = new LinkedList<Node>();
for(Node n: graph.getNodes()){
  if(!(n instanceof rbs.world.designed.Node || n instanceof rbs.world.
      map.RoadNode) ||
       ((n instanceof rbs.world.map.RoadNode) && ((rbs.world.map.
           RoadNode)n).isReachable()) ||
       ((n instanceof rbs.world.designed.Node) && ((rbs.world.designed.
           Node)n).isVisible(Globals.getNow())))) {
    NodeType ct = n.getType();
    for(NodeType at: nodeTypes){
      if(at == ct){
        result.add(n);
        break;
      }
    }
  }
}
return result.toArray(new Node[result.size()]);
```

getNearest

private LinkedList<Match> getNearest(Node[] A, Node[] B, boolean aisworld)

```
LinkedList<Match> result = new LinkedList<Match>();
for(int i = 0; i< A.length; i++){
  double currentdistance, bestdistance = Double.POSITIVE_INFINITY;
  int bestBindex = -1;
  for(int j = 0; j< B.length; j++){
    currentdistance = A[i].distance(B[j]);
    if(currentdistance < bestdistance){
      bestdistance = currentdistance;
      bestBindex = j;
    }
  }
  if(bestBindex > -1){
    if(aisworld)
      result.add(new Match(A[i], B[bestBindex], bestdistance));
    else
      result.add(new Match(B[bestBindex], A[i], bestdistance));
  }
}
return result;
```

getBestMatches

private Match[] getBestMatches(LinkedList<Match> nearest4world, LinkedList<Match> nearest4map)

```
LinkedList<Match> chosen = new LinkedList<Match>();

//find all matches which are nearest for world AND for map
for(Match worldmatch: nearest4world)
  for(Match mapmatch: nearest4map)
    if(worldmatch.equals(mapmatch))
      chosen.add(worldmatch);

//generate a list of notchosen matches, which are later evaluated for
    possibility of adding to the chosen matches
LinkedList<Match> notchosen = (LinkedList<Match>) ((nearest4world.size
    ()>nearest4map.size())? nearest4world.clone(): nearest4map.clone())
    ;
for(int i = 0; i<chosen.size(); i++){
  for(int j = 0; j<notchosen.size(); j++){
    if(chosen.get(i).world == notchosen.get(j).world || chosen.get(i).
       map == notchosen.get(j).map){
      notchosen.remove(j);
      j--;
    }
  }
}

//sort the notchosen matches according to distance, and add these (
    removing any conflicting other options from the list)
Collections.sort(notchosen);
Match current = null;
while(!notchosen.isEmpty()){
  current = notchosen.removeFirst();
```

```
  //remove all conflicting, but inferior due to distance, options from
     notchosen
  for(int i = 0; i<notchosen.size(); i++){
    if(notchosen.get(i).world == current.world || notchosen.get(i).map
       == current.map){
      notchosen.remove(i);
      i--;
    }
  }
  chosen.addLast(current);
}
return chosen.toArray(new Match[chosen.size()]);
```

getAdjecency

private double[][] getAdjecency(Match[] bestmatches, Road[] worldroads, boolean forworld)

```
double[][] result = new double[bestmatches.length][bestmatches.length];
for(int i = 0; i<bestmatches.length; i++){

  //create a search queue for searching from bestmatches[i]
  PriorityQueue<SearchItem> queue = new PriorityQueue<SearchItem>();
  if(forworld)
    queue.add(new SearchItem(bestmatches[i].world, 0d));
  else
    queue.add(new SearchItem(bestmatches[i].map, 0d));

  LinkedList<Node> visited = new LinkedList<Node>();
  visited.add(queue.peek().node);

  SearchItem current;
  while(!queue.isEmpty()){
    current = queue.poll();

    //Find out if the current SearchItem is a goal
    int index = 0;
    while(index < bestmatches.length){
      if(bestmatches[index].equals(current.node)){
        break;
      }
      index++;
    }

    //if the current SearchItem is a goal, continue to the next
       SearchItem
    if(index != i && index != bestmatches.length){
      if(result[i][index] == 0 || current.distance < result[i][index])
        result[i][index] = current.distance;
      continue;
    }

    //else, find neighbours for designed (map) nodes
    else if(current.node instanceof rbs.world.designed.Node){
      for(rbs.world.designed.Node neighbour: ((rbs.world.designed.Node)
         current.node).getNeighbours()){
        if(neighbour.isVisible(Globals.getNow()) && !visited.contains(
           neighbour)){
```

```
            visited.add(neighbour);
            queue.add(new SearchItem(neighbour, current.distance +
                current.node.distance(neighbour)));
          }
        }
      }

      //else, find neighbours for world nodes
      else if(current.node instanceof rbs.world.map.RoadNode){
        LinkedList<Road> roadstocheck = new LinkedList<Road>(Arrays.
            asList(current.node.getRoads()));

        //retain only the possible roads, possibly a very time-expensive
            operation
        roadstocheck.retainAll(Arrays.asList(worldroads));
        for(rbs.world.Road r: current.node.getRoads()){
          if(r instanceof rbs.world.map.Road && ((rbs.world.map.Road)r).
              getRoadBlock() == null && ((rbs.world.map.Road)r).
              isAllowedFrom((RoadNode)current.node)){
            Node neighbour = ((rbs.world.map.Road)r).getDestinationFrom(
                current.node );
            if(!visited.contains(neighbour)){
              visited.add(neighbour);
              queue.add(new SearchItem(neighbour, current.distance +
                  current.node.distance(neighbour)));
            }
          }
        }
      }
    }
  }
}
return result;
```

getMask

private double[][] getMask(double[][] WA, double[][] MA, boolean scored)

```
double[][] result = new double[WA.length][WA.length];
for(int i = 0; i< WA.length; i++){
  for(int j = 0; j< WA.length; j++){
    double currentWA = WA[i][j];
    double currentMA = MA[i][j];
    result[i][j] = (WA[i][j]==0 || MA[i][j] ==0)? 0: scored? Math.exp
        (-(Math.abs(WA[i][j] - MA[i][j])*2)/(WA[i][j]+MA[i][j])): 1;
    if(Double.isNaN(result[i][j]))
      result[i][j] = 0d;
  }
}
return result;
```

getScoredPercentage

private double getScoredPercentage(double[][] edgesimilarity, double[][] adjecency, Graph graph)

```
//calculate the scored true positives
double scoredtruepositives = 0;
for(int i = 0; i< adjecency.length; i++)
  for(int j = 0; j< adjecency.length; j++)
```

```
    scoredtruepositives += edgesimilarity[i][j] * adjecency[i][j];

//divide scored true positives by (twice the) total length of roads in
   the graph and return this
double divider = 2d * graph.getLength();
return scoredtruepositives == 0 && divider ==0? 1: scoredtruepositives
   / divider;
```

List of symbols

τ time (ms)

μ average

σ standard deviation

$\angle$ direction (degrees)

$\emptyset$ an empty set

α σ-threshold for measurements

β distance-threshold between measurements

γ minimal fitness for point-to-edge matching

n size of local look-ahead

t variable for t-distribution

ν degrees of freedom

ζ maximum number of measurements in M

θ threshold for time to live TTL

sTTL threshold (ms) for static time to live

Graph type	node (v)	edge (e)
Regular	○	———
World	●	– – – –
Map	■	———
Measurement	▲	· · · ·

Icons for graph examples used in this thesis.

Glossary

(Balanced) F-Score
The balanced F-score is a method to calculate the harmonized mean of two values. We apply this method to precision and recall to obtain one value for the overall map quality. The following formula is used to calculate the F-score:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Global Positioning System (GPS)
The global positioning system is a space-based satellite navigation system with which receivers can determine their position on the surface of the earth and estimate the error of the measurement. ROMA was developed to operate with position data as provided using GPS signals. GPS measurements are visually represented as △.

Graph
A mathematical representation used to show connections between objects. Nodes (v/inV) represent the objects, and edges ($e = v, v, e/inE$) the connections between nodes. Within this thesis, graphs are represented as $G = (V, E)$ in written form. Nodes in regular graphs are visually represented as ○ and edges as ———.

(Vector) Map
The map developed by ROMA is created as a graph with nodes positioned using aggregated position measurements. The use of a graph to represent a road map makes this a vector map. Nodes in vector-maps are visually represented as ■ and edges as ———.

Maximum Common Edge Subgraph (MCES)
The maximum common edge subgraph is a graph G with as many edges as possible which is isomorphic to two subgraphs. We approximate this concept to find true positives in the developed map by matching edges from the world to the map and vice versa as described in chapter 3.

Precision
We define precision of the generated map as the fraction of edges on the map which is matched to edges on the ground truth map. We apply scoring based on similarity to compensate for difference in path-length. section 3.2 provides the mathematical details and formulae used to obtain a precision value.

Recall
We define recall of the generated map as the fraction of edges in the ground truth map which is matched to edges on in the developed map. We apply scoring based on similarity to compensate for difference in path-length. Section 3.2 provides the mathematical details and formulae used to obtain a recall value.

ROMA
The developed algorithm is called ROMA (Robust Online Map-generation Algorithm). We believe it to be robust since it can adapt to changes in the world. The online nature is found in the ability to process new measurements as they become available. The entire algorithm is a map-generation algorithm since it manipulates a vector map based upon collected position measurements. ROMA falls in the category of trace-merging algorithms [3] and its design is found in chapter 4.

Trace

A series of position-measurements in order of time of measurement used as input for ROMA. ROMA takes traces consisting of 5 consecutive measurements as input. Measurements in a trace are visually represented as △ and connected using dotted lines · · · ·.

World

We use a graph representation of the village of Zoeterwoude-Dorp as the ground truth for the experiments in <Chapter exp>. On this world we simulate traffic which is used as input for ROMA. Nodes in the world are visually represented as ● and edges as – – – –.

Bibliography

[1] G. Agamennoni, J. I. Nieto, and E. M. Nebot. Robust and accurate road map inference. In *2010 International Conference on Robotics and Automation*, pages 3946–3953, Anchorage, AK, May 2010. doi: 10.1109/ROBOT.2010.5509778.

[2] T. Balch and R. C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994. doi: 10.1007/BF00735341.

[3] J. Biagioni and J. Eriksson. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record: Journal of the Transportation Research Board*, 2291:61–71, 2012. doi: 10.3141/2291-08.

[4] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864, 2005. ISBN 1-59593-154-6.

[5] R. Bruntrup, S. Edelkamp, S. Jabbar, and B. Scholz. Incremental map generation with GPS traces. In *Intelligent Transportation Systems*, pages 413–418, Vienna, Austria, 2005. doi: 10.1109/ITSC.2005.1520084.

[6] C2C-CC. Car 2 Car Communication Consortium website, 2015. URL https://www.car-2-car.org/.

[7] L. Cao and J. Crumm. From {GPS} Traces to a Routable Road Map. In *Proceedings of the 17th International Conference on Advances in Geographic Information Systems*, pages 3–12, Nov. 2009.

[8] CBS StatLine. Motorvoertuigenpark; personenauto's per 1000 inwoners per regio in 2014, 2014. URL http://statline.cbs.nl/StatWeb/publication/?VW=T&DM=SLNL&PA=7374hvv&D1=0-5&D2=a&D3=l&HD=130614-1054&HDR=G2,T&STB=G1.

[9] CBS StatLine. Kerncijfers wijken en buurten 2009-2012, 2014. URL http://statline.cbs.nl/StatWeb/publication/?VW=T&DM=SLNL&PA=70904NED&D1=3&D2=10308&D3=2-3&HD=130614-1102&HDR=T&STB=G1,G2.

[10] Y. Chen and J. Krumm. Probabilistic modeling of traffic lanes from {GPS} traces. In *Proceedings of the 18th International Conference on Advances in Geographic Information Systems*, page 81, San Jose, California, 2010. doi: 10.1145/1869790.1869805.

[11] J. J. Davies, A. R. Beresford, and A. Hopper. Scalable, Distributed, Real-Time Map Generation. *Pervasive Computing*, 5(4):47–54, Oct. 2006. doi: 10.1109/MPRV.2006.83.

[12] D. H. Douglas and T. K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, Oct. 1973. doi: 10.3138/FM57-6770-U75U-7727.

[13] S. Edelkamp and S. Schrödl. Route Planning and Map Inference with Global Positioning Traces. In R. Klein, H.-W. Six, and L. Wegner, editors, *Computer Science in Perspective*, volume 2598, pages 128–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-00579-7.

[14] S. Edelkamp, D. Sulewski, F. C. Pereira, and H. Costa. Collaborative Map Generation - Survey and Architecture Proposal. In J. van Schaick and S. C. van der Spek, editors, *Urbanism on track : application of tracking technologies in urbanism*, pages 161–182. {IOS} Press, Amsterdam The Netherlands, 2008. ISBN 9781586038175.

[15] D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence*. MIT Press, 2008. ISBN 978-0262-062718.

[16] R. Grimaldi. *Discrete and combinatorial mathematics: an applied introduction*. Pearson Addison Wesley, Boston, 2004. ISBN 9780201726343.

[17] J.-H. Haunert and B. Budig. An algorithm for map matching given incomplete road data. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, page 510, New York, New York, USA, Nov. 2012. ACM Press. ISBN 9781450316910.

[18] F. Lima and M. Ferreira. Mining spatial data from GPS traces for automatic road network extraction. In *6th International Symposium on Mobile Mapping Technology*, São Paulo, Brazil, July 2009.

[19] H. Mayr. I-Navigate: Intelligent, Self-adapting Navigation Maps. In *14th Annual International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 397–404, Tucson, AZ, USA, Mar. 2007. doi: 10.1109/ECBS.2007.44.

[20] B. Niehöfer, R. Burda, C. Wietfeld, F. Bauer, and O. Lueert. GPS Community Map Generation for Enhanced Routing Methods Based on Trace-Collection by Mobile Phones. In *2009 First International Conference on Advances in Satellite and Space Communications*, pages 156–161. IEEE, July 2009. ISBN 978-0-7695-3694-1.

[21] D. Pfoser and C. S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In R. H. Güting, D. Papadias, and F. Lochovsky, editors, *Advances in Spatial Databases*, volume 1651 of *Lecture Notes in Computer Science*, pages 111–131. Springer Berlin Heidelberg, Berlin, Heidelberg, June 1999. ISBN 978-3-540-66247-1.

[22] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521–533, 2002.

[23] S. Rogers. Creating and evaluating highly accurate maps with probe vehicles. In *Intelligent Transportation Systems*, pages 125–130, Dearborn, MI, USA, 2000. doi: 10.1109/ITSC.2000.881029.

[24] S. Schroedl, K. Wagstaff, S. Rogers, P. Langley, and C. Wilson. Mining GPS Traces for Map Refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, July 2004. doi: 10.1023/B:DAMI.0000026904.74892.89.

[25] B. Thomas, J. Lowenau, S. Durekovic, and H.-U. Otto. The ActMAP - FeedMAP framework A basis for in-vehicle ADAS application improvement. In *2008 Intelligent Vehicles Symposium*, pages 263–268, Eindhoven, Netherlands, June 2008. doi: 10.1109/IVS.2008.4621126.

[26] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. {MIT} Press, Cambridge Mass., 2005. ISBN 9780262201629.

[27] F. van Diggelen. GNSS accuracy: Lies, damn lies, and statistics. *GPS World*, 9(1):41–45, 1998.

[28] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6): 91–108, Dec. 2000. doi: 10.1016/S0968-090X(00)00026-7.

[29] M. Wing. *Geographic information systems : applications in natural resource management*. Oxford University Press, Don Mills Ont. {;;Oxford}, 2nd ed. edition, 2008. ISBN 9780195426106.

[30] S. Worrall and E. Nebot. Automated process for generating digitised maps through GPS data compression. In *Australasian Conference on Robotics and Automation, Brisbane, Australia*, 2007.

[31] L. Zhang, F. Thiemann, and M. Sester. Integration of GPS traces with road map. In *Proceedings of the Second International Workshop on Computational Transportation Science*, page 17, San Jose, California, 2010. doi: 10.1145/1899441.1899447.