

Handling and Analyzing Marine Traffic Data

Master of Science Thesis in the Programme Computer Science – algorithms, languages and logic

ERIC AHLBERG, JOAKIM DANIELSSON

Handling and Analyzing Marine Traffic Data
ERIC AHLBERG, JOAKIM DANIELSSON

© ERIC AHLBERG, JOAKIM DANIELSSON, 2016.

Supervisor: Luis Felipe Sánchez Heres, Department of Shipping and Marine Technology

Examiner: Graham Kemp, Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: "APL Post-Panamax container ships" by National Oceanic & Atmospheric Administration from the California Publication of the National Oceanic & Atmospheric Administration, used under Public Domain / Converted to ASCII using <http://picascii.com>

Typeset in L^AT_EX
Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Handling and Analyzing Marine Traffic Data
ERIC AHLBERG, JOAKIM DANIELSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

With the emergence of the Automatic Identification System (AIS), the ability to track and analyze vessel behaviour within the marine domain was introduced. Nowadays, the ubiquitous availability of huge amounts of data presents challenges for systems aimed at using AIS data for analysis purposes regarding computability and how to extract valuable information from the data. This thesis covers the process of developing a system capable of performing AIS data analytics using state of the art Big data technologies, supporting key features from a system called Marine Traffic Analyzer 3. The results show that the developed system has improved performance, supports larger files and is accessible by more users at the same time. Another problem with AIS is that since the technology was initially constructed for collision avoidance-purposes, there is no solid mechanism for data validation. This introduces several issues, among them is what is called identity fraud, that is when a vessel impersonates another vessel for various malicious purposes. This thesis explores the possibility of detecting identity fraud by using clustering techniques for extracting voyages of vessels using movement patterns and presents a prototype algorithm for doing so. The results concerning the validation show some merits, but also exposes weaknesses such as time consuming tuning of parameters.

Keywords: AIS, Marine Traffic, Big data, Algorithms.

Acknowledgements

We would like to express our gratitude towards Luis Felipe Sánchez Heres for sparking the initial idea, for engaging in fruitful discussions and for providing swift and valuable feedback throughout the whole thesis. We would also like to thank Fredrik Olindersson for sharing his expertise in the marine domain.

Eric Ahlberg, Joakim Danielsson, Gothenburg, May 2016

Contents

| | |
|--|-------------|
| Contents | vii |
| List of Figures | ix |
| List of Tables | xi |
| List of Algorithms | xiii |
| 1 Introduction | 1 |
| 1.1 Background | 1 |
| 1.2 Related work | 2 |
| 1.3 Aim | 3 |
| 1.4 Limitations | 3 |
| 2 Theoretical background | 5 |
| 2.1 State of the art for Big data: processing and storage | 5 |
| 2.1.1 Computing abstractions | 5 |
| 2.1.2 Locality-sensitive hashing | 7 |
| 2.1.3 Z-order curve | 7 |
| 2.1.4 Distributed file system | 8 |
| 2.2 Cluster analysis | 8 |
| 2.2.1 Generalized Density Based Spatial Clustering of Applica- tions with Noise | 9 |
| 2.3 AIS data description | 10 |
| 2.4 Ship handling terms | 10 |
| 3 Method | 13 |
| 3.1 Handling: Development of a large scale data analysis system | 13 |
| 3.1.1 Requirements and admissible solutions | 13 |
| 3.1.2 Technique survey | 15 |
| 3.1.3 Integration | 15 |

| | | |
|----------|---|-----------|
| 3.1.4 | Evaluation | 15 |
| 3.2 | Analysing: Development of initial applications for the system | 16 |
| 3.2.1 | MTA3 Parallel | 16 |
| 3.2.2 | AIS message validation | 17 |
| 4 | Results and discussion | 19 |
| 4.1 | Handling: Development of a large scale data analysis system | 19 |
| 4.1.1 | Requirements | 19 |
| 4.1.2 | Technique survey | 20 |
| 4.1.3 | Integration | 22 |
| 4.2 | Analysing: Development of initial applications for the system | 29 |
| 4.2.1 | MTA3 Parallel | 29 |
| 4.2.2 | AIS message validation | 35 |
| 4.3 | Ethical considerations | 44 |
| 5 | Conclusions | 47 |
| 5.1 | Development of a large scale data analysis system | 47 |
| 5.2 | AIS message validation | 47 |
| 6 | Future work | 49 |
| 6.1 | MTA3 Parallel | 49 |
| 6.2 | AIS message validation | 50 |
| | References | 51 |
| A | Implementation of algorithms | I |
| A.1 | Ship handling algorithms | I |
| A.1.1 | Risk of collision | I |
| A.1.2 | Closest point of approach and time to closest point of approach | I |
| A.1.3 | Risk of collision (parallelized) | II |
| A.1.4 | Risk of collision (parallelized and optimized) | II |
| A.2 | Temporal hashing | III |
| A.2.1 | Contributions to the timehash [26] package | III |
| A.3 | Clustering | IV |
| A.3.1 | GDBSCAN | IV |
| A.3.2 | Post processing algorithm for validating a cluster | VII |

List of Figures

| | | |
|-----|--|----|
| 2.1 | Simple case of a Z-order curve, by interleaving the binary representation of the axis, each square represents a distinct combination. . . | 7 |
| 2.2 | Example of a Z-order curve applied to a map, where the latitude-longitude values are mapped from two dimensions to one dimension to allow for hashing of two-dimensional values. | 8 |
| 4.1 | Overview of the interface which allows the user to select a specific data set and interactively choose a region and filter out vessels according to time period, minimum speed and vessel type. | 24 |
| 4.2 | Overview of the interface where risk of collision calculations are started. The input fields allow the user to specify which of the situations that are of interest. | 25 |
| 4.3 | Overview of the interface which displays information about running and finished jobs as well as the results from each job. | 25 |
| 4.4 | Overview of the interface where the user can choose a data set and plot selected vessel paths according to identification number (MMSI) and time period. Each data point is clickable and provides detailed information. | 26 |
| 4.5 | Overview of the interface for handling data sets, where the user is able to upload data sets to the distributed file system. | 27 |
| 4.6 | A speed comparison of MTA3 and two versions of MTA3 Parallel, with and without sampling. As can be seen, MTA3 shows an early trend of rapidly increasing runtimes and lacks support for files larger than 54 MB. | 34 |
| 4.7 | A speed comparison of MTA3 and two versions of MTA3 Parallel, with and without sampling, only concerning the file sizes with which MTA3 is compatible. | 35 |
| 4.8 | Clustering of two vessels travelling in opposite directions (head on) using dummy data. | 39 |
| 4.9 | Clustering of three vessels using a predicate for capturing position and time. | 40 |

| | | |
|------|---|----|
| 4.10 | Result from the post processing algorithm applied to Figure 4.9b where the voyage in the middle contains two vessels with different identification numbers, thus resulting in an identity fraud. Vessels with different identity numbers are marked with distinct colors and markers. | 41 |
| 4.11 | Geographical representation of AIS data from a fishing boat travelling similar voyages, but during different time periods. | 42 |
| 4.12 | Clustered AIS data from a fishing boat travelling similar voyages, but during different time periods. The data is correctly clustered into several voyages using the position, time and course. | 42 |
| 4.13 | The issue of clustering data with irregularities in the transmission rates, where a voyage is incorrectly clustered into three voyages. . . | 44 |

List of Tables

| | | |
|-----|--|----|
| 4.1 | Hardware specification of the local cluster. | 20 |
| 4.2 | Overview of evaluated computing environments. | 21 |
| 4.3 | Overview of evaluated computing frameworks. | 22 |
| 4.4 | Overview of evaluated user interfaces. | 22 |
| 4.5 | Overview of the endpoints of the REST backend. | 28 |

List of Algorithms

| | | |
|---|---|----|
| 1 | Closest point of approach and time to closest point of approach . . . | 30 |
| 2 | Risk of collision | 30 |
| 3 | Risk of collisions | 31 |
| 4 | Risk of collisions (optimized) | 31 |
| 5 | GDBSCAN | 38 |
| 6 | Validate cluster | 38 |

1

Introduction

To motivate the relevance of the work and why it is an interesting field of study, this chapter provides a background to the marine domain and the problems of interest. In addition, to get an overview of what has been investigated in the area, a review of related work in the field is provided. Finally, the aim clarifies the focus of this thesis whereas the limitations cover what had to be disregarded in order to reach the goals within the given time frame.

1.1 Background

Nowadays, merchant vessels use Automatic Identification System (AIS) transponders to broadcast information about position, course, speed and its navigational status. Originally, the purpose was solely collision avoidance, but as with many new technologies, other applications quickly emerge. For instance, decision makers across industries rely on AIS data for monitoring ship activity worldwide, and there are a number of applications, e.g. companies trading with commodities analyzing global commodity flows for growth prediction in specific regions [1]. AIS data is also used by governmental agencies to discover vessels performing illegal activities such as fishing in prohibited areas or smuggling [1]. In the future, many more applications in different areas are expected to emerge.

Currently, a challenge for users of AIS data is the rate at which new data is generated. Since vessels transmitting AIS messages are moving, messages have to be sampled often enough to realistically model the situation, which may be once every few seconds. For instance, in the Baltic Sea, this adds up to about 6,000,000,000 messages in a year, increasing the demands on the systems trying to use this data for serious marine traffic analyses.

In addition, since AIS was initially built to complement radar in collision avoidance, security was not a major concern. However, with new applications, there has been an increase in ways that users maliciously modify messages. For instance, it is not uncommon that vessels intentionally disable the AIS transponder when approaching illegal fishing waters, but there are also vessels that purposely ob-

secure their destination for strategic purposes. Another example is what is known as spoofing, i.e. users that create ghost ships where none exist, something that may create political tension in certain areas [1]. Since users of AIS data perceive it as a reliable source of information, this is increasingly becoming an issue and there is a demand for ways of verifying the validity of AIS messages.

1.2 Related work

A study by Harati-Mokhtari et al. sheds light on the problem of unreliable AIS data and states that “mariners cannot wholly trust the equipment” and that “this could further jeopardise AIS usage and data quality” [2], showing the importance of the problem at hand. Another organization is the U.S. Department of Transportation which claims that about half of the vessels in their waters have erroneous AIS data. Of these errors, one third are identification errors, one third measurement errors and one third are a combination of the two [3].

Closely related to this thesis is a study by Abghari and Kazemi which investigates the potential of using open data sources in maritime surveillance and proposes a framework for what is called Anomaly Detection, namely to detect events that do not conform to expected behaviour [4]. In the maritime setting, this could for instance mean an event where a vessel has requested a pilot but has not used the service, something that the coast guard would want to investigate.

An organization which has shown interest in the issue of validating AIS messages using static analysis is the Danish Maritime Safety Administration (DaMSA) [5]. By inspection of the AIS messages, the DaMSA has developed a procedure to detect vessels with erroneous International Maritime Organization (IMO) numbers or Maritime Mobile Service Identity (MMSI) numbers. However, analyzing the MMSI number is only one of many ways for validating AIS messages, and exploring and identifying other ways is an objective of this thesis.

At Chalmers’ department of Shipping and Marine Technology, AIS data is used in research, e.g. by replaying interesting scenarios from real situations in a simulator, but also to analyze how vessels handle critical situations in relation to the regulations of the studied area. This latter part is the main application area of a system called Marine Traffic Analyser 3 (MTA3) [6], which was developed at the department. This system is used as a basis in this thesis when determining key features to implement in a new system. Currently, the system runs on a single computer and the limited amount of memory and computation speed has become a bottleneck when handling large amounts of data. In addition, the system does not offer any procedure for determining whether AIS messages are valid or not, something that may lead to insecurity in the analysis.

1.3 Aim

Since the current system has reached its limits with regards to performance and scalability, the main goal of this thesis was:

- To develop a new system for handling and analyzing AIS data of a larger scale that supports key features from the existing system.

Because of the insecurity of the validity of AIS messages, an additional goal was:

- To explore validation methods for AIS data and their implementation in the system.

1.4 Limitations

Since systems can constantly be refined, prioritizations were needed to make this thesis feasible to complete within the given time. Thus, the limitations for this thesis were:

- Since focus was on developing a system for handling and analyzing AIS data of a large scale to facilitate more advanced use cases, it was not a priority to port the old application in its entirety.
- Developing a functionality capable of determining validity with complete certainty is a hard task. Therefore the intention was to use heuristics to assign some kind of trustworthiness.
- The possibility of acquiring new hardware or buying computing time was not dismissed, but the intention was to use existing hardware and open-source software as much as possible.

2

Theoretical background

This chapter aims to provide the reader with the necessary theoretical material needed to understand relevant techniques which have been used during the work of this thesis.

2.1 State of the art for Big data: processing and storage

Since the field of Big data has been extensively studied, especially in the last decade, many techniques both regarding processing and storage have been developed. This section covers theory which is central to understand the Big data techniques used in this thesis.

2.1.1 Computing abstractions

Many of the existing techniques for parallelization stem from the same patterns and concepts. This section intends to explain the central abstractions needed to understand later chapters.

MapReduce

With Google's attempt to index the entire web, the need for processing large data volumes became apparent. Many special-purpose programs for different types of processing were developed and needed to run in a distributed environment on hundreds of computers. However, even with simple programs, the code needed for parallelization, fault tolerance and data distribution made the programs unnecessarily complex. To mitigate this issue, the MapReduce abstraction [7], based on the map and reduce primitives from functional programming languages, was proposed.

The main MapReduce program works on files and runs on multiple computers with a specific node acting as the master and the others as workers. The master

splits up the input file into smaller chunks and assigns map and reduce tasks to the workers. The map workers read and process their chunk of the data, write the intermediate results to disk and tell the master where the results are located. With information from the master and using remote procedure calls, the reducers are now able to find their respective data, process it and append the results to an output file.

To summarize; by writing mapper and reducer functions, the programmer is relieved from the burden of having to write code for dealing with parallelization, fault tolerance and data distribution and can focus on solving the task at hand.

Resilient Distributed Datasets

While MapReduce provides many desirable features including parallelization, fault tolerance and data distribution, it is based on a model which repeatedly reads from disk and writes to disk, operations which take a long time. This imposes a limitation especially noticeable in iterative algorithms where the results have to be written to disk in each iteration. To mitigate this issue, a Resilient Distributed Dataset (RDD) [8] introduces a distributed memory abstraction which provides the same benefits as MapReduce, e.g. parallelization, fault tolerance and data distribution, but with in-memory speeds.

An RDD is a read-only, partitioned collection of items and can be created either by initializing with data from disk or by *transformations* such as map, filter and join which can be applied to multiple data items in an RDD at once. Another type of operation which can be applied to an RDD is what is called *actions*, e.g. count, collect and save. The conceptual difference between transformations and actions is that a transformation states *how* the underlying data should be transformed which means that it does not extract any data, whereas an action will extract the results, something that will start the actual computations. Furthermore, the user is able to specify persistence of an RDD, meaning that if a user knows that a certain RDD will be needed multiple times, explicit control of persistence allows for persisting the RDD in memory to improve performance of repeated usage of the same data. In addition, it is also possible to manually control the partitioning of the data, allowing for optimizations based on data locality. To provide fault tolerance, RDDs use a concept called *lineage*. This means that if an RDD is lost there exists enough information from the other RDDs to recompute that specific partition, resulting in data recovery without the need to do costly replication.

2.1.2 Locality-sensitive hashing

In order to speed up data lookups, a technique called hashing is often used. A common strategy is to use a hash table which groups items into buckets according to keys. When searching for an item, the key is used to determine in which bucket to look to avoid searching through the whole space of items. However, conventional hashing does not say anything about how similar items are mapped. This is where the concept of locality-sensitive hashing (LSH) [9] comes into play. LSH makes use of similarity metrics to maximize the probability that similar items are grouped into nearby buckets, something that is desirable if one wants to make use of a hash table without losing the similarity relation of the data.

2.1.3 Z-order curve

Conventional hashing maps a one-dimensional value to a specific bucket, but if one wants to use hashing for multidimensional data such as coordinate points while preserving locality, one needs to use different techniques. Among them is the Z-order curve [10]. For coordinate points in a geographic coordinate system, this works by simply interleaving the binary representation of the latitude and longitude in order to create a one-dimensional representation, which can then be used in a hash table. This is visualized in Figure 2.1, where e.g. latitude 0 and longitude 1 would belong to the lower right square. The general idea for this can be seen in Figure 2.2, where the concept has been applied to a map.

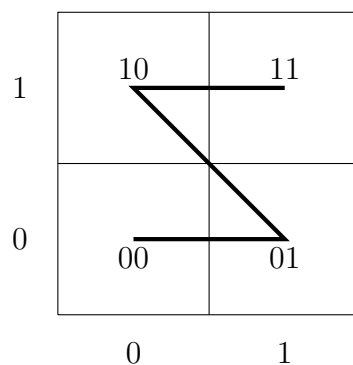


Figure 2.1: Simple case of a Z-order curve, by interleaving the binary representation of the axis, each square represents a distinct combination.

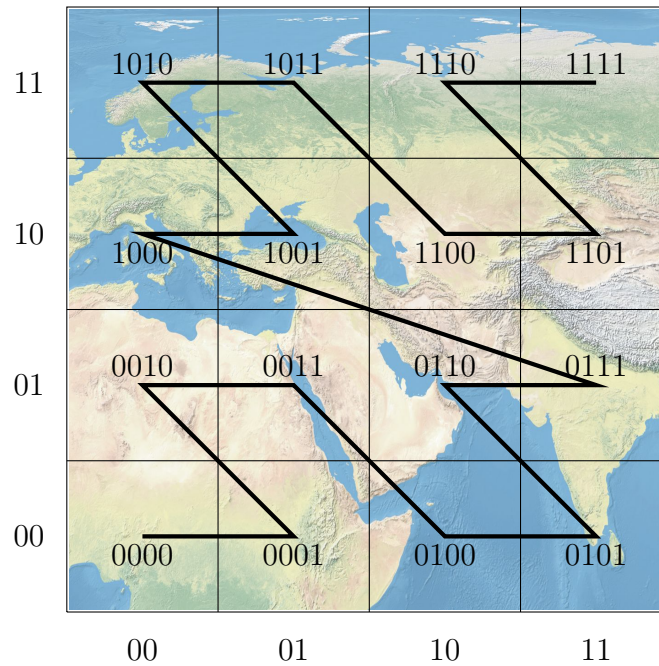


Figure 2.2: Example of a Z-order curve applied to a map, where the latitude-longitude values are mapped from two dimensions to one dimension to allow for hashing of two-dimensional values.

2.1.4 Distributed file system

One way to allow for storage of large data sets reliably as well as to provide the ability to stream data to user applications at a high bandwidth is to use a Distributed File System (DFS). Common features of a DFS are redundancy and location-independent addressing, where redundancy improves the reliability of the data by simply replicating data on multiple nodes in the system. The actual file transfers in a distributed file system are achieved through the usage of network protocols such as Ethernet. Location-independent addressing allows a user to access data in the same way as in a regular file system, i.e. without being concerned about where in the network the actual data resides. In addition, a DFS provides many desirable features commonly found in regular file systems, e.g. listing of directories, a permissions model etc.

2.2 Cluster analysis

The main objective of cluster analysis, or clustering, is to group sets of objects into clusters, such that the objects of each cluster are in some way more similar than

objects within other clusters. Clustering is used in several fields which concern information extraction from data sets, among them exploratory data mining and machine learning. Since clustering is not a specific algorithm, there exist methods of all kinds. The variant which has been the most important in this thesis has been density-based clustering where clusters are defined as regions consisting of a higher density of points compared to other regions. This section describes the specific algorithm used for density-based clustering in this thesis.

2.2.1 Generalized Density Based Spatial Clustering of Applications with Noise

In [11], Ester et. al introduce the Density-based spatial clustering of applications with noise (DBSCAN) algorithm. The main objective of the DBSCAN algorithm is to cluster regions with many nearby points (dense regions) as well as to mark outliers as noise. The main components for accomplishing this are the following:

- The radius that decides the points which are considered as neighbors to a specific point.
- The minimum number of points that is considered a cluster.

In contrast to many other clustering algorithms, DBSCAN provides two valuable properties especially important in this thesis:

- It is not *a priori* required to estimate the number of clusters the data set consist of.
- The ability to cluster regions of arbitrary shape.

However, drawbacks of the DBSCAN algorithm are both that the definition of a neighborhood is limited to a radius, but also that the number of points is what determines the density of a cluster. To improve this, the same authors introduced the Generalized Density Based Spatial Clustering of Applications with Noise (GDBSCAN) in [12]. GDBSCAN extends the notion of a neighborhood to any notion of a neighborhood of an object as long as the binary predicate determining similarity between objects is reflexive and symmetric. In essence, this has the effect that it is possible to cluster objects using both their spatial and nonspatial attributes. In addition, GDBSCAN generalizes how one can determine the density of a cluster using other measures than the minimum number of objects that is considered a cluster. For instance, if one is clustering polygons, it might be more suitable to cluster the polygons using the intersection metric rather than just the number of polygons in each cluster. Another possibility is to cluster objects based on e.g. a range of a certain nonspatial attribute, meaning that objects within that range will be clustered together.

In terms of efficiency, the performance depends mainly on the implementation. Since each object in the data set needs to be inspected and have its neighborhood

calculated, a naïve implementation of GDBSCAN has a time complexity in the order of N^2 , where N is the number of objects. However, using spatial indexing structures, the time complexity for retrieving the neighborhood can be reduced to $\log(N)$ implying a time complexity of $\mathcal{O}(N \log(N))$, achieving sufficient performance even for large data sets.

2.3 AIS data description

AIS messages contain extensive information about a ship's attributes and status. However, only a few of these are related to the ship's movements and therefore, most of the parameters are not used in this thesis. Some of the ones used are rather self explanatory, such as timestamp, heading, longitude, latitude, etc., whereas some of them are more complicated, and explained below.

Maritime Mobile Service Identity Each vessel has a unique identification number called Maritime Mobile Service Identity (MMSI) which is used for distinguishing one vessel from another.

Course Over Ground In contrast to the heading, Course Over Ground (COG) is the direction of a vessel when wind and currents are taken into account. COG uses the units of a compass, i.e. degrees clockwise from north.

Speed Over Ground The speed of a vessel is given by the Speed Over Ground (SOG) parameter, taking wind and currents into account.

2.4 Ship handling terms

To understand the domain specific problems in this thesis, a few terms from the maritime domain are explained here.

Navigation terms Determination of directions between ships or objects at sea can be done in different ways. One way is to use the bearing, defined as the clockwise angle between north and the direction to the target. Another way is to use the relative bearing, defined as the clockwise angle between the own vessel's heading and the direction to the target.

Detecting risk of collision There exist two common metrics that are especially important when analysing risk of collision situations between vessels. These are the Closest Point of Approach (CPA) and the Time to Closest Point of Approach (TCPA). CPA is defined as the minimum distance between two vessels if they both continue with the same course and speed, whereas TCPA is the time until the CPA is reached.

2. Theoretical background

3

Method

This chapter is divided into two parts, where the first part covers the workflow when developing the data analysis system and the second part covers the development of the initial applications for the system.

3.1 Handling: Development of a large scale data analysis system

This section describes how the work was conducted when determining the requirements from the users as well as the strategy for narrowing down the range of admissible solutions. In addition, this section covers how the technique survey was conducted as well as the strategy for integrating the solutions into a complete system.

3.1.1 Requirements and admissible solutions

To get a fully functional and well performing system there were some relevant parameters that had to be specified. It was important to have a functional specification early in order to know which sort of techniques were required. This section describes how the functional specification was developed and how the techniques of the different parts were evaluated.

Functional specification

Starting off, the plan was to identify system requirements such as key functionality in collaboration with users of the current system. This was done using an approach inspired by the “20 queries” law defined by Gray, mentioned by Szalay and Blakeley in [13]. To reach a common ground upon which developers and users of the system can communicate about features and trade-offs, the approach suggests that the users of the system formulate the most important questions that they would like to pose to the system. This part resulted in documentation of what the user

expected from the system and helped to get a good grasp of the end goal of the system.

Computing environment

In order to deliver a system which adheres to its requirements, it is important to select a suitable computing environment. Since the main objective of the system developed in this thesis was Big data analysis, performance and the ability to handle large files were naturally the key metrics. Even though the possibility of buying new hardware or computing time was not dismissed, it was desirable to use available resources. Additionally, it was important that the environment supported adding more hardware without introducing additional complexity, since this allows the system to scale. Finally, since the goal was that the end product should be usable by users with varying amounts of computer experience, a low complexity would be advantageous. The metrics that were considered important when evaluating the computing environment were therefore *performance*, *price*, *scalability* and *complexity*.

Computing framework

Since speedup was the main objective, the performance of the framework was highly prioritized, but because of the limited time frame of this thesis, it was desirable that the framework supported a language that were familiar to the authors of this thesis. Another desirable feature was a mature framework with rich documentation, many code examples and a good compatibility with other techniques. The initial computations were batch oriented, but since there was a possibility of more stream oriented computations in the future, the framework should ideally support both. To conclude, when evaluating the computing framework, the metrics that were important were *programming language*, *processing type*, *features* as well as *compatibility*.

User interface

To make the system as usable as possible it was desirable to develop a user interface that requires as little understanding of the underlying functionality as possible. However, since the user interface was not the main focus, the development phase was not supposed to be prohibitively time consuming. Since the system will be further developed in the future, maintainability was an important parameter as well. Therefore, the metrics that were considered in the evaluation of the user interface were *development time*, *ease of use* and *maintainability*.

3.1.2 Technique survey

Using the functional specification defined by the user and the proposed range of admissible solutions, the work during the technique survey consisted of using key parameters for evaluating and deciding on suitable technologies within the different parts of the system.

3.1.3 Integration

Building a system capable of handling large data volumes entails a certain complexity. In addition, since the field of Big data is constantly evolving with continuous emergence of new frameworks and techniques, there are numerous challenges. First, there is the risk of developing an overly complex system, which only works if every single component is functional. Second, if a user of the system would like to upgrade a component, it is not desirable having to replace the complete system. To avoid issues like this, the intention was to draw inspiration from the fifth law by Gray [13], called "working to working" when integrating the techniques into a complete system. This means focusing on modularity, where each of the components can be replaced without interfering with the rest of the system, using a service-oriented architecture. Furthermore, the plan was to get a prototype up and running quickly, something that would allow experimentation with the system before polishing each separate part.

3.1.4 Evaluation

After implementing the system, a number of tests were made in order to test the system's correctness and performance. This section covers which tests that were decided to be made to fulfill these criteria.

Correctness

A key part of the evaluation was to ensure the correctness of the system. This was done by using unit tests to ensure the correctness of the underlying functions and by comparing the output against the output from MTA3 to ensure the correctness of the whole system, i.e. that the same relevant situations were detected.

Performance

To assess the performance, the system was evaluated by comparing it against the performance of MTA3. This was done both with respect to execution time and the capability of handling large amount of data. However, since the system only

contains a subset of the features of MTA3, the evaluation only considered these features.

3.2 Analysing: Development of initial applications for the system

This section covers how the work progressed when developing an application with the purpose of assessing the performance of the system as well as an application for validating AIS messages.

3.2.1 MTA3 Parallel

In order to test the performance of the system, an application with key functionality from MTA3 amenable to parallelization using the chosen techniques was implemented. Since there was not enough time to port all functions from MTA3, it was important to prioritize which functions to implement. This section describes how the selection of functions was made as well as the strategy for parallelization.

Selection of key functionality

The functions that were chosen to be ported from MTA3 were chosen both using the functional specification (see Section 3.1.1), but also to have a good way of benchmarking the performance of the entire system in order to expose potential bottlenecks. Additionally, since the theoretical speedup of a function is limited by its longest sequential part according to Amdahl's law (originating from [14]), it was important to choose functions that were not inherently sequential in order to be able to benefit from parallelization.

Improvement of the algorithms

To improve the performance of the developed application, there were opportunities both to optimize the domain specific algorithms by studying them in detail, but also to focus on optimizing the parallelized versions implemented in the chosen framework. Since the users of the systems are experts in the specific domain and well aware of the details of the algorithms, it was decided that the system would benefit the most if the work of this thesis focused on optimizations related to parallelism, i.e. how to efficiently leverage the computing environment and framework to achieve maximum speedup.

3.2.2 AIS message validation

Since the AIS ecosystem is built upon the fact that anyone can broadcast AIS messages, it is easy to manipulate the AIS transmitter to send erroneous data. As mentioned in Section 1.1, two of the most common AIS manipulation methods are shutting off the transmitter and “spoofing”. In addition to these, identity fraud, GPS manipulation and obscuring destinations are the most common methods of manipulation [1].

This section covers how the problem of validating manipulated AIS data was approached using the following four steps: how a problem of a suitable scope was selected, how it was approached, how the solution was implemented and finally, how the solution was evaluated.

Problem selection

First, through discussions with a domain expert, an interesting and relevant problem had to be selected. To achieve this, an investigation of the domain for previous attempts at solving the problem of AIS validation was conducted. After the investigation, the plan was to find an approach which should be possible to implement within the given time frame.

Literature survey

After narrowing down the task, a literature survey of interesting techniques was performed in order to get an overview of potential algorithms and what might be applicable to the problem at hand.

Implementation

The plan for the implementation part was to focus on getting an initial attempt at solving the problem up and running before implementing a parallelized version on the system.

Evaluation

The problem of AIS validation has been studied before, but to the knowledge of the authors of this thesis, there is no data consisting of documented cases of invalid data openly accessible. In addition, there is no measure of how often the specific problem occurs in real situations, which means that it might be too time consuming to use real data. Therefore, the evaluation focused on constructing dummy data to realistically model interesting scenarios which could be a sign of invalid AIS messages, and thereby get an indication of how well the solution performs.

3. Method

4

Results and discussion

This chapter covers the results and discussions related to the work of this thesis. The results are divided into two separate sections where the first part covers the system specific results and the second part covers the analysis specific results. Finally, the chapter ends with a discussion about ethical considerations in the context of Big data analysis within the maritime domain.

4.1 Handling: Development of a large scale data analysis system

This section covers the results of constructing a large scale data analysis system using the “20 queries” methodology, the results from the technique survey as well as how the techniques were integrated into a complete system.

4.1.1 Requirements

As a result of the "20 queries" method, the developer of the MTA3 program system came up with the following queries that the system should be able to answer:

- Find situations within a range of six nautical miles where the CPA was below a specific distance.
- Find situations within a range of six nautical miles where the TCPA was below a specific time.
- Find situations within a range of six nautical miles where the course change was larger than a specific degree.
- Find situations where the relative bearing was within a given range in degrees.
- Find situations where the course difference was within a given range in degrees.
- Find situations where the speed difference was within a given range in knots.
- When did a situation, fulfilling specific criteria, occur?
- Where did a situation, fulfilling specific criteria, occur?

- Find vessels according to vessel type(s).
- Which vessels were involved in the situation?
- What was the closest distance between the vessels?

Using these results, a good view of what kind of features that were desirable and how to prioritize the work was achieved.

4.1.2 Technique survey

This section describes the outcome of the survey, including which of the techniques were chosen as well as motivations to why.

Computing environment

As a result from the survey, it was decided to use the local cluster that was available at the institution. To maximize the performance, a distributed file system (described in Section 2.1.4) called The Hadoop Distributed File System [15] was used. The local cluster consisted of six nodes: one head node and five compute nodes and a technical specification of the nodes can be seen in Table 4.1.

An alternative which was considered was to use an optimized desktop application, which would give the benefits of straight-forward development compared to a multi-node system such as a cluster. However, a desktop application has certain drawbacks, e.g. single point of failure, memory limitations and vertical scaling which in this case would require specialized hardware.

Another solution would be to use a cloud computing environment where variable computing power can be bought depending on the requirements. However, before paying for computing power, it is desirable to see how parallelizable a problem is to make sure it is a worthwhile investment. Therefore, the choice fell on a local cluster, even though this entails additional complexity when setting up the cluster environment or when adding new hardware. Table 4.2 shows a summary of the computing environments and the parameters they were evaluated by.

Computing framework

After weighing the advantages and drawbacks, it was evident that the computing framework Apache Spark [16] was the most promising alternative. Apache Spark

Table 4.1: Hardware specification of the local cluster.

| Type | CPU | RAM | Storage |
|--------------|-----------------|-------|---------|
| Head node | Single, 2.4 GHz | 8 GB | 500 GB |
| Compute node | Dual, 2.8 GHz | 16 GB | 250 GB |

Table 4.2: Overview of evaluated computing environments.

| Environment | Performance | Price | Scalability | Complexity |
|-----------------|-------------|----------|-------------|------------|
| Desktop | Low | Low | Low | Low |
| Local cluster | High | Low | High | High |
| Cloud computing | Very high | Low-High | High | Medium |

is based on the RDD abstraction described in Section 2.1.1.

In the evaluation of computing frameworks, the most important metric was that the computations would be batch-oriented, excluding frameworks such as Apache Storm [17] and Apache Flink [18] which are oriented towards streaming. However, since a natural step in the development of the system would be support for stream processing, it was important to choose a framework which supports both processing types. With this in mind, MapReduce [19] (described in Section 2.1.1) which only supports batch processing, could be excluded.

The two remaining candidates were Google Cloud DataFlow [20], based on [21], and Apache Spark. A desirable feature of Apache Spark is that it provides an interpreter, which enables interactively running computations on the data set under analysis. Another advantage was the possibility to use the Python programming language, partly since it is generally considered to be an easy language to learn, and would thus ease the development and maintenance for future users of the system but also that the authors of this thesis had previous experience with the language. In addition, Python is also well known for being a language with a large set of libraries providing extensive functionality. Furthermore, since Apache Spark uses in-memory computations instead of saving results to disk in each iteration like MapReduce, the performance is very high - especially for iterative algorithms. Table 4.3 shows the different computing frameworks that were considered and the parameters they were evaluated by.

User interface

To provide a user-friendly interface together with the possibility to run the system from any computer connected to the network, a web application was chosen.

An alternative which was considered but dismissed was to let the user interact with the system using the command line, since such a solution would make it hard for inexperienced users to interact with the system. Another alternative to increase the ease of use was to develop a desktop application, but this has the drawback of having to distribute new versions each time the application is updated. In addition, it would require manual installation of the application on every computer. With a web application, when new features are added, it is sufficient to update the web application instead of forcing the user to update, eliminating the issue of having

Table 4.3: Overview of evaluated computing frameworks.

| Framework | Language | Type | Compatibility |
|-----------------------|---------------------|------------------|------------------|
| Google Cloud Dataflow | Java | Batch, streaming | Google services |
| Apache Spark | Java, Scala, Python | Batch, streaming | Hadoop ecosystem |
| Apache Storm | Any language | Streaming | Hadoop ecosystem |
| Apache MapReduce | Any language | Batch | Hadoop ecosystem |
| Apache Flink | Java, Scala | Streaming | Hadoop ecosystem |

Table 4.4: Overview of evaluated user interfaces.

| User interface | Development time | Ease of use | Maintainability |
|---------------------|------------------|-------------|-----------------|
| Command line | Low | Low | Hard to upgrade |
| Desktop application | Medium | High | Hard to upgrade |
| Web application | Medium | High | Easy to upgrade |

to keep different versions of the application in sync. Table 4.4 shows a summary of the user interface candidates and which parameters they were evaluated by.

4.1.3 Integration

This section covers how the different components were integrated into the system.

Web frontend

To allow for easy access to the most important functionality of the system by multiple users at the same time, a simple web frontend was developed. Using HTML, CSS and JavaScript, the web front end translates a user's intentions to HTTP requests which are then processed by the web backend. The web interface is divided into five tabs: *info*, *new job*, *job status*, *path plotter* and *handle data*.

The *info* tab consists of a simple landing page with a summary of the capabilities of the system.

In the *new job* tab, a user is able to upload a data set or choose an existing data set and then execute either a *filter* operation or a *calculation*. The part of

the web interface that concerns filtering can be seen in Figure 4.1. The purpose of a filter is self-explanatory, i.e. it takes a data set and filters out all messages fulfilling one or more condition(s). A common use case could be that the user is interested in studying a certain geographical area during a specific time period, something that is easily accomplished by using a filter. The calculation operation performs calculations within combinations of vessels, e.g. the distance between all vessels in a data set, a much more computationally intensive task. This part of the interface can be seen in Figure 4.2. A benefit of separating this into its own section is that since the data set needs to be scanned while filtering a data set, it is easy to calculate the number of messages and provide a time estimation for more costly calculations, a feature that is desirable when execution times span from hours to weeks. The components of the filter section are forms enabling the user to input text data and a map which provides the user with the option to specify a geographical region from a map as depicted in Figure 4.1. The latter functionality was implemented by leveraging the Google Maps JavaScript API which displays an interactive map and allows the user to draw a rectangular box to specify the area of interest, something that can be used to construct a condition that filters out messages from a certain area.

The *job status* tab serves two purposes, where the first is to complement the Spark Web UI, which provides a detailed overview of all running jobs, including details about performance, environment settings etc. This is key for a developer of Spark applications and a system administrator, but for users only interested in specific AIS analytics it is perhaps a bit overwhelming. Second, the purpose is to provide the user with the functionality of extracting the results from the jobs. With this in mind, a more basic interface which displays the currently running jobs as well as the finished jobs along with their results was chosen for implementation. This interface can be seen in Figure 4.3.

The *path plotter* interface provides a simple way for a user to choose a data set and plot one or more vessels' voyages, providing information about each specific data point. In addition, the interface provides a way to evaluate the correctness of the risk of collision calculations since vessel paths can be investigated in detail. The interface is displayed in its entirety in Figure 4.4.

The part called *handle data*, which can be seen in Figure 4.5, simply displays an overview of all files that are available for analysis, and gives the user the ability to upload additional files.

4. Results and discussion

MTA3 Parallel

[INFO](#) [NEW JOB](#) [JOB STATUS](#) [PATH PLOTTER](#) [HANDLE DATA](#)

Filters

AIS file
100_vessels_28mb.csv

Start time

End time

Minimum speed (kn)

Vessel type
 High-Speed Craft Passenger Cargo Tanker All




Figure 4.1: Overview of the interface which allows the user to select a specific data set and interactively choose a region and filter out vessels according to time period, minimum speed and vessel type.

Calculations

Closest point of approach limit (NM)

Time to closest point of approach limit (h)

Relative bearing limit

Manoeuver limit

Course difference limit

Speed difference limit

Type of calculation

Figure 4.2: Overview of the interface where risk of collision calculations are started. The input fields allow the user to specify which of the situations that are of interest.

MTA3 Parallel

[INFO](#) [NEW JOB](#) [JOB STATUS](#) [PATH PLOTTER](#) [HANDLE DATA](#)

Running jobs

| Job ID | File | Type |
|--------------------------------------|---------------|--------|
| 0cf4b1bc-138a-43bc-ac2e-e6e752242c7a | ais_200mb.csv | Filter |

Finished jobs

| Job ID | File | Type | Results | Results (CSV) |
|---------------------------------------|--------------------------------|--------|---------|--------------------------|
| 2e75808d-0304-46bb-8f03-1d6ec246498e | ais_messages_fredrik_547mb.csv | Filter | 1115610 | Download |
| 9d093253-1248-4c63-832b-c24899fa7cfe | ais_200mb.csv | Filter | 278982 | Download |
| 4143d817-1c64-43d8-882b-7893f0187b3d | 50_vessels_17mb.csv | Filter | 26935 | Download |
| 2e334b6c-cca1a-457c-998c-a3173693c4ac | 50_vessels_17mb.csv | Filter | 26935 | Download |

Figure 4.3: Overview of the interface which displays information about running and finished jobs as well as the results from each job.

4. Results and discussion

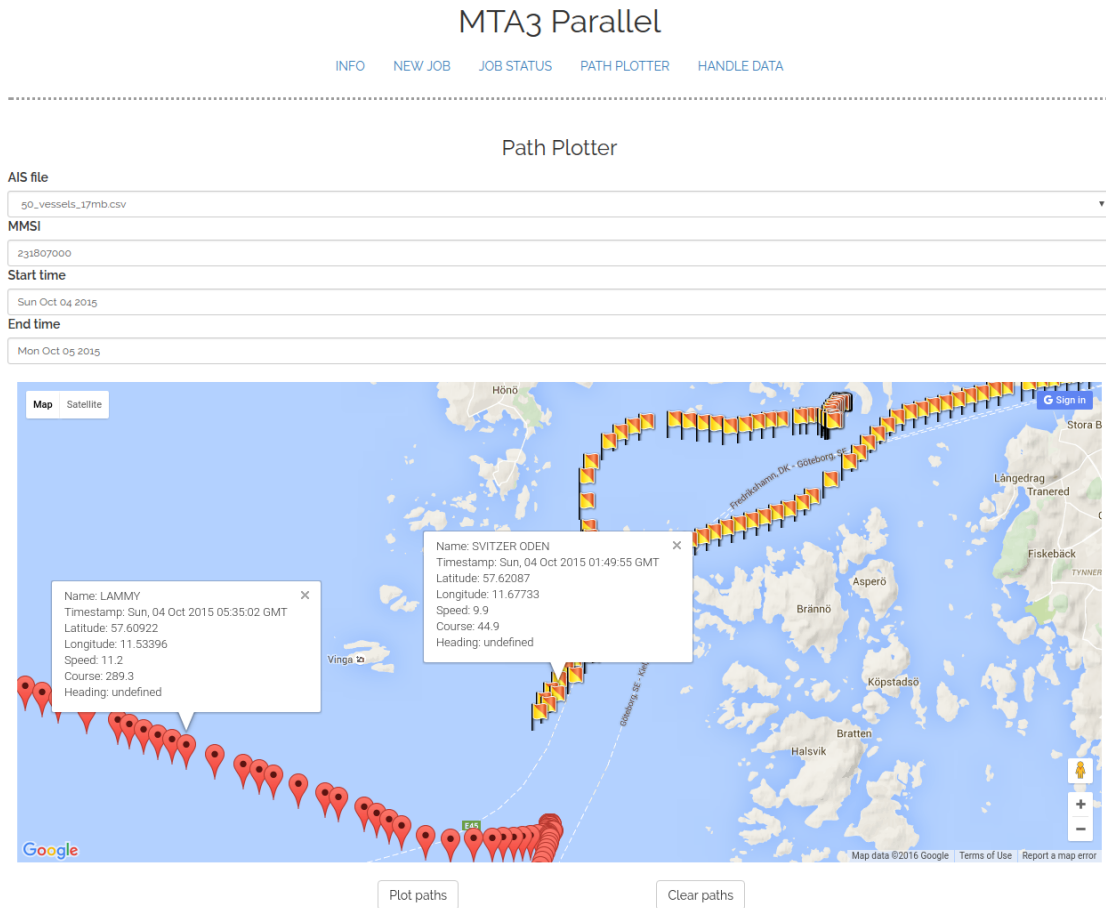


Figure 4.4: Overview of the interface where the user can choose a data set and plot selected vessel paths according to identification number (MMSI) and time period. Each data point is clickable and provides detailed information.

MTA3 Parallel

[INFO](#) [NEW JOB](#) [JOB STATUS](#) [PATH PLOTTER](#) [HANDLE DATA](#)

Upload file

New AIS file (.csv)

Choose File No file chosen

Upload file

Available AIS files

| Filename | File size (MB) |
|---|----------------|
| 100_vessels_28mb.csv | 0 |
| 200_vessels_54mb.csv | 0 |
| 20160428-112401_very_large_ais_messages.csv | 0 |
| 50_vessels_17mb.csv | 0 |
| ais_200mb.csv | 0 |
| ais_messages_1007mb.csv | 0 |
| ais_messages_fredrik_547mb.csv | 0 |

Figure 4.5: Overview of the interface for handling data sets, where the user is able to upload data sets to the distributed file system.

Web backend

To provide the web client with the ability to access the capabilities of the cluster, upload files and see the status of running jobs, the web backend exposes a Representational State Transfer (REST) API with the endpoints listed in Table 4.5.

The REST API translates HTTP verbs (GET, POST, PUT, DELETE, etc.) to actions on actual jobs on the cluster. For instance, a GET request to the `/jobs` URL retrieves a list of all running jobs and a POST request to `/jobs` starts a job on the cluster using the parameters contained in the request. A complication which occurred was that even though jobs may have long running times, the server must immediately respond to the client to avoid freezing the user interface. This is handled through the usage of a Python library called Celery [22] which uses message passing to implement an asynchronous job queue, allowing for the web server to start a job in the background while asynchronously responding to the client. Additionally, the Spark framework uses something called a Spark Context which handles the connection to the cluster. This context is used when expressing the computations, e.g. creating RDDs, transforming RDDs and performing actions on RDDs. However, a limitation is that the framework only supports having one context per application, meaning that concurrent users of the system must use the same context. In addition to mitigating the issue of freezing the user interface, the asynchronous job queue is used for handling the shared Spark Context to avoid concurrency issues as well as to provide a simple First-In-First-Out job queue.

Table 4.5: Overview of the endpoints of the REST backend.

| URL | HTTP Method | Purpose |
|-------------------------------|-------------|---|
| <code>/jobs</code> | GET | Fetches info about all jobs |
| <code>/jobs/:id</code> | GET | Fetches info about a specific job |
| <code>/jobs</code> | POST | Starts a new job |
| <code>/files</code> | GET | Fetches info about all available data sets |
| <code>/files/:id</code> | GET | Fetches info about a specific data set |
| <code>/files</code> | POST | Adds a data set |
| <code>results/:file_id</code> | GET | Fetches results from processing a data set with id <code>file_id</code> |

4.2 Analysing: Development of initial applications for the system

The application that was developed for testing the system was the part of MTA3 which concerns risk of collision situations at sea. In addition, an application for validating AIS messages was developed. This section contains the results from the development as well as the evaluation of the applications, regarding both correctness and performance.

4.2.1 MTA3 Parallel

The first part of the application concerns determining risk of collision situations between vessels. This section covers how the algorithms were implemented and the evaluation including correctness, performance and output comparison with MTA3.

Algorithms

This section covers the most important algorithms for determining risk of collision situations using the same approach as in MTA3, as well as parallelized versions of the algorithms. For brevity, the algorithms are described in high level pseudocode. The implementations of the algorithms can be seen in their entirety in Appendix A.

CPA and TCPA A key algorithm for determining risk of collision is the algorithm for determining CPA and TCPA using two AIS messages. Let m_i and m_j denote the messages from vessel i and vessel j , respectively. Note that the algorithm ignores the timestamps of the messages, but this is handled further on in the optimized version which takes the timestamps of the messages into account. The algorithm can be seen in Algorithm 1.

Algorithm 1 Closest point of approach and time to closest point of approach

```

1: function CPA_TCPA( $m_i, m_j$ )
2:    $v_i \leftarrow$  velocity of  $m_i$  using speed and course
3:    $v_j \leftarrow$  velocity of  $m_j$  using speed and course
4:    $relative\_velocity \leftarrow v_i - v_j$ 
5:    $relative\_velocity_x \leftarrow$  x component of  $relative\_velocity$ 
6:    $relative\_velocity_y \leftarrow$  y component of  $relative\_velocity$ 
7:   if  $relative\_velocity_x \neq 0$  or  $relative\_velocity_y \neq 0$  then
8:      $heading \leftarrow$  direction of  $relative\_velocity$ 
9:      $speed \leftarrow$  length of  $relative\_velocity$ 
10:     $bearing \leftarrow$  clockwise angle from north to  $m_j$ 's position seen from  $m_i$ 
11:     $relative\_bearing \leftarrow$  relative bearing between  $m_i$  and  $m_j$ 
12:     $distance \leftarrow$  distance between  $m_i$ 's position and  $m_j$ 's position
13:     $CPA \leftarrow distance * \sin(|relative\_bearing|)$ 
14:     $TCPA \leftarrow (distance * \cos(|relative\_bearing|))/speed$ 
15:    return ( $CPA, TCPA$ )

```

Risk of collision The determination of a risk of collision situation depends on the CPA and TCPA between two vessels. Let m_i denote a message from vessel i and m_j denote a message from vessel j . Let cpa denote the CPA and $tcpa$ denote the TCPA between the first and the second vessel. Let the variables cpa_limit and $tcpa_limit$ denote the maximum distance of the CPA and the maximum time for the TCPA to count as a risk of collision situation. The algorithm can be seen in Algorithm 2.

Algorithm 2 Risk of collision

```

1: function ROC( $m_i, m_j, cpa\_limit, tcpa\_limit, time\_limit$ )
2:    $t\_diff \leftarrow$  time difference between  $m_i$  and  $m_j$ 
3:   if  $t\_diff \leq time\_limit$  then
4:     ( $cpa, tcpa$ )  $\leftarrow$  CPA_TCPA( $m_i, m_j$ )
5:     return  $0 \leq cpa \leq cpa\_limit \ \& \ 0 \leq tcpa \leq tcpa\_limit$ 

```

To determine the risk of collision between vessels, one can calculate the CPA and TCPA between all pairs and filter out the pairs which do not satisfy risk of collision. Let M denote the set of all messages and the variables cpa_limit and $tcpa_limit$ denote the maximum distance of the CPA and the maximum time for the CPA for counting as a risk of collision situation. The algorithm can be seen in Algorithm 3.

Since this solution generates all pairs of messages and calculates the risk of collision between each pair, this will return the correct results. However, with

Algorithm 3 Risk of collisions

```

1: function ROCS( $M, cpa\_limit, tcpa\_limit, time\_limit$ )
2:    $pairs \leftarrow$  generate all possible pairs in  $M$ 
3:    $cpa\_tcpa \leftarrow$  map the CPA function on each pair in  $pairs$ 
4:    $situations \leftarrow$  filter out items in  $cpa\_tcpa$  where ROC returns true
5:   return  $situations$ 

```

a time complexity of $\mathcal{O}(M^2)$, this solution quickly becomes infeasible when the number of messages grows.

Risk of collision (optimized) The optimized version of the algorithm for extracting all risk of collision situations can be seen in Algorithm 4, where M denotes the set of all messages and the variables cpa_limit , $tcpa_limit$ and $time_limit$ denote maximum CPA distance, maximum TCPA time and maximum time difference between messages for counting as a risk of collision situation.

Algorithm 4 Risk of collisions (optimized)

```

1: function ROCS( $M, cpa\_limit, tcpa\_limit, time\_limit$ )
2:    $buckets \leftarrow$  hash each message and assign to its respective bucket
3:    $all\_situations \leftarrow$  empty list
4:   for  $bucket \in buckets$  do
5:      $situations \leftarrow$  ROCS( $bucket, cpa\_limit, tcpa\_limit, time\_limit$ )
6:      $all\_situations \leftarrow all\_situations + situations$ 
7:   return  $all\_situations$ 

```

The main idea behind the optimization is that in order to speed up the calculations, an option is to sacrifice precision for performance, i.e. when looking at the risk of collision between vessels, it is only interesting to observe “nearby” vessels. Thus, it may be worthwhile to avoid calculating the risk of collision between vessels in Gothenburg, Sweden and Wellington, New Zealand to decrease the number of computations. In our case, “nearby” has two meanings: nearby in space and nearby in time. The optimization uses both space and time metrics by applying a technique for spatial and temporal hashing called spatiotemporal hashing [23]. The technique involves hashing the items (in our case messages) by space and time into buckets, using the concept of locality-sensitive hashing described in Section 2.1.2. However, even though the resulting time complexity is $\mathcal{O}(B * max(B))$, where B is the number of buckets and $max(B)$ is the size of the largest bucket, the worst case is still $\mathcal{O}(M^2)$, where M is the number of messages, since there exists at least a theoretical possibility that all messages are from the same time

interval and within the same geographical space and therefore end up in the same bucket.

In detail, the technique for spatial hashing that is used is called geohash [24] and it is an application of a Z-order curve, described in Section 2.1.3. Geohash involves dividing the geographical space into squares, where the size of each square is determined by the desired precision, ranging from meters to kilometers. Then, the vessel’s two-dimensional coordinate point is hashed to a one-dimensional value which represents the square the vessel belongs to. An issue is that it is not enough to hash vessels to a specific square and then compare within the square, since if a vessel is near the edge of the square, it is not unlikely that there are vessels in a nearby square that are within risk of collision. This means that vessels within a square must be compared against vessels within its specific square as well as all eight surrounding squares. However, since nearby points are hashed into nearby geohashes, it is easy to use the geohash scheme to do a proximity search when determining risk of collision between the vessels.

For temporal hashing, a similar approach called timehashing can be used. With timehashing, all messages with a timestamp within a specific time interval are hashed to the same bucket. Analog to the case of spatial hashing, there is the issue that it is not sufficient to compare vessels within the same time interval since there may be risks of collisions with vessels within nearby time intervals. However, since the scheme for hashing is known and the fact that the time is in one dimension, a comparison between adjacent time intervals can be conducted, resulting in a total of three time intervals per vessel.

To conclude, by combining spatial and temporal hashing into spatiotemporal hashing, each message will be placed in 3^3 buckets. Compared to the naïve approach, this uses more buckets and therefore more memory, but since the messages are spread out within the buckets, this decreases the size of the largest bucket. In addition, since the performance is determined by the size of the largest bucket, this results in a significant speedup.

The Spark implementation of the algorithm can be seen in Appendix A.1.4. The functionality for spatial hashing was implemented using the library *python-geohash* [25]. Temporal hashing was implemented using the Python library *timehash* [26] after the addition of a few missing features which were required for our particular application. This contribution can be seen in Appendix A.2.1.

Correctness

The correctness of the system was ensured using two different methods: unit testing and output comparison. The output was compared both against MTA3’s output and the output from a naïve solution, the latter to ensure that the optimizations did not throw away any relevant situations.

Tests In order to ensure the correctness of the ported functions from the MTA3 software, unit tests were developed, where the implementations of the CPA and TCPA functions were tested against the implementations used in the MTA3 software. However, minor deviations which are believed to come from precision in rounding, were found, but since these deviations were small, they were considered negligible in terms of the end results. In the future, the unit tests can be used as regression tests when extending the functionality of the system.

Output analysis To ensure that the situations which MTA3 registers are registered by MTA3 Parallel as well, comparisons of the outputs were made. Ideally, the same situations would be registered and therefore only relevant results from MTA3 Parallel would be saved for later analysis in MTA3. When comparing the results, there were situations which MTA3 Parallel registered but MTA3 did not. To understand why, a number of these situations were investigated in detail together with the domain expert and developer of MTA3 using the path plotter in the web interface, displayed in Figure 4.4. Additionally, since MTA3 has to sample in order to get reasonable performance, there is a risk of throwing away data that contains key information. When studying these situations manually, it was concluded that this was indeed the case and the situations were correctly registered as interesting situations by MTA3 Parallel, but not by MTA3.

Comparison of optimized and naïve solution To ensure that the optimized code did not discard any relevant situations, comparisons were made against a naïve solution which simply checks all combinations. The risk of the optimization is that the time window and the geographical space is too small, resulting in missed situations. By inspecting the output from the different algorithms, it was evident that the output from the naïve algorithm consisted of a few more risk of collision situations. The reason for this was concluded to be due to the fact that the naïve algorithm compares messages with a larger distance than six nautical miles, which is the upper limit from the given functional specification (see Section 4.1.1). However, since all the missed situations were outside the spatial range given in the functional specification, the optimized algorithm was deemed correct.

Performance

The second part of the evaluation concerns the performance of MTA3 Parallel. This was done by comparing the performance against MTA3 with respect to two metrics: speed and file size.

Speed comparison The speed comparisons were made with six files ranging from 17 MB to 1.3 GB, where the range was determined for compatibility reasons

with MTA3. Since MTA3 samples the input data for performance reasons, i.e. it uses one message per vessel every sixth minute, a sampling function in MTA3 Parallel was implemented to achieve a measure of the performance with approximately the same amount of computations. As is evident in Figure 4.6, the runtime of MTA3 shows an inclination to rapidly grow with increased file size at an early stage and lacks support for file sizes larger than 54 MB. This is even more clear in Figure 4.7, which shows a performance comparison only considering the file sizes which MTA3 was able to handle. Evident in Figure 4.6 is that MTA3 Parallel and MTA3 Parallel with sampling handle larger files at lower runtimes. However, MTA3 Parallel shows a trend where the runtime increases considerably when the files approach a certain size. This increase is believed to be due to the drawbacks of hashing - namely that the performance on datasets with a high concentration of vessels in certain regions will be determined by the number of vessels in the region with the highest concentration. This is not as evident in the latter case, which is mainly believed to be due to all the data points that are being discarded because of the sampling step, but would likely occur when run on files of larger sizes. To mitigate the issues of handling regions with a high concentration of vessels, an improvement of the underlying algorithm which would balance the regions, e.g. using spatial indexing structures, is believed to be greatly beneficial.

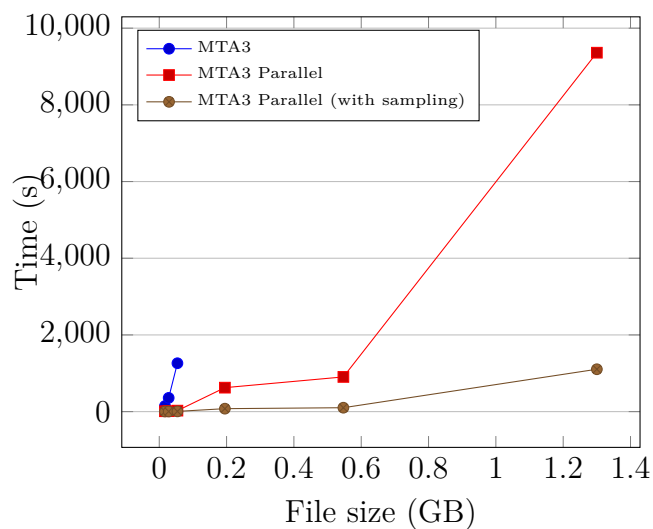


Figure 4.6: A speed comparison of MTA3 and two versions of MTA3 Parallel, with and without sampling. As can be seen, MTA3 shows an early trend of rapidly increasing runtimes and lacks support for files larger than 54 MB.

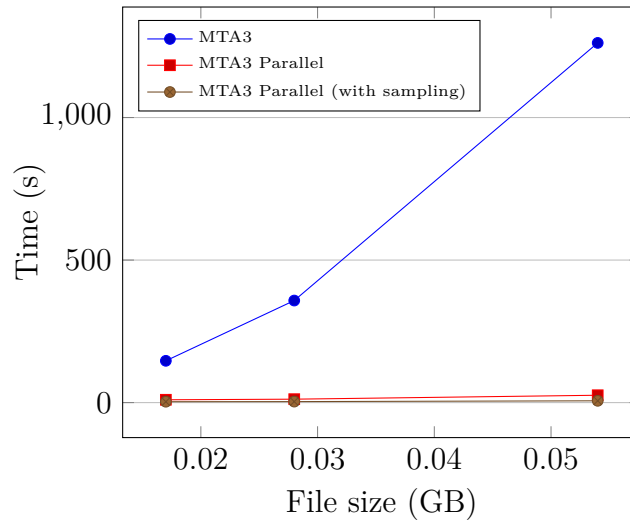


Figure 4.7: A speed comparison of MTA3 and two versions of MTA3 Parallel, with and without sampling, only concerning the file sizes with which MTA3 is compatible.

4.2.2 AIS message validation

Malicious modification of AIS messages is not a new problem and there are many different types, e.g. the top five most common, mentioned in Section 3.2.2. Tampering with any of these can in some sense be seen as decreasing a message's validity. This section covers how this problem was approached, presents a survey of potential solutions and the resulting solution.

Approach

To be able to fit the task of validation into the scope of this thesis, it was decided that the approach should be limited to the issue called identity fraud. During this thesis, the alternatives which were initially considered for detecting identity fraud were:

- Static validation, i.e. checking that the messages conform to the syntax of an AIS message.
- Validation using external data sources, e.g. using information from a port to check if a docked vessel really is the one it says it is.

Static validation was a viable option, but considering that this has been done by the DaMSA in [5], it was discarded. Using external data sources was considered an option and investigated, but since the task was to validate AIS messages, it was apparent that there was a need for a different source of information. After surveying the domain and discussing with domain experts, it was concluded that

data sources that could vouch for the validity of an AIS message were neither openly available nor easily accessible.

After discarding the previous alternatives, an interesting approach was proposed by the supervisor Luis Felipe Sánchez Heres, namely to analyze the movement patterns of vessels. To the knowledge of the authors of this thesis and the supervisor, this had not been explored to this date. The idea is that given the position and time of messages, it is possible to decide which messages that together make up a voyage of a vessel without observing any details like identification number, etc. Then, by using a post processing step, one can recognize potential frauds, for instance where a vessel has changed its identification number.

Literature survey

The task of extracting information for determining a voyage from a vessel's movement pattern could potentially be tackled in many different ways. The approach that was chosen in this thesis was clustering, using the GDBSCAN algorithm described in Section 2.2.1.

There exist many clustering algorithms and before settling on GDBSCAN, a survey of potential candidates with the purpose of getting a good overview of the differences and what would be the most suitable approach was conducted. The clustering methods that showed suitable during the initial survey were K-Means [27], Affinity propagation [28], Spectral clustering [29] and DBSCAN.

When evaluating the clustering algorithms, the following was especially important:

- Vessels have different rates for transmitting messages, resulting in certain areas having a significantly higher concentration of data points. Therefore, the clustering algorithm should handle clusters of varying sizes.
- There is no way to know beforehand how many voyages a given data set consists of and the cluster algorithm should not require the user to supply an estimate of how many clusters the data set holds.
- The rules for deciding whether an item belongs to a cluster depends on specific fields of an AIS message and the algorithm should handle clustering using custom attributes, e.g. time and position.

A widely used method for clustering is K-Means, but since K-Means clusters data into the form of Voronoi cells rather than the shape of a trail and requires an *a priori* estimate of how many clusters the data set consists of, it was determined that K-Means would not yield optimal results.

For the same reasons as K-Means, the Affinity propagation algorithm was also discarded.

Spectral clustering allows for clustering of more general shapes, but has the same requirement of estimating the numbers of clusters in the data set, thus it was also

discarded as a viable alternative.

More promising was the DBSCAN clustering algorithm, partly since the clusters can be of any shape, but also since it naturally clusters points according to their spatial vicinity. Unfortunately, DBSCAN only supports clustering using a distance metric, but there are alternatives which all stem from the DBSCAN algorithm which mitigate this. Among them are the ST-DBSCAN [30] and GDBSCAN algorithms, where the main difference is that the ST-DBSCAN algorithm concerns clustering using spatiotemporal attributes, whereas the GDBSCAN algorithm is the more general variant, allowing for clustering using both spatial and nonspatial attributes as long as the predicate for clustering two items is reflexive and symmetric. With this in mind, the choice naturally fell on GDBSCAN.

Implementation

Since the clustering algorithm only concerns the task of extracting information about which messages that belong to a certain voyage, the complete problem of identifying a possible identity fraud is not solved. To solve this, the strategy was to complement the clustering algorithm with a post processing step which would analyze each cluster and recognize possible identity frauds. Thus, the complete procedure for solving the problem can be summarized as follows:

- Cluster the data according to position and time, where each cluster represents the voyage of a vessel. Since the input data could potentially be collected during a large time period and contain data from high traffic routes, this was done to separate messages from the same position but with different timestamps. The course attribute was used to make sure vessels which travel the same path at the same time but in opposite directions would not be clustered together.
- For each cluster, analyze the messages using a post processing algorithm which looks at the identification number and timestamp of the messages in order to mark specific voyages as possible identity frauds.

When using the GDBSCAN algorithm, there are two main parts that need to be determined. First, to discern between noise and a cluster, the cardinality of a cluster has to be defined. In our case, the minimum number of messages was used as the cardinality measurement. Second, an essential part is to construct a predicate which decides whether two messages belong to the same cluster or not. Since the task in our case was to cluster the voyages of vessels, a predicate which makes use of a vessel's position, time and course had to be constructed. The predicate which decides whether two messages belong to the same neighborhood was constructed using the following attributes:

- The distance which determines if messages are classified as neighbors.
- The time difference which determines if messages are classified as neighbors.

4. Results and discussion

- The course limit which determines if the messages shall be classified as one vessel travelling in one direction or two vessels travelling in different directions.

An implementation of the algorithm can be seen in Algorithm 5. The complete implementation, including the procedure for expanding a cluster can be seen in Appendix A.3.1. A specific issue which became evident was that messages from two vessels travelling in parallel were often clustered together. To mitigate this, an algorithm which analyzed the messages' time stamps was implemented, using a Python version of the *Maximum number of overlapping intervals* algorithm from [31]. The logic behind this was that if two vessels within the same cluster are sending messages more or less at the same time, it is an indication that the messages are from two different voyages. See Algorithm 6 for pseudocode and Appendix A.3.2 for the complete code.

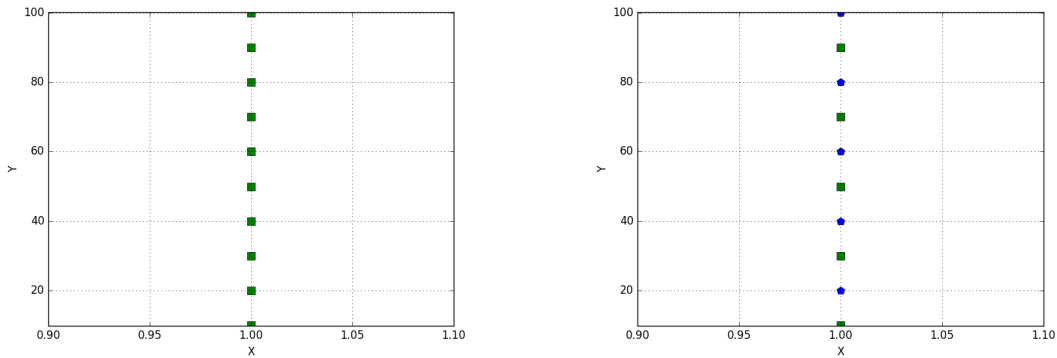
After running the clustering algorithm, the resulting clusters should only contain messages from one vessel. By scanning the results for clusters which consist of different identification numbers, it is possible to recognize potential frauds.

Algorithm 5 GDBSCAN

```
1: function GDBSCAN(points, pred, card, w_card)
2:   cluster  $\leftarrow$  0
3:   for  $p \in$  points do
4:     if  $p.cluster == UNCLASSIFIED$  then
5:       expand  $\leftarrow$  EXPAND_CLUSTER(points,  $p$ , cluster, pred, card, w_card)
6:       if  $expand == true$  then
7:         cluster  $\leftarrow$  cluster + 1
8:   return points
```

Algorithm 6 Validate cluster

```
1: function VALIDATE_CLUSTER(cluster)
2:   mmsis  $\leftarrow$  unique mmsis in cluster
3:   start_times  $\leftarrow$  empty list
4:   end_times  $\leftarrow$  empty list
5:   for  $mmsi \in$  mmsis do
6:     messages  $\leftarrow$  all messages in cluster for mmsi
7:     start_times  $\leftarrow$  start_times ++ EARLIEST_DATE(messages)
8:     end_times  $\leftarrow$  start_times ++ LATEST_DATE(messages)
9:   return LENGTH(mmsis) == MAX_OVERLAP(start_times, end_times)
```



(a) Geographical representation of the input data.

(b) The algorithm correctly separates the input data into two voyages, marked with distinct colors and markers.

Figure 4.8: Clustering of two vessels travelling in opposite directions (head on) using dummy data.

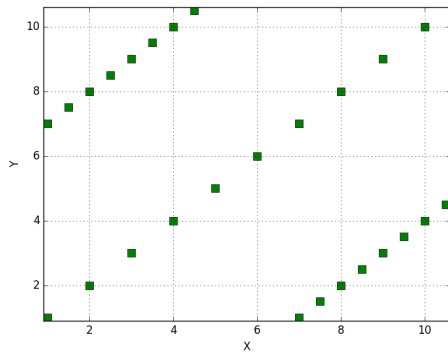
Evaluation

To expose the capabilities and flaws of the algorithm, the strategy was to construct data which aims to model scenarios which could occur in reality. The correctness evaluation of the algorithm was separated into three different parts:

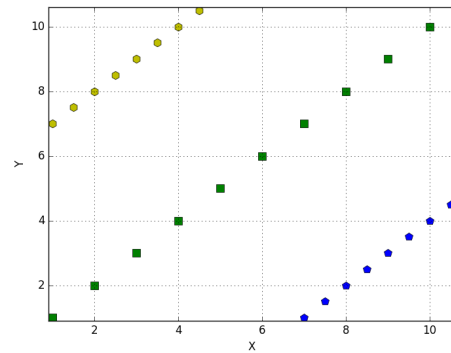
- Ensure that data points are clustered correctly using dummy data.
- Ensure that the post processing algorithm correctly recognizes possible identity frauds in the resulting clusters.
- Ensure that data points are clustered correctly using real data.

For the first part, the following scenarios were used:

- Two vessels travelling during the same time period and in the same area (within the distance threshold), but with opposite directions. This scenario is displayed in Figure 4.8. The correct thing would be to cluster the data into two different voyages. With the input data shown in Figure 4.8a, the results of the clustering algorithm can be seen in Figure 4.8b, where different voyages are marked with distinct colors and markers. By inspection, one can see that the algorithm recognizes the course difference and correctly clusters the input data into two different voyages.
- Data from three vessels, which is displayed in Figure 4.9, should be clustered into three separate voyages, with respect to time, position and course. By observing the results in Figure 4.9b, it is evident that the data has been correctly clustered into three voyages.



(a) Geographical representation of the input data.



(b) Result from clustering where three voyages, marked with distinct colors and markers, have been separated.

Figure 4.9: Clustering of three vessels using a predicate for capturing position and time.

For the second part, the following scenario was used: three vessels are travelling in the same direction with a pairwise distance larger than the clustering threshold. Two of the voyages are valid, while on the third voyage, the vessel has changed its identity number during the trip. The goal of the algorithm is first to cluster the data points into three different voyages, and then to recognize the voyage which contains the identity fraud. Figure 4.9a displays the input data before clustering. In Figure 4.9b, the algorithm has clustered the data into three separate voyages, while in Figure 4.10, one can see that the post processing algorithm has marked two different identity numbers in voyage number two, which is a sign of a possible identity fraud.

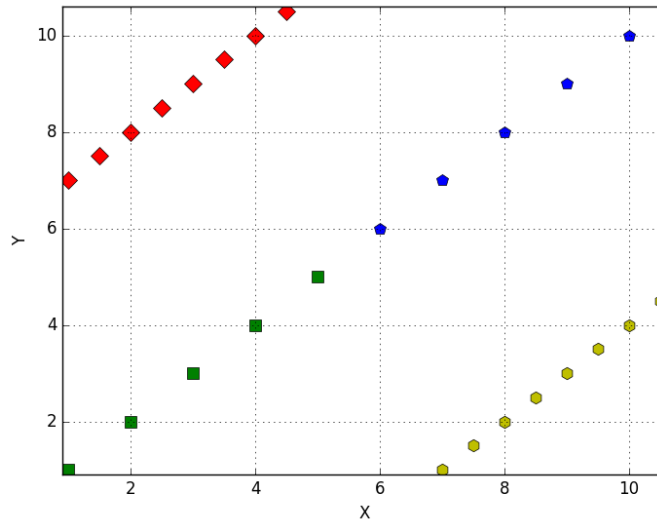


Figure 4.10: Result from the post processing algorithm applied to Figure 4.9b where the voyage in the middle contains two vessels with different identification numbers, thus resulting in an identity fraud. Vessels with different identity numbers are marked with distinct colors and markers.

Finally, the algorithm was tested on a small set of real data. In Figure 4.11, the geographical representation of the AIS data from a fishing boat which is regularly traveling similar paths but at different times, is presented. The clustered data is presented in Figure 4.12, where we can see that the AIS data has been clustered into several voyages, correctly taking into account the difference in time. The different voyages are probably a result of the vessel shutting of the AIS transmitter while being at port, hence starting a new voyage each time it has gone out to sea.

Since identity frauds are rare, the algorithm would likely need to run on large amounts of real data to find such a situation which has occurred in reality. However, since the focus has been on developing an experimental version to investigate the potential of such an algorithm, performance has not been prioritized and with a quadratic time complexity, it is not feasible to run on large sets of real data without optimizing the algorithm.

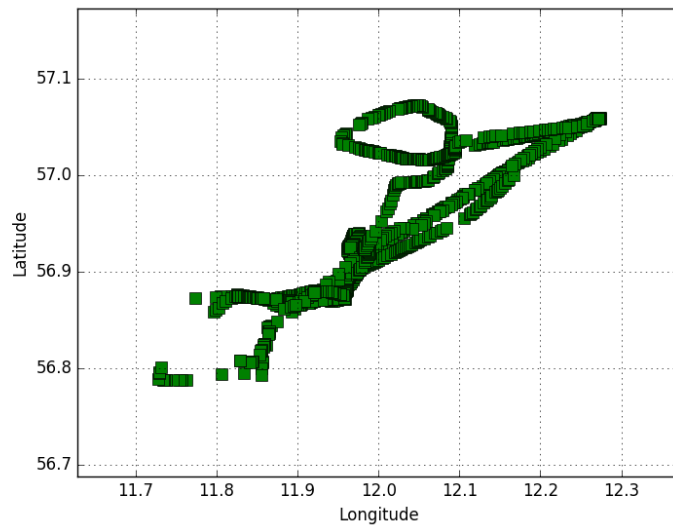


Figure 4.11: Geographical representation of AIS data from a fishing boat travelling similar voyages, but during different time periods.

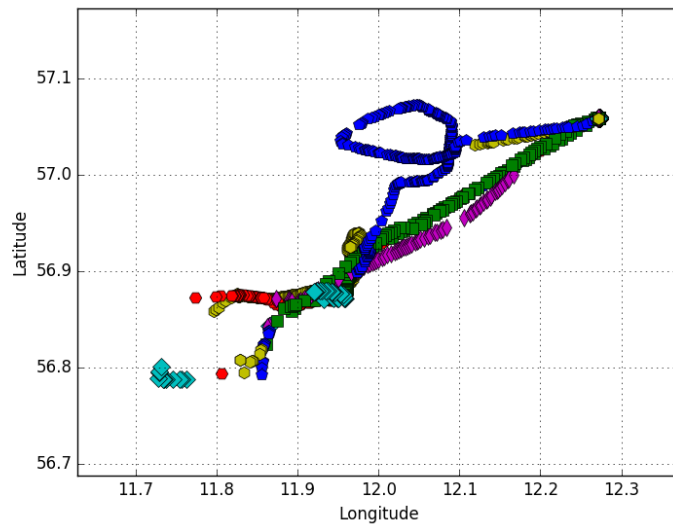


Figure 4.12: Clustered AIS data from a fishing boat travelling similar voyages, but during different time periods. The data is correctly clustered into several voyages using the position, time and course.

Capabilities and limitations

During the experimental implementation, the algorithm has shown promising results when tested on some types of situations. The following covers what the

algorithm correctly handles:

- Handles clustering of AIS data which has been evenly transmitted into voyages.
- Separates voyages consisting of messages from the same time period which have been erroneously clustered together. This could occur when vessels are moving close to each other in parallel.
- Handles clustering of vessels with a large course difference, something that occurs e.g. if two vessels are travelling in the opposite direction (head on).
- Handles clustering of vessels travelling in the same area, with the same course, but during different time periods.

When implementing the solution for clustering voyages, a number of issues surfaced. The following points are the limitations of the algorithm:

- When the data coverage is limited, the algorithm has difficulties. Since the exact borders of AIS areas are unknown, it makes it hard to differentiate between a suspicious situation, i.e. an identity fraud and e.g. a vessel which appears at the border of the AIS coverage area, since both situations look as if a vessel suddenly appeared, whereas only the first situation is fraudulent. This may result in false positives which have to be handled after the clustering phase by manual analysis of the situations, a mundane and time consuming task.
- Difficulties with transmission gaps. Since communication issues arise and the fact that vessels may intentionally disable their AIS transmitter, a gap in the data feed can result in one voyage being clustered into multiple voyages. This issue can be seen in Figure 4.13, where one voyage has been clustered into three voyages due to gaps in the transmission. This is a drawback which has shown to be hard to address, since increasing the neighborhood area in order to allow for gaps in the data feed also means that the accuracy of the clustering algorithm decreases.
- Difficulties with disparate movement patterns of vessels. Since vessels travel at, sometimes significantly, different speeds and have varying maneuver capabilities, the formula for determining suitable parameters for algorithm specific parameters such as neighborhood etc. is not universal. This means that a parameter set may work well when clustering vessels travelling at a low speed, while the same parameter set may be unsuitable for vessels travelling at a higher speed.

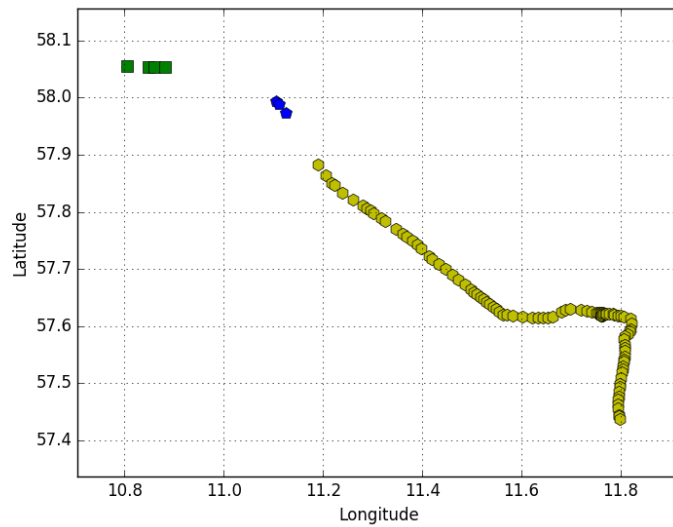


Figure 4.13: The issue of clustering data with irregularities in the transmission rates, where a voyage is incorrectly clustered into three voyages.

4.3 Ethical considerations

With immense amounts of AIS data and advances in the field of Big data, it is now possible to track vessels all over the globe. While this presents a number of interesting applications, the ethical questions connected to this matter should not be overlooked. For instance, if AIS messages can be used to recognize vessels performing illegal fishing in prohibited areas, what would stop e.g. pirates of the 21st century from using the technology to identify potential victims? And what about the sailors? It is not fair if they have to give up their privacy if a system meant to be used for collision avoidance is instead used for surveillance purposes.

Additionally, since the technique is used by companies to track global commodity flows, one can question how appropriate it is to use a technique initially constructed for collision avoidance for such different purposes? What if a company produces a forecast based on commodity flows using AIS data, stating that an increase in the oil price will result in lower sales figures? This may motivate the layoffs of a certain number of employees, while at the same time being based on AIS data, which has no mechanism for validity or security. Consequently, the decision may be based on fake information and it would be unfortunate if the employees would have to pay that price. And should someone carry the burden of ensuring the validity of the data? Should it perhaps be the data analyst who uses the data for support in decision-making or should it be the developers of systems designed for using

AIS data for analytical purposes? These questions may sound far fetched, but in the world of AIS data, described in [1] as “one of the last Wild West frontiers”, questions like these cannot be completely ruled out.

5

Conclusions

The aim of this thesis has been to develop a system for handling and analyzing large sets of marine traffic data, including investigating potential solutions for mitigating the issue of unreliable data. This chapter concludes the outcome of the work carried out in this thesis.

5.1 Development of a large scale data analysis system

Through the study of state of the art Big data techniques and the development of a proof of concept-application, the work during this thesis has shown that it was possible to implement key features from the MTA3 software into the MTA3 Parallel system, with the benefit of increased performance and an accessible user interface. Using the Apache Spark framework and a distributed file system provides the benefit of performance, but also fault tolerance, both against failing computations and data corruption. With regards to the performance, by using spatiotemporal hashing techniques, the optimized version of the initial implementation has shown significant speedups compared to the old MTA3 system, without any kind of sampling. If sampling is used, the performance increase is higher but then the risk of missing interesting situations arises. In terms of the user interface, it provides the ability to anyone with a web browser within the same network to access the underlying computing system, as well as the benefit of hassle-free upgrades.

5.2 AIS message validation

The results from the work concerning AIS validation show some merits when using the developed algorithm for detecting identity fraud situations. The GDBSCAN algorithm for clustering provides an intuitive way to capture movement patterns of vessels in a way that is easily implemented and adapted since it allows for

a predicate using both spatial and nonspatial attributes. In addition, due to the possibility of improving the neighborhood query of the algorithm using appropriate spatial indexing, GDBSCAN should be capable of handling large data sets. However, since movement patterns of vessels vary significantly, constructing a predicate which works well on all types of vessels is challenging. Thus, achieving good performance requires fine tuning of parameters, which may be prohibitively time consuming. Using a post processing algorithm for recognizing identity frauds within the clusters works well on experimental data, but may entail significant complexity when working with real data sets with data from varying areas and when handling false positives.

6

Future work

During this thesis, many interesting possibilities for future improvement of the system have emerged. This chapter describes things that can be improved in the MTA3 Parallel system as well as new areas which may be worthwhile to explore.

6.1 MTA3 Parallel

First, since the focus has been on exploring the potential of using Big data frameworks for AIS analytics, all of the features of the previous MTA3 software have not been implemented. Thus, to complete the transition and completely replace the MTA3 software, it would be a good idea to port the remaining features.

Second, a change which would further strengthen the system would be to move from the basic file based approach where the user uploads a file and then uses that file for analysis, to a system which continuously fetches and dumps AIS data in a data store, allowing for refined querying and increased performance.

Third, another feature would be to expand the system with the possibility of using live data. Apache Spark has a library called Spark Streaming which could make that possible. By using live data the system could directly sieve out invalid data, and perhaps even perform analytics on the fly to increase the performance of future computations.

Fourth, an area which may have a positive impact on performance and usability would be to investigate other scheduling algorithms. As of now, the system uses a basic FIFO queue, which means that running jobs occupy all computational resources of the system. If the number of users of the system increases, it may be worthwhile to look into the more sophisticated scheduling algorithms available in the Apache Spark framework in order to have more fine-grained control.

Finally, with regards to the implementation details of the algorithms, there are a number of areas where improvements can be made. First, since the algorithm for calculating risk of collision uses hashing techniques where the time complexity is dependent on the size of the largest bucket, this in essence means that if all messages are from the same spatiotemporal space, the performance will suffer.

This could be mitigated through the usage of more advanced spatial indexing structures which could balance the input data.

6.2 AIS message validation

A clear drawback of the implementation of the GDBSCAN algorithm for validating AIS messages is its performance. Since the neighborhood has to be calculated for each point, the resulting time complexity is quadratic. By using e.g. spatial indexing structures, the performance of the neighborhood computation could be improved, resulting in a time complexity of $\mathcal{O}(N \log(N))$. In practice, such a solution would make it possible to run the algorithm on larger data sets and make it possible to validate the data before doing further analysis. Furthermore, since the implementation of GDBSCAN is of experimental nature, it has not yet been parallelized. If one would address the drawbacks and conclude that it is worth using for further analysis, it may be beneficial to explore the possibility of parallelizing.

References

- [1] Windward, *AIS Data on the High Seas : An Analysis of the Magnitude and Implications of Growing Data Manipulation at Sea*, <http://www.windward.eu/wp-content/uploads/2015/02/AIS-Data-on-the-High-Seas-Executive-Summary-Windward-October-20-2014.pdf>, [Online; accessed 21-December-2015], 2014.
- [2] A. Harati-Mokhtari, A. Wall, P. Brooks, and J. Wang, “Automatic Identification System (AIS): Data Reliability and Human Error Implications”, *Journal of Navigation*, vol. 60, no. 03, pp. 373–389, 2007.
- [3] D. Winkler, *Enhancing the Reliability of AIS through Vessel Identity Verification*, https://www.fhwa.dot.gov/2015datapalooza/presentations/Safety.4_Winkler.pdf, [Online; accessed 22-December-2015], 2015.
- [4] S. Kazemi, S. Abghari, N. Lavesson, H. Johnson, and P. Ryman, “Open Data for Anomaly Detection in Maritime Surveillance”, *Expert Systems with Applications*, vol. 40, no. 14, pp. 5719–5729, 2013.
- [5] Danish Maritime Safety Administration, *AIS information quality report of static AIS messages: “AIS Information Quality Report” Region : HELCOM*, http://efficiensea.org/files/mainoutputs/wp4/efficiensea_wp4_13.pdf, [Online; accessed 22-December-2015], 2011.
- [6] F. Olinderson, “Development of a software to identify and analyse marine traffic situations”, in *International Conference on Ship Manoeuvrability and Maritime Simulation*, 2015.
- [7] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2012, pp. 2–2.
- [9] A. Gionis, P. Indyk, R. Motwani, *et al.*, “Similarity Search in High Dimensions via Hashing”, in *VLDB*, vol. 99, 1999, pp. 518–529.

- [10] G. M. Morton, *A COMPUTER ORIENTED GEODETIC DATA BASE AND A NEW TECHNIQUE IN FILE SEQUENCING*. International Business Machines Company New York, 1966.
- [11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, in *KDD*, vol. 96, 1996, pp. 226–231.
- [12] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, “Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications”, *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.
- [13] A. S. Szalay and J. A. Blakeley, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, 2nd ed., T. Hey, S. Tansley, and K. Tolle, Eds. Redmond, Washington: Microsoft Research, Oct. 2009, pp. 5–11, ISBN: 978-0-9825442-0-4.
- [14] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities”, in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ACM, 1967, pp. 483–485.
- [15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System”, in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, IEEE, 2010, pp. 1–10.
- [16] Apache Software Foundation, *Apache Spark*, <http://spark.apache.org/>, [Online; accessed 13-May-2016], 2014.
- [17] Apache Software Foundation, *Apache Storm*, <http://storm.apache.org/>, [Online; accessed 13-May-2016], 2014.
- [18] Apache Software Foundation, *Apache Flink*, <https://flink.apache.org/>, [Online; accessed 13-May-2016], 2014.
- [19] Apache Software Foundation, *Apache Hadoop*, <https://hadoop.apache.org/>, [Online; accessed 13-May-2016], 2011.
- [20] Google, *Apache Cloud Dataflow*, <https://cloud.google.com/dataflow/>, [Online; accessed 13-May-2016], 2015.
- [21] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, “The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing”, *Proceedings of the VLDB Endowment*, vol. 8, pp. 1792–1803, 2015.
- [22] Ask Solem, *Celery*, <http://www.celeryproject.org/>, [Online; accessed 1-April-2016], 2007.
- [23] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, “Spatio-temporal Indexing in Non-relational Distributed Databases”, in *Big Data, 2013 IEEE International Conference on Big Data*, IEEE, 2013, pp. 291–299.

-
- [24] Gustavo Niemeyer, *geohash*, <http://geohash.org>, [Online; accessed 1-April-2016], 2008.
- [25] Leonard Norrgard, *python-geohash*, <https://github.com/vinsci/geohash/>, [Online; accessed 31-March-2016], 2015.
- [26] Abe Usher, *timehash*, <https://github.com/abeusher/timehash>, [Online; accessed 31-March-2016], 2014.
- [27] J. A. Hartigan and M. A. Wong, “Algorithm AS 136: A K-Means Clustering Algorithm”, *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.
- [28] B. J. Frey and D. Dueck, “Clustering by Passing Messages Between Data Points”, *Science*, vol. 315, no. 5814, pp. 972–976, 2007.
- [29] U. Von Luxburg, “A tutorial on spectral clustering”, *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.
- [30] D. Birant and A. Kut, “ST-DBSCAN: An algorithm for clustering spatial-temporal data”, *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.
- [31] Zahid, *Maximum number of overlapping intervals*, <http://www.zrzahid.com/maximum-number-of-overlapping-intervals/>, [Online; accessed 10-May-2016], 2015.

A

Implementation of algorithms

This appendix includes Python implementations of the algorithms used in this thesis.

A.1 Ship handling algorithms

A.1.1 Risk of collision

```
def risk_of_collision(cpa_and_tcpa, cpa_limit, tcpa_limit):
    date1 = cpa_and_tcpa[0].date
    date2 = cpa_and_tcpa[1].date

    if date1 >= date2:
        in_risk = (date1 - date2) <= timedelta(minutes=2)
    else:
        in_risk = (date2 - date1) <= timedelta(minutes=2)

    if cpa_and_tcpa[2]:
        cpa = cpa_and_tcpa[2][0]
        tcpa = cpa_and_tcpa[2][1]
        return all([in_risk,
                    0 <= cpa <= cpa_limit,
                    0 <= tcpa <= tcpa_limit])
```

A.1.2 Closest point of approach and time to closest point of approach

```
def cpa_and_tcpa(m1, m2):
    rel_velocity = np.subtract(velocity(m2),
                               velocity(m1))
```

```

x = rel_velocity[0]
y = rel_velocity[1]

if x != 0 or y != 0:
    (heading, speed) = course_speed_from_vector(rel_velocity)
    bearing = calc_bearing(m1, m2)
    if bearing is not None:
        rel_bearing = calc_rel_bearing(bearing + 180,
                                       heading)

        dist = distance(m1, m2)
        return (dist * math.sin(abs(rel_bearing) * TO_RAD),
                dist * math.cos(abs(rel_bearing) * TO_RAD) / speed)
    else:
        return None

```

A.1.3 Risk of collision (parallelized)

```

def calc_risk_of_collision(vessels):
    cpa_and_tcpa = vessels.cartesian(vessels)
        .filter(lambda vs: vs[0].mmsi != vs[1].mmsi)
        .map(lambda vs: (vs[0], vs[1],
                        cpa_and_tcpa(vs[0], vs[1])))
    return cpa_and_tcpa.filter(lambda x: risk_of_collision(x))

```

A.1.4 Risk of collision (parallelized and optimized)

```

def calc_risk_of_collision(vessels, cpa_limit, tcpa_limit):
    hashed = vessels.flatMap(lambda x: buckets(x))
    grouped = hashed.reduceByKey(lambda x, y: x + y)
    cartesian = grouped.flatMapValues(lambda x: itertools.product(x, x))
    vs = cartesian.filter(lambda vs: vs[1][0].mmsi != vs[1][1].mmsi)
    cpa_tcpa = vs.map(lambda x: (x[1][0],
                                x[1][1],
                                cpa_and_tcpa(x[1][0], x[1][1])))
    rocs = cpa_tcpa.filter(lambda x: risk_of_collision(x,
                                                       cpa_limit,
                                                       tcpa_limit))

    return rocs

```

A.2 Temporal hashing

A.2.1 Contributions to the timehash [26] package

```
def after(hashcode):
    i = 1
    for c in reversed(hashcode):
        padding = (i - 1) * '0'
        pos = len(hashcode) - i
        if c != 'f':
            ret = hashcode[:pos] + __neighbormap[c][1] + padding
            return ret
        else:
            i += 1

def before(hashcode):
    i = 1
    for c in reversed(hashcode):
        padding = (i - 1) * 'f'
        pos = len(hashcode) - i
        if c != '0':
            ret = hashcode[:pos] + __neighbormap[c][0] + padding
            return ret
        else:
            i += 1

def neighbors(hashcode):
    return [before(hashcode), after(hashcode)]

def expand(hashcode):
    return [before(hashcode), hashcode, after(hashcode)]
```

A.3 Clustering

A.3.1 GDBSCAN

```
UNCLASSIFIED = -2
```

```
NOISE = -1
```

```
def GDBSCAN(points, n_pred, min_card, w_card):
    cluster_id = 0
    for point in points:
        if point.cluster_id == UNCLASSIFIED:
            if _expand_cluster(points, point, cluster_id, n_pred, min_card,
                               w_card):
                cluster_id = cluster_id + 1
    clusters = {}
    for point in points:
        key = point.cluster_id
        if key in clusters:
            clusters[key].append(point)
        else:
            clusters[key] = [point]
    return list(clusters.itervalues())

def _expand_cluster(points, point, cluster_id, n_pred, min_card, w_card):
    if not _in_selection(w_card, point):
        points.change_cluster_id(point, UNCLASSIFIED)
        return False

    seeds = points.neighborhood(point, n_pred)
    if not _core_point(w_card, min_card, seeds):
        points.change_cluster_id(point, NOISE)
        return False

    points.change_cluster_ids(seeds, cluster_id)
    seeds.remove(point)

    while len(seeds) > 0:
        current_point = seeds[0]
        result = points.neighborhood(current_point, n_pred)
```



```
    if w_card(result) >= min_card:
        for p in result:
            if w_card([p]) > 0 and p.cluster_id in [UNCLASSIFIED, NOISE]:
                if p.cluster_id == UNCLASSIFIED:
                    seeds.append(p)
                    points.change_cluster_id(p, cluster_id)
        seeds.remove(current_point)
    return True

def _in_selection(w_card, point):
    return w_card([point]) > 0

def _core_point(w_card, min_card, points):
    return w_card(points) >= min_card

class Points:
    def __init__(self, points):
        self.points = points

    def __iter__(self):
        for point in self.points:
            yield point

    def __repr__(self):
        return str(self.points)

    def get(self, index):
        return self.points[index]

    def neighborhood(self, point, n_pred):
        return filter(lambda x: n_pred(point, x), self.points)

    def change_cluster_ids(self, points, value):
        for point in points:
            self.change_cluster_id(point, value)

    def change_cluster_id(self, point, value):
        index = (self.points).index(point)
```

```
        self.points[index].cluster_id = value

def labels(self):
    return set(map(lambda x: x.cluster_id, self.points))

class Point:
    def __init__(self, x, y, mmsi, date, cog):
        self.x = x
        self.y = y
        self.mmsi = mmsi
        self.date = date
        self.cog = cog
        self.cluster_id = UNCLASSIFIED

    def __repr__(self):
        return '%s, %s, %s, %s, %s, %s' % (self.x, self.y, self.date,
                                           self.mmsi, self.cog,
                                           self.cluster_id)

def n_pred(p1, p2):
    return all([within_distance(p1, p2, 1),
               abs(p1.date - p2.date) <= timedelta(minutes=20)])
```

A.3.2 Post processing algorithm for validating a cluster

```
def validate(list_of_clusters):
    return [c for c in list_of_clusters if valid_cluster(c) is False]

def valid_cluster(cluster):
    mmsis = set([p.mmsi for p in cluster])
    nr_of_vessels = len(mmsis)
    start_times = []
    end_times = []
    for mmsi in mmsis:
        vs = [x for x in cluster if x.mmsi == mmsi]
        start_times.append(min([x.date for x in vs]))
        end_times.append(max([x.date for x in vs]))
    return nr_of_vessels == max_overlap(start_times, end_times)

def max_overlap(start, end):
    overlaps = 0
    current_overlap = 0
    sorted_start = sorted(start)
    sorted_end = sorted(end)
    i = 0
    j = 0
    m = len(sorted_start)
    n = len(sorted_end)
    while(i < m and j < n):
        if sorted_start[i] < sorted_end[j]:
            current_overlap += 1
            overlaps = max(overlaps, current_overlap)
            i += 1
        else:
            current_overlap -= 1
            j += 1
    return overlaps
```