Performance Engineering of Route Planning Algorithms

University of Oulu
Department of Information Processing Science
Master’s Thesis
Rauno Kangastalo
20.11.2013
Abstract
The archipelago of Finland is especially difficult for a skipper. The huge amount of islands, islets, channels and shallow waters may be treacherous even for an experienced skipper. UpWind is a navigational software application intended especially for a sailboat skipper. It has been developed in the University of Oulu in a series of student projects, including the algorithms for the route planning.

Long-term algorithm calculates the route along the major channel system on a chart. Short-term algorithm calculates the set of laylines, both for starboard and board side of the sailboat, that shows the optimal route to sail in the current wind conditions. The data used in the route calculation is stored in a separate database on a separate machine. The data is either read to random access memory (long-term planning) or it is queried on runtime (short-term planning).

The algorithms have to run in a given time frame. They must not block the user interface nor cause delay in calculations that might lead to hazardous situations while sailing. Also, the resources on board are scarce, especially electricity. Even the smallest optimizations on electricity consumption have been considered important.

A literature study of the current methodologies used in evaluating software performance was carried out in order to find out if those methodologies or a combination of them could be utilised in evaluating the algorithms. The focus of algorithm evaluation was in finding out their time-complexity properties.

It turned out that most of the current performance engineering methods, like the methods based on queuing theory, do not serve very well to the evaluation of the algorithms. Those algorithms do not implement any queuing phenomena nor the current performance engineering methods scale down well to algorithm level. Most of the methods are aimed to aid in designing and evaluating of the performance in the early stage of the development. In UpWind application the algorithms have already been implemented. The algorithm analysis scales down very well to the implemented algorithms. Algorithm analysis together with running time measurements of the algorithms and their analysis were selected as the evaluation method.

The algorithm analysis results revealed that the growth-rate is either \( n \) or \( n^2 \) for all algorithms and sub algorithms. The running times of the algorithms are thus fast and no bottlenecks were found in them.

The measurements and the analysis of the measurement results revealed one severe bottleneck in the short-term route planning algorithm. A major portion of the total calculation time is consumed by database queries when checking for obstacles in between the sailboat and the destination checkpoint. The measurements result analysis combined with the results of algorithm analysis of the long-term route planning showed that the calculation speed is sufficient for current size map. If there is intention to use bigger maps, it is suggested that the algorithm is moved to its own thread, or otherwise indicate the ongoing route calculation to the user.

The proposed improvement to the short-term planning is to fetch a larger amount of obstacle data on demand. This trapezoid area of obstacles is limited by the master turning points, boat position and destination checkpoint. An algorithm for checking the obstacle-free route calculation should be developed in future. It was suggested that the calculation should only be done when wind conditions change more than a given wind angle or the skipper gets too far from the selected short-term route. It was also suggested that only one of the sides of the short-term route should be calculated if the
skipper starts to follow either route. A method and limits, for determining when the wind change or distance between boat and short-term route is enough to trigger a new calculation, should also be developed.

*Keywords*
software performance testing, algorithm performance, performance engineering, route planning algorithms
Preface

This master's thesis has been done at the University of Oulu, Department of Information Processing Science.

What a voyage! This all started in autumn 2012 from the project course where I got acquainted with the concepts of theory of sailing. As I was born and have lived by the sea all my life the idea on making thesis around sailing was tempting and of natural interest to me. I'm glad that I could give my contribution to this challenging project.

Without other people around me I could never have completed this thesis. I would sincerely like to thank first of all my professor Samuli Saukkonen in his excellent guidance and support for my thesis. His enthusiasm and wits combine to a powerful fuel. I would like thank my family and friends from the support and encouragement. Especially I would like to thank my beloved wife Jaana Kangastalo for her support and endurance!

Rauno Kangastalo

Oulu, November 8, 2013
## Contents

Abstract .................................................................................................................................................. 2  
Preface .................................................................................................................................................. 4  
Contents .................................................................................................................................................. 5  
1. Introduction ....................................................................................................................................... 6  
2. Research problem and method ....................................................................................................... 8  
  2.1 Research problem ......................................................................................................................... 8  
  2.2 Prior research ............................................................................................................................... 9  
  2.3 Phases of the research ................................................................................................................... 10  
  2.4 Research Method ........................................................................................................................ 10  
    2.4.1 Constructive Research Method .............................................................................................. 10  
    2.4.2 Design Science ...................................................................................................................... 11  
  2.5 Quantitative data analysis ............................................................................................................ 12  
3. Performance metrics .......................................................................................................................... 13  
  3.1 Metrics ......................................................................................................................................... 14  
  3.2 Summary ..................................................................................................................................... 15  
4. Software Performance Engineering .................................................................................................. 16  
  4.1 Petri-nets ..................................................................................................................................... 16  
  4.2 Queuing Network model ............................................................................................................... 18  
  4.3 UML Performance modeling ....................................................................................................... 19  
  4.4 Summary ..................................................................................................................................... 20  
5. Algorithms ......................................................................................................................................... 21  
  5.1 What is an algorithm? .................................................................................................................. 21  
  5.2 How algorithms are analysed? .................................................................................................... 22  
  5.3 Recursive algorithms ................................................................................................................... 23  
6. Route calculation algorithms and their analysis ............................................................................. 25  
  6.1 Short-term calculation algorithm ............................................................................................... 25  
  6.2 Summary of the short-term route calculation algorithms ............................................................ 28  
  6.3 Long-term algorithm .................................................................................................................... 28  
  6.4 Queuing behaviour of the algorithms ........................................................................................ 30  
7. Measurement of the algorithms' performance ............................................................................... 32  
  7.1 Requirements for the measurements ......................................................................................... 32  
  7.2 Description of the Measurement method .................................................................................... 33  
    7.2.1 Short-term measurement description .................................................................................. 33  
    7.2.2 Long-term measurement description .................................................................................. 34  
  7.3 Hardware and software setup ....................................................................................................... 34  
  7.4 Obstacle data ............................................................................................................................... 35  
8. Measurement results and their analysis ......................................................................................... 36  
  8.1 Short-term algorithm .................................................................................................................... 36  
  8.2 Long-term algorithm .................................................................................................................... 43  
9. Performance enhancements ............................................................................................................... 46  
  9.1 Trapezoid of obstacles ................................................................................................................ 46  
  9.2 One-sided short-term route ......................................................................................................... 47  
    9.2.1 Preliminary conditions and method ...................................................................................... 47  
10. Discussion ....................................................................................................................................... 50  
11. Conclusions .................................................................................................................................... 52  
References .............................................................................................................................................. 53
1. Introduction

Navigation of a sailboat may be tricky for any sailboat captain. Especially in the archipelago of the Baltic Sea with all its islands and shallow waters where you really need to know the waters, unless you have some other navigation aid. Navigation aiding software aimed for sailboat captains has not been in abundance, mainly due to the fact that similar archipelagos as in the Baltic Sea do not really exist in many other places in our globe.

Professor Samuli Saukkonen, enthusiastic sailor himself, has decided to tackle this deficiency. He started a project called UpWind in the University of Oulu in the department of Information Processing Science. This project has been conducted as a series of student projects for several years. Originally the software was implemented with Java programming language, but has been later on ported to Qt-framework.

The UpWind application is an application where user can see a chart of the water area, and can select a destination point where he or she wants to sail. The application then calculates the optimal long-term route to the destination point. The compound word “long-term” here means the route on the marked channel system on chart. It also refers to the final destination where the user wants to sail. The sailboat get it's location from the Global Positioning System (GPS) as coordinates. The GPS system is located physically and logically outside the UpWind application, and is connected by RS-232 port to the machine that runs the UpWind application. The GPS coordinates are provided as National Marine Electronics Association (NMEA) strings together with boat speed, boat heading, wind speed and wind angle data.

As the sailboat moves on sea, the application will calculate a specific, optimised, route called short-term route. Short-term route is comprised of laylines connected together. A single layline is a straight line on sea that points the optimal direction in the current weather conditions. That short-term route tells the skipper the optimal path on the sea to the nearest possible point in the long term route avoiding the obstacles. The algorithm calculating the short-term route seek to avoid the obstacles in the sea as it calculates the optimal path while taking in to account the current boat characteristics, GPS position together with wind angle and wind speed. (i Royo, 2010) The short-term algorithm runs automatically in fixed interval of time.

The heart of the UpWind application are the route planning algorithms that calculate the optimal routes in a given situation. The algorithms are already implemented and this research mostly focuses on the performance analysis of those algorithms. Namely, two algorithms exist: long-term route planning algorithm and short-term route planning algorithm.

Resources on a sailboat are very limited. As the UpWind application runs in personal computer on board where electricity is not in abundance, it is essential to minimize the power consumption. Measures have already been taken to diminish the power consumption. For example short-term route algorithm currently runs in 10 seconds interval, which should be a sufficient time for the sailboat captain to react as the speed of the sailboat is not that fast. But furthermore it is important to know how much calculation time the algorithms take and why. Which parts of the algorithm take up majority of the time, and how the calculation time could be optimised?
An algorithm is a computational method that takes in input data and give out the output data. (Cormen et al., 2009) Algorithms can be assessed from several different aspects. The efficiency of algorithms can be defined by its time complexity and space complexity. Time complexity simply means the algorithms efficiency in time domain; how long does it take to run through a particular set of calculations. On the other hand the algorithm may have a time requirement; that is the time that the algorithm may spend on some calculation. Time requirement may be absolute or proportional.

Absolute time requirement means that an algorithm has to finish in a certain amount of time, it may not spend more time. Proportional time requirement means that the time spent on calculations may grow in proportion to the size of input data. In computer science, space complexity means the memory consumption of those calculations. Similarly as with time complexity, an algorithm may have a space requirement i.e. algorithm has only certain amount of memory available. (Sedgewick & Flajolet, 2013) In this research the space complexity is not a subject of research.

An algorithm can be implemented in different ways. For example, an algorithm can be run serially one command at a time, or in parallel where more than one commands is run simultaneously. Algorithms and algorithm performance evaluation and effectiveness will be discussed in detail in Chapter 5. One very important aspect is the correctness of the algorithm. (For non-stochastic phenomena) With exactly the same input, the output of the algorithm should be exactly similar when calculations are repeated. This is the most important aspect in evaluating an algorithm. This research does not deal with algorithm correctness as they already have been proven correct in previous studies (i Royo, 2010).

Many methods for design, evaluation and measurement of software performance have been developed. Methods like queuing networks (Smith, 2002) and UML performance scenarios (Object Management Group, 2005) provide efficient guidelines to evaluate and implement good performance. Also automated performance evaluation tools are available.

The focus of this research is to find out the overall and detailed performance of the algorithms. The bottlenecks, in other words the parts of the algorithms that prevent or slow down the performance of the algorithm, are looked up and pin-pointed. In order to find out the performance issues some methodology for measuring and assessing the performance has to be introduced. In addition to this method, the design artifact of this research also are the introduced performance optimization ideas to the found bottlenecks.
2. Research problem and method

In this chapter the research topic are discussed and the selected research methods are introduced. Some previous research on the topic of software performance are studied, as well as algorithm performance and that knowledge will be reflected with the research topic. This will lead to the rationale of the decision of the selected evaluation methods.

2.1 Research problem

The UpWind application is a real-time software. It has been mostly implemented already. Hence, this research handles late-in-development performance engineering of the algorithms.

In this chapter it is studied how the previous research information of performance modelling and algorithm analysis can be combined to model, analyse and predict the route calculation algorithms performance with different data loads.

The research problem:

➢ How can the existing research of software performance modelling and algorithm analysis be combined and utilised in modelling and predicting the performance of the route planning algorithms?

The performance requirements for the algorithm's speed have not previously been presented for the algorithms. The requirements are time based, and so far one concrete requirement has only been specified for the short-term algorithm. That is the 10 seconds interval in which the calculation is triggered, and the calculation must be faster than the interval. Thus, in this research the focus is on the current performance of those algorithms and in identifying the performance bottlenecks.

The UpWind application as such is not sufficient to navigate a sail boat. An additional data-gathering part is needed. The data-gathering part (separate, embedded system) gathers the data from natural phenomenon like wind speed in real time, analysis component will run the algorithms in order to calculate the ideal route accordingly, and user interface part will show the results to the user. This research does not cover any aspects of the data gathering part to the performance of the algorithms as it is a separate device from which the information is transferred to the hosting computer via RS-232 port, and thus the UpWind project cannot contribute to it's performance.

This research will also seek answers to the following research questions. These questions will serve as basic level guidance in this research in answering the research problem. Questions one, two and three directly support in focusing on the research problem. Question four will offer a point of reference for the performance evaluation after the measurements have been implemented and conducted. Answer to question five will offer the code level structure of the algorithms and it forms the basis for modelling the algorithm's performance. It will also reveal the essential parts of the code in which the algorithms are situated for implementing the additional coding needed for the performance measurements. Questions from one to four will be answered in the phase
one (see Chapter 2.3). The questions from six to eight will test the achieved model for prediction and modelling of the algorithms performance.

1. How can the performance of a software component be measured as a whole?

2. How can the previous research on performance engineering be adapted to a component and sub-component level?

3. How can the performance of a software artifact be measured and how those artifacts contribute to the whole performance of the software component?

4. What are the performance requirements for the route planning algorithms?

5. What is performance of the current implementation of those algorithms?

6. What are the performance bottlenecks in those algorithms (if any)?

7. How may the performance be enhanced?

8. What kind of performance benefits can be acquired by those enhancements?

2.2 Prior research

In this chapter a short introduction to the prior research on existing performance analysis or modeling methods is provided. First, a short introduction to each method is given, and in the later chapters a more thorough description.

Research material on both software performance and algorithm performance is in abundance. A lot of the software performance research aims to assess the overall performance of a specific application. (Cortellessa et al., 2003; Balsamo et al.; 2003; Cormen, 2009) Also it is noteworthy that most of the latest research aims to automate the performance analysis. In many studies the premise is to tackle the performance issues early on in the development. However, the algorithms used in the sailboat route calculations on sea have already been developed.

Methods like stochastic Petri-nets, stochastic process algebra, deterministic Petri-nets, UML performance scenarios and queuing networks are commonly used for assessing the performance of a software (Koziolek, 2009). The first two method introduced by Koziolek (2009) address the stochastic manner of some given phenomena. Some of the methods are more suited for designing the performance. However, the software application under study in this research, UpWind, has already been developed and especially the route planning algorithms are already written. This research focuses on the methods that are more suitable for evaluating already existing implementations, and the focus of this chapter will be in identifying those methodologies.

Queuing network model is a graphical presentation in which the system is presented as a connected network of queues and servers. Queues represent the incoming flow of requests to the server. The history of queuing networks dates back to early 20th century as Danish scientist Agner Karup Erlang developed a theory for solving the delay problem in an early telephone network (Bojkovic et al., 2010). A special case for analysing performance bottlenecks are the finite capacity queues (Balsamo et al., 2003).

UML Profile for Schedulability, Performance, and Time Specification has been introduced to standardize the UML usage. This can be utilised in modeling the
performance and in determining how the system may be optimised in early development. (Object Management Group, 2005).

2.3 Phases of the research

This research is divided into four separate phases. These phases will give this research structure and rigor. In each phase the preliminaries and outcomes are defined and assessed regardless whether they were met or not.

Phase One

The performance engineering related literature was studied and the most relevant methodologies were selected for further study. The evaluation method(s) of the algorithms were selected.

Phase Two

The selection of the algorithm evaluation methods and their application in the algorithm evaluation are performed in this phase.

Phase Three

The analysis of the results from the evaluations. The performance of the algorithms are analysed. The possible performance bottlenecks are identified. If any performance bottlenecks are identified the suggestions to improve the performance are also described in this phase.

Phase Four

This phase will report the found performance bottlenecks and suggested solutions to improve the performance. This phase will conclude the research.

2.4 Research Method

In this chapter the research methodology that is applied to in this research is described. The principles and rationale behind the constructive research method are covered. Later in this chapter the constructive research method is compared to the design science methodology. The justification of design science as the chosen approach is discussed.

2.4.1 Constructive Research Method

Constructive research is an eclectic research method and can thus have many utilizing approaches. Many different concepts of the method can be found. This statement reveals us that not only one set of rules or some restricted multitude of equations can be applied to in order to perform a scientific study. On the contrary, many different approaches can be utilised. Both quantitative (Hevner et al., 2004) and qualitative approaches can be applied in research.

There exist many different approaches or rather guides how to do research using constructive method. Their methodologies do not so much contradict but have different basis on research. Some of them can be applied more generally and some are meant to be used in some generic discipline, here information science.
March & Smith (1995) divides the research field into two basic categories: natural science and design science. Natural science tries to characterize the natural phenomena by creating a common language to describe it. Natural science aims at creating “laws, models and theories” of our reality. Then those new “laws, models and theories” are tested to conform with some common norms of truth.

The counter part for this according to March & Smith (1995) is design science. Whereas natural science does not try to create anything new design science does. “Design science attempts to create things that serve human purposes” (March & Smith, 1995). March justifies design science by saying that if the purpose of science is to produce “credential knowledge” then “design science is an important part of it” (March & Smith, 1995). Design science aims at producing constructs, models, methods and implementations. By construct March mean the “basic language concepts” that can be used to describe the phenomena under research. Furthermore March & Smith claim that from design science arise new artifacts that can be further researched in academic setting (March & Smith, 1995).

2.4.2 Design Science

*Foundations of Design Science*

March & Smith (1995) have divided their approach on design science in two main domain: research activities and research outputs. Accordingly research outputs are Constructs, Models, Methods and Instantiation. Method describe the steps, in the language set by the constructs, how to execute a task. Instantiation is simply the implementation of the suggested result. (March & Smith, 1995)

The research activities are Build, Evaluate, Theorize and Justify. A researcher builds an artifact to see if his idea(s) work. As the work progresses and the newer artifact is introduced, those artifacts can be compared and evaluated if there has been any progress. Evaluation needs metrics or other means to measure the progress, thus metrics define the goal of the research. Out of evaluation can be drawn theories, which can be justified further. Justification means that more information has to be gathered to support the theory. (March & Smith, 1995).

*Design Science in this research*

In this research the design science approach will be utilised as the research method. Design science aims at producing constructs, models, methods and implementations. (March, 1995) It leans on the socio-technologist approach. Hevner (2007) has defined a framework of three design science cycles that will be utilised as a structural framework in my research. The three cycles, relevance cycle, design cycle and rigor cycle, give guidance through this research as it gives clear structure to the research work. (Hevner, 2007).

In the relevance cycle the requirements and acceptance criteria for the research are defined. It is an iterative process as the term “cycle” suggests, and where the requirements and acceptance criteria for the research results are not only defined but also evaluated and revised. (Hevner 2007) In this research the relevance cycle will be performed and utilised constantly in assessing the different performance analysis methods and the outcomes of this research. The requirements for the research will be gathered, and also the acceptance criteria for the research results in phase one. The
solution for the research problem will be evaluated in the phase two. There will be an opportunity to revise. Phase two will also conclude the relevance cycle.

The rigor cycle gives the research a solid ground to stand on. In this cycle the existing information on the research topic is gathered and analysed. This gathered material acts as a basis for the creation of new information and ideas during the research effort. (Hevner, 2007) In this research rigor cycle is present in every phase to find the needed information for backgrounding and for supporting the work.

The requirements from the relevance cycle and the background information meet in the design cycle. That information is constantly utilised in creating the design artifact and in evaluating the outcome of that design. (Hevner, 2007) In this research design cycle situates in phases two, three and four. In this cycle the research problem will be answered as a resulted method for assessing the algorithms. Then this resulted method will be used in estimating the algorithms' performance.

The performance measurement enablers to the source code will be implemented and utilised in the performance measurements of the two algorithms. Then the results from the measurements (of different datasets) will be used to analyse how well the selected performance evaluation models correspond to the acquired results. That information can be used in evaluating the performance of the method that was created as a response to the research problem. Hence, here the design cycle is a three layered framework where information from child layer will be utilised in analysing the parent layer.

2.5 Quantitative data analysis

The performance measurements provide a large amount of statistical data of the performance of the algorithms; calculation times of different parts of the algorithms and repetition of some functions or procedures. This information needs to be analysed and interpreted. The primary data of the running times are collected with sufficient population for the analysis. The whole population of data is be collected, no sampling is used.

Statistical data analysis and interpretation principles can be used in conjunction with measurements. It will reveal the amount of consumed time on each function or procedure recorded in the measurements. The principles of quantitative research method can be used in interpreting the results. Bivariate descriptive analysis is utilized in analysing the results. Variance, covariance, mean, standard deviation and mode can be calculated to describe the characteristics of the acquired data population. (Sims, 2000) Also statistical analysis can be utilised in calculating the proportions or percentages of each individual part of the algorithms or the procedures can be acquired from the data. (Card et al., 2002)
3. Performance metrics

Software performance has been a hot topic since the invention of the computer. The question is how to efficiently use the computing resources of a particular computer. As the calculation capacity has grown over time (as Moore's law suggests the amount of transistors on a microchip will double in every two years. (Moore, 1965)) also the requirements by the software have been growing very rapidly. In other words modern day computers spend a lot of computer resources, and do not always spend the resources efficiently. Especially hot topic in the early PC-era in 1990's was the adequacy of random access memory (RAM). As the technology evolved and the price of the memory chips fell, it was claimed that computer applications started to waste memory.

As the software systems are getting gradually more and more complex and vast, the importance of the software engineering grows rapidly. Smith & Williams (2002) emphasize in their book that it has been a common mistake that performance related problems would be as easy to fix as coding errors. Software performance engineering is considered a quantitative approach to meet performance requirements starting from the beginning of software development. (Smith & Williams, 2002) It also includes the approaches on describing and improving the performance. (Woodside et al., 2007). If performance issues are not taken into account from the beginning of the development, the cost of fixing performance problems might be high. Woodside et al. (2007) agree, and also emphasize the importance of early specification of performance requirements. Woodside et al. (2007) give a concrete list of activities that have to be performed when initiating a software project. The activities can be found from Picture 1. below. The important activities for this research, as the implementations already exist, are the Performance Requirements, Performance Test Design and Performance Testing. The other activities are either not appropriate or cannot be be applied in the UpWind case. For example analysis of the early performance models cannot be done because the algorithms have already been implemented and the early performance models do not exist. The total system analysis is out of scope because this research concentrates on algorithms that are only a small part of the system.

![Picture 1. The different activities in software engineering process. (Woodside et al., 2007)](image-url)
The other measurable variable of software performance is consumed time. Now the relevant question narrow down to question how fast a particular calculation can be executed and how fast is fast enough? Despite the fact that the amount and speed of memory chips and processors increase this question has still been a hot topic. Many solutions to the speed problem have been introduced, such as using multiprocessor architecture and ready solutions for many algorithmic problems (Sedgewick & Flajolet, 2013).

Smith & Williams (2002) state that software performance is anything one can measure with a stop watch. For this research this simple statement is enough: in this study the topic of memory consumption of the route calculation algorithms is not handled. That statement act as a tree trunk that will grow more sophisticated branches.

The performance should be planned from the very beginning of a software project. The fixes to an application late in the development phase may be very expensive. (Smith & Williams, 2002). Alas, both the long-term and short-term route calculation algorithms have already been developed in earlier research (i Royo, 2010) thus the work in this research is concentrated on finding out the late-in-development properties of the performance of those two algorithms.

3.1 Metrics

Adan & Resing (2001) introduce performance measures based on queuing theory that have later been applied to other research topics including software performance engineering (Smith & Williams, 2002). The metrics by Adan & Resing are the sojourning time, distribution of the number of customers in the system, the distribution of amount of work in the system and distribution of the busy period of the system. The sojourning time stands for the total time a customer spends in a system: the waiting time and service time.

Little's law gives the queue length (Smith & Williams, 2002) when the mean sojourn time and the arrival frequency of the customers are known (Adan & Resing, 2001). Smith & Williams (2002) use a bit different terms, which should be used in this research as they are well-established terms in software performance engineering. Residence time is the same as Adan's sojourning time. Throughput is the average rate of job completion as function of time. Utilisation is the average percent of time that a service is busy. Queue-length is the average number of jobs at the server, waiting or getting service. By calculating the average time the server (in this case an algorithm) spends in servicing a customer some other metrics can be easily calculated. Those metrics can then be utilised in analysing the performance with different arrival rates.

Smith & Williams (2002) introduces metrics for acquiring the characteristics of a system. These metrics are general and can be applied on a single system resource over a known period of time. Utilization, throughput, mean service time, residence time and queue-length describe the system's performance behaviour in the metrics by Smith & Williams (2002).

These metrics can be utilised in the analysing the current performance of a system but also in predicting the performance with different data loads. However, they cannot be applied on a queuing network model where several system resources are interconnected. The average values describe the system. The variance of the distribution is zero, and it is meaningless to try to utilize any probability theories. These metrics are a good analysis tool for deterministic queues. The characteristics of a system can be acquired by actual measurements of the system performance. The metrics can be calculated with the
measured values and the system performance may be predicted with imaginary or expected datasets. Smith & Williams (2002)

The queuing network model in detail will be covered in Chapter 4.2. Two kinds of queuing networks exist: open queuing network model and closed queuing network model. They both have different metrics. Smith & Williams (2002)

In open queuing network the arrival rate can be random, bursty or constant having some distribution. Special case of arrival rate distribution is the Poisson process. Poisson process can be used to describe average rate of arrivals that is either random or constant. The probability of an arrival at some instant is zero and there is no way to predict the probability for the arrival at any given moment. Thus, the Poisson arrival has been defined as a sequence of interarrival times. (Adan & Resing, 2001) The arrivals to the long-term algorithm are random, the interval determined by user clicks on the user interface. But as the inter-arrival rate is quite slow, it is not necessary to analyse the system as an open-queuing model.

Smith & Williams (2002) give equations for acquiring the metrics from open queuing network model. Device throughput is the system throughput multiplied by number of visits to the device. This simply means that the system consist of devices, and if some device (in UpWind algorithm case a method acts as a device) is visited more than once per device execution, the amount of throughput is the amount of device throughput multiplied by the system throughput. Device utilization is the device throughput multiplied by average service time of that device. Residence time of a visit to device describes the time a client spends at service. Now the queue-length of the device can be calculated by multiplying the arrival rate with residence time. The total queue-length for the system is the sum of the device queue-lengths. The system response time is the Little's formula. If the algorithms are proven to be deterministic and their queuing network model is also proven to be open, these metrics can be applied quite directly.

\[ E(L) = \lambda E(S), \]

where \( E(L) \) is the mean number of customers in the system, \( \lambda \) is the arrival rate and \( E(S) \) is the mean waiting time. (Adan & Resing, 2001)

Formula 1. Little's formula.

As described in Chapter 4.2 closed queuing network model is stochastic. Thus, probability theory principles have to be applied. But neither of the algorithm are stochastic, thus this approach can not be utilised neither. In closed model the probabilities of a transition leaving service i arriving in service j can be solved from the transition matrix \( P \). (Jensen & Bard, 2003)

3.2 Summary

The metrics for performance analysis in software engineering heavily lean on the queuing theory. Whether or not to use metrics in the evaluation of the algorithms depend on their behaviour in runtime. The route-calculation algorithms require closer study to find out if they implement a queuing network. The most of the metrics require acquiring the average characteristics of the server. For deterministic systems no probability theories cannot be used, but the metrics provided by Smith & Williams (2002) are enough. Both the open and closed queuing network models have their own metrics.
4. Software Performance Engineering

Many approaches have been developed for modeling and measuring the performance parameters of software systems. Many of the modeling methods are probabilistic and require usage of probability theories like Markov chains. A method that is directly suitable for analysing the algorithm or a method that could be applied with changes in the analysis need to be identified. In this chapter the different approaches determining software performance in detail is discussed. The summary chapter concludes the different methods and assess their meaningfulness in assessing and modeling the performance of the route calculating algorithms.

4.1 Petri-nets

As software engineering is such a wide field of research a variety of software engineering methods for performance engineering have been developed. One of the very first methods for assessing information processing systems were the Petri-nets, their purpose is to work as a tool for qualitative analysis of processes. (Balko, 1993) Soon their usefulness was discovered especially in modeling and analysing parallel and concurrent systems. Petri-nets can be utilised in performance analysis for any system that can be described graphically like flow charts and the description is required for representing concurrency or parallel computations. (Murata, 1989) As a reflection to the UpWind application there is quite a lot of concurrency indeed. In Qt-framework the user interface should always run in it's own thread and other computations should run in some other threads (Wysota, 2009). This is the case in the UpWind application.

Petri-net is a directed graph i.e it has a initial state in Petri-net language known as marking $M_0$. (Johnsonbaugh & Murata, 1982) Marking $M(p)$ denotes the amount of tokens in a place.

Petri-nets consist of two kinds of nodes: places (P) and transitions (T). Places can be of two variety: input places and output places. Input places may include information of preconditions, input data, input signals, needed resources, conditions and buffers. Output places may correspondingly include information of post conditions, output data, output signals, released resources, conclusions and buffers. Graphical notation of places are circles. (Murata, 1989)

Transitions represent events, computational steps, signal processors, tasks or jobs or clauses in logic and processors. Transitions may have multiple input and output arcs that represent the pre-conditions and post conditions of the transitions. Transitions are enabled if they are connected by incoming arcs that have place with at least one token on the sending end of the arc. (Johnsonbaugh & Murata, 1982) Enabled transition may (or may not) fire if the event takes place. When fired tokens are removed from the input place by the amount indicated the weight (W) of the input arc, $w(p,t)$ and are added to the output places by the weight of the output arc $w(t,p)$. Special cases of transitions are transitions without input place or output place. The transition without input place is called a source. The transition without output place is called a sink. Firing this kind of transition only consumes the tokens the input places have but do not produce any tokens on their own. (Murata, 1989)
Places and transitions are connected with arcs. An arc can only go either from a place to a transition or from transition to place, so for example if you go from input place to output place there has to be a transition in between. The arc is always directed indicating the direction in which the processing proceeds. (Murata, 1989)

Stochastic Petri-nets are widely used in analysing complex systems where many different possible states exist or execution can be random. In stochastic Petri-nets the delays of transitions or places are always probabilistic, hence stochastic. (Murata, 1989) Probability theory has to be applied in the analysis of stochastic Petri-nets (Marsan et al., 1998). Petri-nets without any probabilities in delays of transitions and places are called deterministic timed Petri-nets (Murata, 1989). Marsan et al. (1998) focuses more in stochastic Petri-net analysis. The justification why stochastic analysis model is not usable in the UpWind case will be covered and elaborated shortly.

Marsan et al. (1998) state that due to the probabilistic nature of stochastic Petri-nets some aspects of probability theory have to be taken into account when determining the firing in a given circumstance. Probability theory is needed to solve the Petri-nets.

Deterministic Petri-nets do not involve any probabilistic behaviour. Assume firing delay $\tau$ as the time from the initial firing back to the situation where the fired sequence is again firable (all transitions in sequence are fired at least once). This period of time is called cycle-time. The resource-time product is the number of tokens in a place and the time the tokens are within the place. With deterministic Petri-nets the minimum and maximum cycle times may be determined. Murata (1989) The needed calculations are quite complex for Petri-nets with variating transition weight.

From the structure of Petri-net it can be deduced that it is very useful tool when analysing state-machines and transitions between states. Stochastic Petri-nets are a good tool for determining the path that some execution of a state machine is likely to take, as Marsan et al. (1998) point out. Even for generalised Petri-nets it would be required to measure the actual delays in a system under study, and then the equation could be used to calculate the fastest cycle-time for the system. The route calculation algorithms in UpWind application may be considered as really simple state machines that are either calculating or idle. The algorithms could be chopped into pieces and this could be followed by measuring the running time for each piece. But that would not make any sense: Measuring the running time for one calculation would be equally easy.

The both route planning algorithms act quite similarly when it comes to firing the transitions in the algorithms, with one difference. The long-term route planning algorithm only has immediate transitions. As soon as user clicks on the destination point he or she wants to reach, that coordinate is passed to the algorithm and Dijkstra's shortest path algorithm is applied to calculate the path. There is no pauses in the execution, each method is executed once called, without any delays in place. However, in short-term route planning algorithm, the first transition is timed. The algorithm itself is timed, currently running at 10 seconds interval. As soon as the thread where the algorithm performs is started, the timer starts running. That is the first transition, and it is timed. After that all transitions are immediate until the execution is back at the first transition where there is again timed transition.
Example of a Petri-net. The implementation of the class that implements the short-term calculation algorithm as a Petri-net. The Petri-net was drawn with PIPE v2.4.1 application.

As it can be seen from Picture 2, the underlying Petri-net model of the short-term calculation is quite complex and detailed. Each place represents a method performing a particular task. Transitions represent the transitions between places. As it can be seen from the picture there are not any states in the algorithm. It can be seen that all transitions are synchronised, there is no concurrency. It can also be seen that all transitions are immediate. What cannot be seen is the execution order of the transitions. From the actual algorithms it can be seen that the transitions between the places are sequential and strictly ordered. That also means that there are not any stochastic firings, but every transition is deterministic with weight of value one. Thus, Petri-nets are lacking the presentative power to describe the low level software architecture in this case.

4.2 Queuing Network model

The model consists of servers and customers, that are coming to get service, comprising a queue. The arrivals on a queue usually arrive at random order and the arrival rate can be determined with the probability theory. The queuing theory focuses on systems or processes with single queue. The queuing theory base on the many different probability distribution theories like Poisson distribution (calculating the probability for some events to occur in some fixed interval of time for a given number of events with a known average rate occurring in a fixed interval of time) which are based on the Markov chains. (Adan & Resing, 2002)

In the past queuing networks were used in evaluating hardware or manufacturing systems. But later it has been applied to software systems as well. (Cortellessa et al., 2003). Queuing network model is a high level notation where the details of the customers and servers are abstracted. These abstractions can then be analysed for revealing concurrency and synchronization. (Balsamo et al., 2003)
There are two basic actors in queues: customers and services. Customers arrive to receive service. Adan & Resing (2001) describe queues with following terms: the arrival process of the customers, the behaviour of the customers, the service times (of the servers), the service discipline, the service capacity and the waiting room. Smith & Williams (2002) adds workload intensity and service requirements that are more specific terms for queuing networks.

The arrival process describes the arrival times to the server. The arrival times are individual and have some common distribution. The arrival might occur one by one or they come in bigger numbers. The customer may be willing to wait until it gets service or it will leave the queue. This can happen for example if the waiting time is too long for a particular customer. The customer even may not even be allowed to the waiting room if it is already full. The waiting time can be prolonged if the service has too much customers for it's capacity. The customers may be served one by one or multiple customers are served simultaneously. The service may have some priority for the customers, or the customer may get service by Last-In-First-Out (LIFO) principle or by random order. (Adan & Resing, 2001)

The queuing network model can be divided into two subcategories: open and closed queuing network models. Open queuing network means that customer enters from outside of the system and leaves the system upon completion i.e. in software engineering the system is called for service and might be abandoned afterwards. Instead, in closed network the customer is internal, it does not enter or exit the system. (Smith & Williams, 2002) Closed queuing network can be understood as a loop of servers. This means that in the open model the customers change over time. In comparison, in the closed model the same customer circulates in the system.

Some important performance measures comprise an interesting part of the queuing theory. These performance measures are a direct basis for the performance metrics presented by Smith & Williams (2002). The sojourning time can be calculated as waiting time plus service time. The distribution of the amount of work in the system tells the total sum of waiting times of the customers and the time left for the customer getting service. The distribution of the busy time in the server tells the time the server has serviced constantly, without any breaks. (Adan & Resing, 2001)

The classes that comprise route calculation algorithms in the UpWind application both only serve one customer at a time, thus the queuing network model is very simple. The long-term algorithm can be considered as an open queue network where the first node is outside of the class (Smith & Williams, 2002). The algorithm is called on demand and after execution it will be destructed from the memory. The short-term algorithm could be considered as a closed queue network as it runs constantly at given interval (delay). The first node of the network is actually inside the class (method “start”) and it triggers the algorithm and upon completion the execution returns to that same method. But according to Smith & Williams (2002) closed network model has no external arrivals or departures. The short-term algorithm does have them both, arrivals and departures. It receives the current NMEA data describing the GPS position of the boat together with wind conditions data and the long-term route vector, and it returns the calculated short-term route. Thus, the queuing model is not a good approach for the short-term algorithm analysis.

4.3 UML Performance modeling

A specific extension to the UML standard is the Profile for Schedulability, Performance and Timing. The standard include a description of Performance Modeling which
designed for getting the performance requirements and performance related characteristics from a design in early development phase of a system. This information can be used in determining how the existing system performance can be improved and on the other to reveal performance bottlenecks. The notation is based on the existing UML building blocks. (Object Management Group, 2005)

The UML Performance Models can be interpreted by different analysis methods like (stochastic) Petri-nets and (stochastic) queuing networks (Woodside et al., 2005). The model consists of domain concept details that describe the performance of the system like performance context and performance resources. (Xu et al., 2003) The model provides different kinds of methods for describing the system performance factors like sequence diagrams, activity graphs and hierarchical activity diagrams. Sequence diagrams have also been utilised in analysis of queuing models. (Balsamo et al., 2003) This approach will not be covered in detail because UML modeling does not include building blocks for such fine-grained contexts as algorithms. UML Performance Modeling is an excellent tool for designing and assessing a complete system. But when it comes to the route calculation algorithms of the UpWind application this model simply lacks the tools for describing them.

4.4 Summary

Woodside et al. (2007) quite heavily criticise the current methods for performance engineering and especially analysis. The methods require quite a lot of work, and are quite complex to understand. Woodside et al. declare that many developers do not trust or understand the performance models. The complexity of the evaluation of Petri-nets and queuing networks, especially in the cases where probability theory and Markov chains are taken into account, inhibits their use in wide scale. Petri-nets and queuing networks are tools for evaluating a large system in higher level and in most cases they require commercial software to calculate the performance.

Many of the methods introduced in this study need specific analysis tools for analysing the performance of a given software. Majority of those tools are implemented for academic purposes, though some commercial and shareware tools exist, like PIPE for drawing and analysis of Petri-nets. Without automatised tools it is quite difficult to acquire information on the performance of the system under study.

However, the queuing theory could give an interesting and simple basis for performance evaluation if the Little's formula and it's derivatives are combined in a way that could be utilised in detailed analysis of the route calculating algorithms. However, as previously described neither long-term nor short-term algorithms conform to the queuing theory. For this reason these methods have to be omitted.
5. Algorithms

In this chapter algorithms and algorithm analysis is discussed in detail. Also description of the route calculation algorithms is given and their behaviour is analysed. The underlying time-complexity of the algorithms is be revealed and their impact on the calculation performance is analysed.

5.1 What is an algorithm?

An algorithm is a detailed instruction of a task or process that can be followed in order to solve a particular problem. Algorithms are used in a huge variety of purposes in many different areas of science and engineering, not only in the fields of mathematics and computer science. In computer science an algorithm is a programmatic procedure that takes in some input and returns some output. The procedure consist of multiple parts called steps. The steps may include conditions and commands. (Cormen et al., 2009) In this study algorithms are considered in the context of the domain of computer science.

Algorithms are analysed in order to discover how they behave. The classical analysis of algorithms in computer engineering seems to concentrate mostly on the speed of the algorithms calculating large (enough) data sets. (Sedgewick & Flajolet, 2013; Cormen et al., 2009) As the analysis methods have been developed, more efficient algorithms have been created as the result. Cormen et al. (2009) emphasize especially the importance of recurrence).

An equally important aspect in algorithm analysis is the correctness of the algorithms, that is, the requirement that an algorithm produces correct output with any given input data and stops when calculation is ready. However, this research will not pursue the correctness of the route calculation algorithms as it already has been pursued in research conducted previously (i Royo, 2010).

The efficiency of an algorithms is very important and many methods for the analysis of algorithms have been developed. In computer engineering the algorithms' efficiency can be split in two parts: time domain behaviour and space domain behaviour. (Sedgewick & Flajolet, 2013) The most simple approach to evaluate the efficiency of an algorithm is to implement the algorithm programatically and then run it with different data sets and measure it's efficiency on that platform. Partly, the efficiency of an algorithm can be considered depending on the platform they are being run on: the amount of memory, the processor speed (instructions per second) and the processor's instruction set has an effect on the efficiency. (Cormen et al., 2009)

As already stated this research focuses on the calculation speed of the route calculation algorithms, and in this research the space domain behaviour will not be covered. However the different characteristics of the different platforms makes it very difficult to assess the effectiveness of an algorithm. Hence, more sophisticated methods for assessing algorithms' speed have been developed. Algorithms are also analysed, besides in order to get their current properties, to find out if they can be improved and how.

In information processing science and computer science, and also in this research, the algorithms are often presented in pseudo-code. (Cormen et al., 2009)
makes the algorithms more readable but also hides some details of their actual implementation. The pseudo-code presented in this research is taken directly from the actual implementation of the algorithms with simplifications and trimming.

5.2 How algorithms are analysed?

The running time analysis of an algorithm can be roughly categorised into three approaches: best-case, average-case and worst-case analysis (Sedgewick & Flajolet, 2013). Cormen et al. (2009) largely omits the best-case analysis: best-case analysis means the performance in optimal conditions and hence does not describe the algorithms performance in a way that could be utilised in assessing the bottlenecks. For example if the input data for a sorting algorithm is already in a sorted order, it may be that the algorithm does not run at all. There hardly is any sense to analyse that kind of situation. Both Cormen et al. (2009) and Sedgewick & Flajolet (2013) rather concentrate on worst-case and average-case algorithm analysis.

Assessment of computational complexity helps in classifying the algorithms performance characteristics (Sedgewick & Flajolet, 2013). As this research is focused on analysing the calculation speed of an algorithm, it is very important to find out how a particular algorithm behaves with different input data loads. Especially, how the calculation time depends on the input data and how the time grows as the size or quality of the input data changes. This is called the growth of a function. It describes how the running time changes as a function of the input data. In order to make the growth of the running time relevant the size (n) of the input data has to be large enough. This is called asymptotic efficiency of an algorithms (Cormen et al., 2009) meaning the limits of the growth of the running time. The algorithm is split into steps that each represent a sub-task in the (sub-)algorithm denoted by c.

The growth of the running time may be limited either from above, from below or from both below and above. Correspondingly big-Θ-notation describes the limits for the growth of the function from above and below in such way that the growth of the function f(n) may be equal to the either bound but may not cross the limits. The exact definition is defined in Formula 2. In the formula c₁ is the lower bound, c₂ is the upper bound and g(n) is the function that is examined. The function f(n) is said to be asymptotically tight bound, and this requires that every variation function is asymptotically non-negative for all sufficiently large n. (This means that the value of the function may also be zero, where as in asymptotically positive the value of the function is always positive).

\[
\Theta(g(n)) = \{ f(n): \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}
\]

Formula 2. θ-notation (Cormen et al., 2009).

The big-Ω-notation limits the growth from below in such a way that the growth of the function may be equal to the bound but may not cross it. (Cormen et al., 2009) Formula 3 defines the Ω-notation, now c is the lower bound of growth of function f(n).

\[
\Omega(g(n)) = \{ f(n): \text{there exists positive constants } c \text{ and } n_0 \text{ such that }
\]

\[ 0 \leq c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0 \]  

**Formula 3.** O-notation (Cormen et al., 2009).

The third notation is the big-O notation. It limits the growth from above in such way that the growth of the function may be equal to the bound but may not cross it. (Cormen et al., 2009) O-notation is defined in Formula 4. \( c \) denoting the bound that growth of function \( f(n) \) may not cross from below.

\[ O(g(n)) = \{ f(n) : \text{ there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

**Formula 4.** O-notation (Cormen et al., 2009).

The little notations are correspondingly little-o notation and little-\( \omega \) notation. These notations correspond to the “big” notations, excluding the value equal to the limit: the growth of function never “-touches” the limit. With these notations the limits of the growth rate of the function can be described. (Cormen et al., 2009) The growth rate is considered to be dominated by the highest order term for sufficiently large \( n \).

The growth rate is usually considered to be exponential, in example \( n \)-fold \( (n^m) \), polynomial \( (n^m + 1) \) or logarithmic \( (\log(n)) \). Logarithmic growth rate is the inverse of exponential growth, and thus the growth rate is slow. Exponential growth rate grows ever faster as the exponent grows opposed to the logarithmic growth where the growth will start steeper and then slows down as the coefficient (“exponent”) grows. Thus logarithmic order of growth is considered more efficient. In algorithm analysis the effect of the lower terms are considered meaningless when the \( n \) is sufficiently large. The higher order terms will dominate the growth of the function.

As the analysis proceeds, first the different steps are detected from the pseudo-code and their effect to the running time are assessed. Repetition is especially important to discover. Then the highest order term is determined.

How to determine when the \( n \) is sufficiently large? There is not any unambiguous equation to determine it. Depending on the hardware setup, the calculation time may vary. Also, the requirements must be taken into account when determining the calculation speed for a given data set. Let's assume a order of growth with \( \Theta (n^2) \). This kind of algorithm grows in quadratic order. So calculation speed quadruples when data size doubles and continues to grow quadratically as data size grows. If the calculation speed for data of a given size is known, the maximum data size that meets the required calculation speed could be determined.

### 5.3 Recursive algorithms

In recursive algorithms the problem is divided into smaller instances of that same problem which Cormen et al. (2009) calls subproblems. Then the subproblems are solved by calling the function recursively and finally the results are combined if needed. Though none of the algorithms used in the route calculation use recursion in the UpWind application, it is beneficial to study the concept. If bottlenecks are encountered in the algorithms, recursion may offer a way around it.

Recursive algorithms may be faster than iterative algorithms for large data sets. Again, the question is how big is large? There is no any unambiguous answer to that question.
The intuitive method is to create both a recursive algorithm and an iterative algorithm, analyse them and compare the time complexity.

The book by Cormen et al. (2009) represents three different ways of analysing the recursive algorithms’ time complexity. Let’s take a look at the last method called “the master theorem”. In the master theorem recursive algorithms are categorised in three categories. Cormen et al. (2009) defines the recursion in Formula 5, as follows:

\[ T(n) = aT\left( \frac{n}{b} \right) + f(n), \text{ where } a \geq 1 \text{ is the number of subproblems and } b > 1 \text{ is the size of the original problem and where } n/b \text{ is either ceiling or floor function} \]

**Formula 5.** Definition of recursion by Cormen et al. (2009)

Now the recursion has asymptotic bounds as follows. (Direct quote from Cormen et al.)

1. If \( f(n) = O(n^{\log_b a - \varepsilon}) \) for some constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \). Recursion dominating the running time of the algorithm. Growth of the function \( f(n) \) is bounded from below and above.

2. If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \). Growth of function is bounded from both below and above.

3. If \( f(n) = \Omega(n^{\log_b a + \varepsilon}) \) for some constant \( \varepsilon > 0 \), and if \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \). Growth of the function is linear, and bounded from below and above.

All three cases have a common factor \( n^{\log_b a} \) that describes the runtime of the recursive term in a formula. Solving that factor results the growth of the function \( f(n) \). Not all recursions can be solved by the master theorem. Those recursions may be solved by recursion trees or by substitution method.

Recursion tree is a layered construct where each layer represent one run time of the recursion. Recursion tree is formed by splitting the recursion in to parts, and the obtained parts are progressively split until the recursion stops. To obtain the growth-rate of the function the running time of each layer are first summed up. After that the sums of each layer are also summed up to obtain the growth of the function \( f(n) \).

Another method (that can be used together with recursion tree) is the substitution method. In substitution method the form of the solution is guessed then the constants of the function \( f(n) \) is solved by mathematical induction. Recursion tree can be utilized in making a good guess in the substitution method.
6. Route calculation algorithms and their analysis

In this chapter a description of each algorithm is given and their order of growth is analysed by means of algorithm analysis. Then the result is be analysed and pondered over to find out the possible reasons for poor performance of the algorithm, if needed. The route-calculation algorithms may be considered to consist of several different sub-algorithms. Some parts of the algorithm may be executed multiple times in the actual execution of the software, making the analysis a bit more complex. (i Royo, 2010) Thus, each sub-algorithm is be analysed as one entity. Then the growth rates are compared to determine the biggest contributor to the performance. Those major contributors will also reveal the bottlenecks of the algorithms' performance.

6.1 Short-term calculation algorithm

The algorithm has been split into methods that each carry out a sub task, which is here called a sub-algorithm, and all of those sub-algorithms are analysed separately. Thus, detailed information can be acquired of the algorithm's behaviour and it also simplifies the analysis process. The order of growth of the whole algorithm is determined by the highest order of growth identified in the sub-algorithms. Those parts of the algorithms where the data intensity is small have intentionally been left out. The short-term algorithm assumes that the data is handed by the long-term algorithm. That data is a vector of points that indicate the planned route of the sailboat. It also requires the current boat position, wind angle and wind speed data. The algorithms were implemented in a previous research by i Royo (2010). Each step of the algorithm is marked with c(r), where r is a running number in the following analysis.

```java
void nearest_point = 0;
for (int i = 0; i < long_term_route.size(); i++) {
    double temp = getDistance(boatPosition, route.at(i));
    if (temp < distance) {
        distance = temp;
        nearest_point = i;
    }
}
return nearest_point;
```

**Algorithm 1.** Algorithm to find the smallest distance between the boat and the route.

The first sub-algorithm calculates the shortest distance between the boat and the long-term route. The input data for the algorithm is the long-term route as a vector of points and the current boat position on a chart. The whole input will be in the for loop, and the distance between the point on long-term route and boat position is calculated. The point on long-term route with smallest distance to the sailboat is returned. The highest term factor can be derived from the for-loop in the algorithm.

The algorithm's running time approximation is \( T = c_1 + c_2 + (c_3 + \ldots + c_7) \). The time complexity of this algorithm is \( T(n) = \Omega(n) \), where \( n \) is the number of points in the long-term route-vector. Now, the growth of this sub-algorithm is linear. As the size of the dominating input data (variable “route” that is a vector of points, and “boatPosition” is always a point) grows by size \( n \) the running time also grows by multiplier \( n \). The algorithm is thus quite inefficient with large route sizes. But on the other hand, the
typical route sizes are tens or hundreds of route points and hence the effect to the total calculation time is actually quite small. If the running time is to be enhanced, a logarithmic growth rate would be required to achieve. This could be acquired by introducing recursiveness to the sub-algorithm (Cormen et al., 2009).

The second algorithm calculates a special case for a long-term route of size of one. It checks if there are obstacles between the single path point and the boat and the destination point (the long-term route point). It checks from the chart database if there is an obstacle-free route available. The algorithm runs until an obstacle-free projection is found between the boat and the point on the long-term route (left side or right side). The algorithm stops when an obstacle-free projection is found or it determines that the distance from the original checkpoint (the nearest point on long-term route) to the point in the new heading gets too far (offset). This is determined in method “checkOffset”. The method calls “checkIntersection”, “lineToHalf” and “checkOffset” run in constant time. The algorithm returns the heading as an angle to a path with no obstacles.

```
while ( !ready ) {
    if ( ( obstacle_r && checkIntersection( "obstacles_r", heading ) ) ||
         ( obstacle_l && checkIntersection( "obstacles_l", heading ) ) ) {
        last_heading = heading;
        heading = lineToHalf( heading );
    } else {
        if ( checkOffset(heading, last_heading, offset ) )
            ready = true;
        else
            heading = averageLine( heading, last_heading );
    }
}
return heading;
```

Algorithm 2. Algorithm for finding the optimal obstacle-free projection between the boat and the destination checkpoint for a route size of one.

Remarkable in this algorithm is that the calculation time may vary even with exactly the same size of data if the coordinate or obstacle data varies between different execution runs. If there are no obstacles between the boat and the destination point the algorithm stops there. The algorithm will run the maximum time when there is not obstacle-free projection to any of the tested destination checkpoints (projections from sailboat in direction of the headings) and it stops when the “checkOffset”-method returns boolean value “true” indicating that the offset to the original destination checkpoint is too big. The while-loop is executed until either execution rule is met. Thus, the growth of this algorithm is limited from above. The algorithm's growth rate is acquired by summing up the steps during the iteration of the while loop: \( T = c_1 + (c_2) + (c_3) + (c_4) \) and \( T(n) = O(n) \).

The next algorithm handles the case where there is more than one point in the long-term route. It creates triangles bound by coordinate points between the boat and two adjacent points in the long term route, and uses database functionality to check if the formed triangle contains any obstacles. This routine is repeated until an obstacle is spotted or all the triangles between boat and long-term route have been inspected. If the algorithm finds an obstacle from a triangle, it starts to calculate the nearest possible evasion point of that obstacle. The input data is the long-term route as a vector of points and the boolean values from the database queries. As in algorithm 2, the method calls “checkIntersection”, “averageTriangle_onePointVersion” and “checkOffset_onePointVersion” run in constant time.
Algorithm 3. Algorithm for finding the optimal obstacle-free projection between the boat and the destination checkpoint for a route larger than one.

The outer while-loop checks if there are obstacles in the projection from the boat to a point in the long-term route. When the outer-loop encounters an obstacle (“checkIntersection”-method returns boolean value “true”) the execution is moved to the inner while-loop. The inner while-loop starts to finetune the destination checkpoint in such way that the encountered obstacle is avoided at safe distance. It runs until the offset from the original destination checkpoint (the nearest point on the long-term route) is determined to be too big and stops. The execution stops in the outer while-loop if the triangle is infinitely narrow.

The algorithm running time approximation is \( T = c_1 + \ldots + c_1*(c_5*(c_6 + \ldots + c_{10})) \). The growth rate of this algorithm is \( T(n) = n*n = n^2 \). As in Algorithm 2., the calculation time heavily depends on the data in the database.

The execution time of the algorithm is at its maximum when an obstacle is encountered at the last point of the long-term route. And correspondingly the case where the obstacle is found at the first formed triangle is the best case.

The following algorithm calculates the actual short-term route to the long-term route point where no obstacles were found between the sailboat and the long-term point. It calculates one side of the short-term route at a time. The input data for this algorithm is the current boat position, the calculated destination checkpoint from Algorithm 3. and the maximum allowed number of turning points for each side of the short-term route. The algorithm calculates the master turning points (for both sides of the short-term route, that is overkill) with method “getTurningPoints”, that are the bounding points on the map the algorithm may not cross. The “getTurningPoints”-method is runs-in constant time.

The algorithm starts forming triangles that are bounded by the boat position, the current turning point (initially the master turning point) and the destination checkpoint. The triangles are then checked for obstacles by method “checkIntersection”, that forms and executes a database query. If an obstacle was found inside the triangle, the triangle is splitted to half, and again it is checked if there are obstacles in that triangle. This procedure is iterated until an obstacle-free triangle is formed. The methods “triangleToHalf” and “avgTriangle” run in constant time.
The algorithm running time approximation is $T = c_1 + \ldots + c_1*(c_14*(c_15 + \ldots + c_22)) + (c_23 + \ldots + c_26)$. The time complexity of this algorithm is $T(n) = n^2$. The outer while-loop is limited by constant max_turns that limit the amount of gybes and tacks on a route. The outer while-loop in the algorithm always runs at least once or until the maximum allowed number of turning points has been calculated, thus the time-complexity of the algorithm is $T(n) = \Theta(n^2)$.

6.2 Summary of the short-term route calculation algorithms

The analysis of the algorithm's revealed the potential running time growth as the $n$ grows. The growth rate is either $n$ or $n^2$ that both are very good, and not much improvement to the running time can be achieved by introducing for example recursiveness. The order of growth of the whole algorithm is determined by the higher order terms $n$. No bottlenecks were revealed by algorithm analysis. Although the "getTurningPoints" calculates the master turning points for both left and right side of the short-term route where one side would suffice. Only the other side path is calculated at one go. It is suggested to remove the extra calculation from that method as overkill.

Without information on the actual data sizes for each algorithm it is difficult to decide whether the analysis reveal the bottlenecks or not. With measured data of the running times of the algorithms and the input data sizes it is possible to further analyse if bottlenecks exist in the algorithms.

6.3 Long-term algorithm

The long-term algorithm implementation is based upon a well tested and well known one source shortest path algorithm introduced by Dijkstra, 1959. The algorithm is greedy algorithm, because it always chooses the vertex with least weight: (Cormen et al., 2009) The algorithm's growth-rate is $O(n^2)$, and it is thus limited from above. (Cormen et al., 2001) The algorithm is based on graph theory where vertices are
connected by edges that have weight (in the UpWind case distance). It seeks to find a path where the total summed weight between the start and the end vertices is the smallest possible.

Algorithm 5. Long-term route algorithm using Dijkstra one source shortest path algorithm

Goldberg & Tarjan (1996) have analysed the algorithm in detail and have introduced three priority queue operations; insert, extract-min and decrease-key. These are steps in calculating the shortest path.

The input data for the algorithm matrix of edges and vertices that represents the map data and start (boat position) and end (the destination) points for the calculation. The algorithm running time approximation is \( T = c_1 + c_1...c_8 + c_9*(c_{10} + ... + c_{15}) + c_{15} + c_{16}*(c_{17} + ... + c_{20}) + c_{21} + c_{22}*(C_{20} + ... + C_{24}) + c_{25} \). The time complexity of this algorithm is \( T(n) = n + n^2 + n = n^2 + n \), thus \( T(n) = O(n^2) \) as confirmed by Cormen et al. (2001). The algorithm is bounded from below.

Two different version of the algorithm have been introduced in the literature. Furthermore these two versions may use different kinds of heaps as source data, usually binary or Fibonacci-heaps. (Chen et al., 2007) First version supports the Decrease-Key operation and the other supports Insert and Delete-Min operations. Decrease-key operation searches the point from a set that has the minimum distance value of the vertex, inserts new vertices into the queue, removes the smallest distance item and finally decreases distance value of a vertex and adds the value to the path variable.
The implementation in UpWind is based on binary-heaps: each vertex may have two or more branching edges.

Which is faster solution, binary-heap or Fibonacci-heap solution? According to Cormen et al. (2009) binary-heap solution is faster for sparse graphs. Graph is considered sparse enough for binary-heap implementation when:

\[ E = o(V^2/\log V) \]

where \( E \) is the number of edges and \( V \) is the number of vertices.

In the Oulu region chart, the amount of vertices and edges is exactly the same, 452. Thus the graph is considered sparse, and binary-heap would be theoretically the fastest Dijkstra algorithm implementation.

The second implementation, Insert and Delete-Min-operation on the other hand scans the distance array on every iteration in order to find the closest vertex, and does not perform decrease-key operation. In the literature, first implementation is considered to be slower. Goldberg & Tarjan (1996) suggest that Decrease-key method is not faster in realistic heap sizes, making the Fibonacci-heap implementation not faster than the binary-heap implementation. The reason for this is that in the decrease-key implementation the amount of the decrease-key operations is much higher than extract-min operations. Chen et al. (2007) support this view. Chen et al. (2007) made a comparative research on the different implementations on Dijkstra single source shortest path algorithm and measured the running times of the implementations. They came into a conclusion that the implementation without decrease-key operation is faster in most cases, although the algorithm with decrease-key operation performs better in sparse graphs. The implementation in the UpWind program follows the first version of the algorithm (with binary heaps).

If the measurements indicate poor performance especially with large heaps, it should be taken into consideration to modify the algorithm by replacing it with the latter described version. Or replacing it with a totally different algorithm like Gabow's scaling algorithm (Gabow, 2000).

6.4 Queuing behaviour of the algorithms

The short-term algorithm is currently started at 10-second intervals. If the algorithm runs faster than that 10-second interval and no queuing behaviour exists, then all execution times are run in due time. But if the execution takes more than that interval, queuing behaviour emerges. This does not advocate usage of the performance evaluation methods, though. The reason is simply the fact that longer running times results from bottlenecks in the algorithm' performance. The algorithms are designed to be run as fast as possible and no sign of intended queuing behaviour can be found from the algorithms because there are no waiting states nor any handlers for queues.

A calculation time that is longer than the current interval may lead to very serious consequences. As the captain of the sailboat does not receive information on the short-term route on time, she or he might make a serious maneuvering mistake that might lead to an accident.

Thus, usage of those performance engineering methods would lead to faulty analysis and results and cannot be combined to reveal the real performance of the algorithm. Some of the metrics could be utilised in the analysis, like the calculation of throughput,
but those metrics would be taken out of their context and the results would be more or less useless.

Measurements and statistical analysis can be used in combination with the algorithm analysis. The results from the algorithm analysis should be confirmed with the results from algorithm analysis with measurements. The question is how well the acquired growth-rates match with real measured data. If the results from the measurements show that all calculations run within the given interval, it should be estimated how big data amounts the algorithm can handle within the interval.
7. Measurement of the algorithms' performance

To predict the calculation times of the algorithm the current performance of the algorithms needs to be obtained. In order to obtain the performance in some hardware setup, measurements are needed. Although both Sedgewick & Flajolet (2013) and Cormen et al. (2006) argue that analysis is sufficient for obtaining the performance, in this context it is not enough. The algorithm analysis reveal the order of growth, but it does not reveal the underlying bottlenecks like I/O actions. In this chapter the requirements for the measurements are defined, the measurement method developed and a description of the implementation of the measurement method is given. In the end of this chapter the results are analysed and the bottlenecks in the calculation are detected. The obtained results will be further used in the analysis of the calculation times of the algorithms.

The purpose for the measurements is to reveal the calculation times of each algorithm. Some method to measure the individual calculation times of the algorithms is thus needed. The implementation of both the long-term algorithm and short-term algorithm are divided into functional entities that perform a single comprehensive task that are here called sub-algorithms. This division of the whole algorithm into sub-algorithms enables easy access to more detailed information of the atomic performance of the sub-algorithms.

7.1 Requirements for the measurements

The calculation time measurements have to be valid and reliable and thus requirements for the measurements have to exist. The measurements are dynamically: the software is running during the measurements (Smith & Williams, 2002).

Requirements:

1. The implementation has to be designed in such a way that the measurement does not have any influence on the results.

2. The measurements only measure the phenomena of interest.

3. The measurement of the individual sub-algorithms must not interfere or have any influence on the measurement results of other sub-algorithms.

4. The measurements have to be repeatable; the results of the subsequent measurements have to be equal with some defined tolerance. In other words the variance of the measurement results should be below a limit. This also provides the reliability of the measurements.

5. The object of the measurement is homogeneous, it does not change during the measurements or between the measurements.

6. The environment in which the measurements take place should not variate during the measurements. This means that the hardware setup should not
change between the measurements, the software environment should not change
and the stress level should not change.

7. The requirement six gives some further good advice for setting up the
environment: always run only the absolutely necessary application for the
measurements and that the environment should be monitored by some means.

8. The measures have to fit to the measured phenomena.

7.2 Description of the Measurement method

The measurements must reveal the absolute calculation time for both the sub-algorithms
and the whole algorithm. Intuitive method is to use timers. (Smith & Williams, 2002) In
Qt-framework there is a QElapsedTimer-class that provide access to the computer
reference clock and provides time scale of nanoseconds. QElapsedTimer provides a
nSecsElapsed-method that returns the running time of timer since the last call of that
timer-object in nanoseconds.

The measurement method is direct measurement. The running time of the algorithms
and repetition of calculations will be measured when plausible. The running time
measurements are inserted into the methods that implement the algorithms. The
measurements are implemented in a such fashion that their impact to the measured
values are at least minimal, preferably none. An algorithm is allowed to end and
immediately after that the consumed running time value will be recorded.

The cumulative running time of the algorithms is calculated outside the algorithms. The
impact of this implementation to the measured values cannot be known; the time
resources for each individual timer cannot be measured. The expected impact of using
timers to the measurement results is not significant. The results will be analysed by
means of quantitative research methods that will reveal any deviation in the results.

With running time and repetition of calculations it is possible to determine the running
time per calculation entity or step in an algorithm. By analysing these single calculation
entities it can be determined what or which parts of the algorithms have the biggest
running times and which are possible bottlenecks.

How to select which running times to measure? The two algorithms are entities whose
implementation consists of several methods. Furthermore, several different methods
constitute one logical unit that has to be measured as a whole. The goal of the
measurements is to reveal the underlying running times and and the algorithm's total
running time. Thus, it is logical to measure the running times of the different methods
and the whole running time of the algorithm.

7.2.1 Short-term measurement description

The measurements are conducted while the program runs. The time values are captured
by introducing a timer to each function before the code of the algorithm starts. After the
algorithm code ends the running time is recorded by writing it into a log file. The
overall running time is acquired by summing the individual running times of the
different parts of the algorithm.

An important aspect that has to be taken into account is the fact that the chart data used
in the calculation of the short-term algorithm exist in a PostgreSQL-database located on
a separate server. The consumed time of the database queries have to be recorded. It is assumed that a situation, in which an obstacle is found at the farthest edge of the checked major triangle, and which consists of the current sailboat position and the two points in the long-term route, will cause the biggest number of database queries and thus the longest running time for the algorithm.

The measurements are conducted so that the sailboat is maneuvered on the chart, and the route data is then recorded and stored into a text file. The route data is in form of NMEA. During the measurements the sailboat is forced to follow the recorded route. Two different routes were recorded both starting from the same point and ending in different places. The chosen routes are deliberately difficult in order to stress the behaviour of the algorithm in both best-case and worst-case scenarios.

The best-case scenario has not any obstacles in the triangle formed by two points in the long-term route and the sailboat position. In the worst-case scenario the number of obstacles between the sailboat and the long-term route is so big, that it is not possible to calculate the short-term route. It is expected that the worst-case occurs in Algorithm 3., where an obstacle-free projection between the boat and long-term route is searched. In the worst-case situation the outer while-loop has to iterate through all the points in the long-term route vector. The inner while-loop forms new triangles until the triangle becomes so small that it is considered a line, and no obstacle-free projection can be found.

The maximum number of turns allowed for the path on each side of the short-term route is 5 during the measurements. As shown in Algorithm 4., the growth-rate of the while-loop checking whether the maximum amount of turning points has been reached is quadratic. Hence, the effect of the maximum number of the turning points will not be measured by growing that value. It can be seen that the nested loops have a much more significant effect on the calculation time with the complexity $n^2$.

### 7.2.2 Long-term measurement description

The long-term route is calculated when user clicks the screen. Thus, the intuitive way to test the performance is to click on different locations on the user interface to generate routes of different length and then measure the running time of the algorithm. The Oulu chart area channel system consists of 456 vertices and 456 edges.

Dijkstra algorithm is well tested and a lot of research has been conducted around it. (Goldberg & Tarjan, 1996) The implemented algorithm in the Upwind application is situated in a single method, and it cannot be split into coherent sub-algorithms without disturbing it's current time-complexity. Hence, it will be measured as one entity. The running time and the resulting path size will be measured. These values can be acquired without disturbing the running time of the algorithm.

### 7.3 Hardware and software setup

The measurements are conducted on an Lenovo T61p laptop with 8 gigabytes of RAM and and (→ toisto) Ocz Vertex 3 SSD hard disk. The processor is Intel Core2 Duo 2.2GHz. Operating system is Linux, Ubuntu 12.04 LTS operating system. The version of Qt Creator is 4.8.0 (32 bit). The version of Virtual box is 4.1.12_ubuntu. The load of the computer was monitored with Linux command line application “Top” that reports the usage of resources by the different processes and the total usage of resources. The
load of course changed while measuring but most importantly no shortage of resources was observed.

7.4 Obstacle data

The obstacle data is written into the PostgreSQL database in the startup of the application. The coordinates of the obstacles are taken from the database and inserted into two new tables: “obstacles_r” and “obstacles_l”. Obstacle data include objects of different shapes: points, lines and polygons. The obstacles may be objects like rocks, wrecks, navigation aids and depths less than 2 meters.

The obstacle checking during short-term route calculation is queried from the database by PostGIS library function 'intersects'. A SQL “select” clause is constructed at runtime taking into consideration what kind of object (shape) is tested for obstacle intersections in the query. The query returns true if one or more obstacles were found in the tested object or false otherwise.
8. Measurement results and their analysis

8.1 Short-term algorithm

The population of the measurements totaled altogether 459 separate results on two different routes. Both routes started from in between Riutunkari harbour in Oulunsalo and Huikku harbour in Hailuoto (illustrated in Picture 3., travelled route drawn with black line) and ended to Haukipudas by Hietakari island and near Hietaniemi in Hailuoto (illustrated in Picture 4.). The algorithm was measured as a logically coherent set of methods.

Picture 3. Route from Oulunsalo to the vicinity of Hietakari, Haukipudas
The analysis method depends on the behaviour of the algorithm at run time. If queuing behaviour is detected then queuing theory principles can be combined with metrics and utilised in the analysis. However there is no queuing behaviour except in the worst-case scenario.

<table>
<thead>
<tr>
<th>Name of the measured entity</th>
<th>Average consumed time (ms)</th>
<th>Average percentage of total consumed time</th>
<th>Average percentage of the consumed time by queries (%)</th>
<th>Best-case consumed time by database queries (%)</th>
<th>Worst-case consumed time by database queries (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetNearest Point</td>
<td>0.013</td>
<td>0.008</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>GetNext Point</td>
<td>685.65</td>
<td>54.68</td>
<td>69.48</td>
<td>20.08</td>
<td>99.79</td>
</tr>
<tr>
<td>PopulatePolarDiagram</td>
<td>0.007</td>
<td>0.002</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>GetPath (left side)</td>
<td>257.84</td>
<td>10.03</td>
<td>68.20</td>
<td>0</td>
<td>99.82</td>
</tr>
<tr>
<td>GetPath (right side)</td>
<td>268.46</td>
<td>9.75</td>
<td>65.28</td>
<td>0</td>
<td>99.85</td>
</tr>
<tr>
<td>Other</td>
<td>39.68</td>
<td>25.53</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1. The average usage percentages of the total used time of the set of methods.
Table 1 shows the average, best case and worst case values of the different sub-algorithms. The set of methods are named after the purpose they serve in the calculation. As can be seen from table 1, getting the nearest point on the long-term route is fast. The long-term route data is obtained from the class that performs it’s calculations as a vector of points, the route data is stored in RAM. The memory access and making the projections from sailboat to each point of the long-term route is fast, as Graph 1. shows. The maximum time consumed on getting the nearest point was 0.18 ms with of 0.0011 percent of used time on short-term route calculation, median being 0.04 ms shown in table 2. The variance is small, almost zero indicating nearly non-existent variance in the result set. In average this part of the algorithm consumes 0.0075 % of the total time used. The time used is negligible compared to the total consumed time.

<table>
<thead>
<tr>
<th>Name of the measured entity</th>
<th>Variance of calculation time</th>
<th>Median of calculation time (ms)</th>
<th>Standard deviation of the calculation time (ms)</th>
<th>Mode of the calculation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetNearestPoint</td>
<td>0.00012</td>
<td>0.04</td>
<td>0.011</td>
<td>0.041</td>
</tr>
<tr>
<td>GetNextPoint</td>
<td>1564.09</td>
<td>20.25</td>
<td>39.54</td>
<td>0</td>
</tr>
<tr>
<td>PopulatePolarDiagram</td>
<td>0.0000025</td>
<td>0.0067</td>
<td>0.0016</td>
<td>0.007</td>
</tr>
<tr>
<td>GetPath (left side)</td>
<td>4332.96</td>
<td>18.46</td>
<td>58.59</td>
<td>0</td>
</tr>
<tr>
<td>GetPath (right side)</td>
<td>2117.50</td>
<td>14.95</td>
<td>46.02</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Statistical attributes of the time consumed by the sub-algorithms.

“GetNextPoint”-method finds the destination checkpoint on long-term route where the short-term route calculated with already calculated nearest point as the starting point. It runs until an obstacle-free line-of-sight is found between the sailboat and the calculated point. It fine-tunes the destination checkpoint until it finds an obstacle-free line of sight between the sailboat and the destination checkpoint. This method consumes majority of the used time in the short-term route calculation.

Graph 2. shows the percentage of the total calculation time the method consumes. It can be seen from the Table 1 that in average it consumes 55.68 percent of total calculation time. Thus, this method is the biggest bottleneck in the short-term route calculation. In the best-case situation the time consumed by database queries is 79.71 percent less than in the worst-case situation. This confirms that data-base queries heavily dominate the consumed running time. Variance, shown in Table 2., between the measured time for the calculation is quite big, 1564.09, indicating the varying situations on the chart; the amount of obstacles between sailboat and destination checkpoint determines the number of intersection checks and the used time.
Graph 1. The percentage of the total (from all sub-algorithms) time used by the sub-algorithm which calculates the nearest point on the long-term route to the sailboat.

Graph 2. Time consumed by the algorithm calculating the destination checkpoint of the total consumed time.

The algorithm constructs a triangle with one point on sailboat and two points on the long-term route. The running time depends on how many times the algorithm performs the triangle construction and the check to find out if there are obstacles in the projection between the sailboat and the currently reviewed point on long term route.
The checks for the obstacles are performed with database queries. Graph 3. shows the percentage of the consumed time. In the worst cases these queries very heavily dominate the time used. In average 86.48 % of the time is consumed by the queries. In average one intersection check lasts 40.86 ms.

**Graph 3.** The percentage of time used in database queries when calculating the next destination checkpoint.

**Graph 4.** Number of database queries in each measurement in sub-algorithm calculating the next destination checkpoint.

Table 1 shows that most of the consumed time is consumed by the intersection queries from the database, in average 69.48 %. In the found worst-case situation 99.85 % of the time is consumed by SQL queries.
Graph 5. Percentage of database queries in each measurement for the right side short-term route in the method GetPath.

Graph 5. shows the on/off nature of the time consumption when calculating the short-term route. When there are no obstacles found in the area formed by the sailboat, destination position and the turning points for the short-term route, the “getPath”-method is not called and thus the amount of SQL queries is minimal, in the best case only two queries are needed to find out that there is no obstacles inside the formed rhomboid. The results of calculating the left path are almost similar to calculating the right path; the calculation is done with the same method, and the main difference in consumed time results from the difference in the number of calculated turns in the path.

The maximum time consumed by the short-term route calculation is 20.6 seconds and, which is more than twice the 10 second interval after which the calculation is triggered. This leads to queuing behaviour by the algorithm. The next calculation is delayed by twenty seconds.

Graph 7. shows the portion the database queries take in each measurement. The mode of the database query percentage is 0. This has a significant effect on the average value. If the zero values are removed from the average, the average percentage value is 99.12. Thus, anytime there are database queries, the queries start to heavily dominate the proportion of the consumed time as shown in Graph 6.; almost half of the calculation time is consumed by the database queries. This indicates that the effect of the other functionalities like the construction of the triangles are insignificant to the time consumption. Graph 6. shows the growth of running time of the whole short-term route algorithm as a function of number of database queries. As was found out in Chapter 6.1 in the sub-algorithms where the obstacle checks occur, that is, in methods “GetPath” and “GetNextPoint”, the growth-rate of the running time is \( n^2 \). The trend curve shown in Graph 6. quite closely follows that quadratic order of growth.
**Graph 6.** Growth of the total calculation in function of number of database queries.

**Graph 7.** Describes the portion of database queries of the total used calculation time.
8.2 Long-term algorithm

The measurements consists of 34 separate measurements of the algorithm. Graph 8. illustrates the calculation time for the route. Graph 9. shows the number of points in the resulted route. The average calculation time is 8.51 ms with median 7.86 ms. The maximum calculation time was 16.46 ms where route length was 50 points. The correlation between the calculation time and the resulted route size is 0.31 indicating a small dependence.

As shown in Graphs 8. and 11. the calculation time is quite constant throughout the whole range of measurements, except for the measurement 7 where the resulted route size was one. This indicates that the time was not mostly consumed by the algorithm itself, but by the initialization of the method. When the resulting route grows, the proportional time per route point grows shorter. Graph 10. shows that the number of resulted path points has little or no effect at all to the running time of the algorithm. This confirms that the running time depends solely on the matrix size.

Long-term route calculation runs in the same thread as the user interface. Thus it is mandatory that the algorithm runs faster than the user interface frame rate. Typically the frame rate in a modern LCD display is 25 frames per second, resulting 40 ms interval between the frames. The interval between frames is sufficient for the algorithm to run without blocking the user interface when using Oulu water area chart.

The running time can roughly double without blocking the user interface. With quadratic order of growth of the algorithm, as proved in Chapter 6.2., the maximum matrix size (of the long-term input data) is 703, where the estimated calculation time is 39.91 ms. With larger matrices user interface starts getting blocked by the calculation. To improve the performance of the user interface, the algorithm should be moved to its own thread. It would also be beneficial to indicate the user of the ongoing route calculation with very large maps. For example, calculating through matrix of size 10000 will take 8.076 seconds in the tested environment.

Graph 8. The running time of the implemented Dijkstra algorithm on each measurement.
**Graph 9.** The resulted route size on each measurement.

**Graph 10.** The time consumed by the Dijkstra algorithm in the function of resulted route size.
Graph 11. Calculation time as a function of resulted route size indicating that the calculation speed grows per path point as the route size grows.
9. Performance enhancements

The long-term algorithm is based on the well and widely tested single source shortest path algorithm by Dijkstra. (Goldberg, 1996) The algorithm analysis showed that the time-complexity of the algorithm is relatively low ($n^2$). It is strongly suggested that the algorithm should be moved to its own thread to avoid user interface blockage, or otherwise indicate to the user that the route calculation is ongoing. Currently the algorithm runs in the same thread as the user interface. The algorithm runs sufficiently fast in the used test environment with 2.43 times bigger maps as the Oulu region map used in performance testing. However, performance blocking phenomena may occur with slower machines and bigger map sizes.

The performance problems of the short-term algorithm are more severe. The measurement results and the analysis showed that the actual short-term calculation algorithm is sufficiently fast as it is. The performance blocking phenomena are the recurring database transactions. Thus, the focus of the enhancements has to be in reducing those transactions. The other performance bottlenecks should be searched again after the current problems are fixed and new performance level is assessed with the requirements.

The whole obstacle data for Oulu region chart contains altogether 700064 rows of data and requires 100.15 mega bytes of memory. This could be loaded into RAM without loss of performance in modern machines. However, with big charts, for example the whole chart data of the Finnish sea area, this kind of approach might consume a major portion of the RAM and thus cause severe performance problems. Furthermore iterating through long arrays may require too much time. In this chapter some ideas are proposed on how to reduce the time consumed by the database queries.

9.1 Trapezoid of obstacles

The short-term route is calculated inside a trapezoid formed by the destination checkpoint, the master turning points on both sides and the sailboat position. ( Royo, 2010) This formed trapezoid may or may not have obstacles inside it. The obstacle data should be read only when the destination checkpoint changes. Picture 5. illustrates the area of interest.

The obstacle data of this formed trapezoid can be queried from the database relatively fast. In this proposition the data is read each time before the short-term route calculation begins.

The average time spent in the intersection queries from the database is in the original solution 1243.87 ms and the median value 641.16 ms. The retrieval time was measured with the same routes as the measurements were conducted in chapter 8.1. The average retrieve time for the above described trapezoid of obstacles is 180.16 ms, while the median time is 56.99 ms. In average the save in database queries is 1063.71 ms, the used time is only 14.5% of the original average retrieval time.
The average time in iterating through the whole read trapezoid of obstacles takes in average 384.14 ms with average data size of 202 read obstacles. The maximum amount of read obstacles was 1794 with read time 724.15 ms and the iteration time 2260.09 ms.

Although the time for checking the intersection is significantly lower the actual time saved cannot be predicted prior to the implementation of the method checking the intersection from the data read from the database. Anyhow the savings in calculation time are expected to be significant.

The intersection check in the current solution is conducted by the 'intersects' method of the PostGIS add-on. In this proposition new method(s) for checking the intersection has to be implemented to polygonal, line and point geometries.

If the queries to the tables obstacles_l and obstacles_r return empty result set, no obstacles were found and an optimal short-term route can be calculated immediately. Otherwise the intersections have to be checked.

9.2 One-sided short-term route

At the moment the short-term route is calculated on both sides of the sailboat, starboard and port. However no captain can follow both lines at the same time. Thus, it suffices to calculate only one side depending on which of the short-term route's sides the captain decides to follow. This should roughly halve the short-term route calculation time, depending on the obstacles.

Of course initially, after the long term route change, both sides need to be calculated. As the boat moves, the distance between the selected short-term route and the boat is calculated, and it is determined which of the two given routes is closer to the boat. After that only the closer route needs to be recalculated until the recalculation conditions are met.

9.2.1 Preliminary conditions and method

A method for finding out whether the sailboat follows the short-term route or not needs to be developed. An agreed projected distance between the boat and the route should be
set as a boundary limit. If the distance remains within the limit it can be said that the boat is actually following the route. If not, both sides of the short-term route need to be calculated.

The wind direction together with the sailboat's polar data governs the calculation of the turning points of the route (i Royo, 2010). How to determine when the wind direction changes enough to trigger the calculation? The polar data gives a chart of the angles and the theoretical speeds for a given wind direction. A simple check in which the polar data is compared with the current wind direction in case where the sailboat is beating with the current wind direction could solve this problem.

This information can be used to decide whether the calculation of the short-term route should be done or not. If the angle changes so much that the sailboat is beating against the wind, the short-term route is recalculated. Otherwise the boat is reaching and no calculation is necessarily needed. This proposal requires implementation of new functionality into the PolarDiagram-class of the application. The new functionality should be able to determine with the aid of current true wind angle and wind speed if the sailboat is beating or running. Of course one can argue that the optimal short-term route should always be at hand. However, it should be considered whether the short-term route should change every ten seconds or if it should change at all in case the wind conditions do not change. As it will change for each recalculation as the sailboat moves on the sea.

Let's assume a situation where the short-term route has been calculated and sailboat's captain has decided to follow either one. If the captain seems to be following the selected short-term route, there is no reason to calculate the route again until reaching the destination checkpoint or unless the wind conditions change dramatically. Of course, if the captain sails out of the boundary limits the both short-term routes have to be calculated again. A threshold distance should be set in which the routes should be calculated again or alternatively the calculation is done at the destination checkpoint. The check whether the routes should be calculated again or not should be done in fixed intervals, for example every time the NMEA data is received.

As an example the boundary limit from the original selected short-term route is 50 meters and the wind direction change threshold for recalculation is 10 degrees. Picture 6. illustrates this method. Initially the sailboat follows the selected route. The black line represents the trapezoid of obstacles and the blue line represents the boundary limit to the selected route. Here the boundary limit is drawn around the route to the distance of 50 meters perpendicular to the route. If the sailboat moves outside the boundaries, the recalculation is triggered. In Picture 6. wind direction has changed 37 degrees resulting recalculation of the both routes (left and right).
Picture 6. Illustrating the boundary limit and wind direction change.
10. Discussion

The main research question of this research is to assess the suitability of the current popular software performance engineering methods to be utilised in analysing the two route calculation algorithms. The research of the current software performance engineering methods is based on the queuing theory and involves the application of the complex probability theory, including Petri-nets, Queueing network theory and software performance metrics. UML Performance Modeling is a tool for high level performance analysis especially in the early stage of the software development process. These methodologies turned out to be too complex and cumbersome for the case under study, and they lack the representative power to describe sub-algorithm level details. The analysis of the algorithms revealed that they do not have any queuing behaviour, if the algorithms are considered to be applied as intended. The inevitable conclusion is that the software performance engineering methodologies studied previously cannot be adapted to implement performance analysis of the route calculation algorithms.

Since the current literature on software performance engineering fails to provide methodologies for the assessment of low level algorithms, some other means had to be come up with. Algorithm analysis together with statistical analysis of the runtime behaviour of the algorithms and sub-algorithms can be used to analyse the performance of the algorithm in both algorithm and sub-algorithm level. The main goal of this research is to find out the possible bottlenecks of the two algorithms, and provide solutions to the found performance problems. Algorithm analysis reveals the major bottlenecks where as the statistical analysis of the measurement results reveals the more granular performance data. The two methodologies support and complement one another, thus proving the decision for the analysis methods correct. The algorithm analysis did not reveal any bottlenecks but proved the algorithms' low time-complexity.

The measurements of the algorithms are base on the algorithm performance concept called time-complexity and the common principles of measurement uncertainty. The algorithm and sub-algorithm level running times were measured and saved for statistical analysis. Statistical, comparative analysis revealed the relationship between the performance of the different algorithms. Algorithm analysis and statistical analysis of the measurement results constitute the methodology to assess the algorithms, and hence that methodology is the solution to the research question.

The software components of the short-term route-calculation algorithm were not measured as a whole, as they consist of many independent parts which had to be measured separately in order to reveal the granular run-time problems. The algorithm was divided into pieces as it was implemented in the source code: on method level. On the other had, it was necessary to measure the long-term algorithm as one entity since it has been implemented as one method, and dividing into sub-algorithms and measuring their run-time properties with timers would have disturbed the run-time behaviour of the algorithm.

The performance requirements were discussed briefly. Only one concrete time requirement relevant to the short-term algorithm emerged: it has to complete within the given ten-second time frame. The long-term algorithm is running at the same thread as the user interface and has to complete within the time-frame of display update to avoid the blockage of the user interface.
The short-term route calculation algorithm is a tailor made solution whereas the long-term algorithm is based on the Dijkstra single source shortest path algorithm. The long term algorithm is sufficiently fast and it is not blocking the user interface with the tested Oulu region map. With bigger maps and slower machines performance blocking phenomena may occur, and it is suggested that the algorithm is moved to a separate thread from user interface thread or otherwise indicate to user that the route calculation is ongoing. The statistical analysis of the running time measurements revealed severe problems in the running time of the short-term algorithm. The current implementation of the short-term route calculation algorithm does not meet the ten-second limit in which the calculation has to complete.

The sole contributor to bad performance are the recurring database queries which in worst-case situations consume almost all of the time used in the calculations. These queries take place in two methods: “GetNextPoint” and “GetPath”. The performance can be enhanced by reducing the number of database queries and optimizing the algorithm by introducing new logic. Without the implementation of those suggested improvements to the algorithm at hand, the expected performance cannot be measured or predicted precisely. This and the corresponding research question number eight will be left for future research. Anyhow, the performance is expected to improve n-fold.
11. Conclusions

Algorithms in software are low level components in software entity. Thus, the performance engineering methodologies do not apply on algorithm level, limiting the availability of analysis methods in the case of route calculation algorithms. The research showed that algorithm analysis can be successfully combined with the statistical analysis of an algorithm running time measurements. Although the both methodologies are executed separately the results can be used in analysis and for ascertaining validity of the result mutually.

The performance of the route calculation algorithms were analysed in detail. The algorithms' time-complexity is sufficiently low, although some bottleneck's were found from the short-term algorithm's running time measurement result analysis. A solution frame to improve the short-term calculation algorithm was proposed. That frame consists of the proposal to modify the logic triggering the calculation as well as reducing the amount of recurring calculations and database queries. This solution should be designed, implemented and assessed in the future research.
References


