A SPATIO-TEMPORAL DATA REPRESENTATION FRAMEWORK WITH APPLICATIONS TO ANOMALY DETECTION IN THE MARITIME DOMAIN

by

Vladimir Radu Avram

B.Sc. (Computer Science, Microbiology and Immunology),
University of British Columbia, 2010

a Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

APPROVAL

Name:	Vladimir Radu Avram
Degree:	Master of Science
Title of Thesis:	A Spatio-temporal Data Representation Framework With Applications to Anomaly Detection in the Maritime Domain

Examining Committee: Dr. Veronica Dahl
Chair

Dr. Uwe Glässer, Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Peter Borwein, Professor, Mathematics
Simon Fraser University
Co-Supervisor

Dr. Vahid Dabbaghian, Adjunct Professor, Mathematics
SFU Examiner

Date Approved:

Abstract

Maritime security and disaster response are critical for many nations to address the vulnerability of their sea lanes, ports, and harbours to a variety of illegal activities as well as to prevent, minimize, and respond to environmental and human disasters. With increasing volume of spatio-temporal data available from systems like the Automatic Identification System, satellite, marine radar, and other sources it is increasingly problematic to analyze this enormous volume of data. This work builds on the state-of-the art in spatio-temporal anomaly detection by proposing a representation framework for spatio-temporal data, providing a software implementation as an API, and evaluating it. The aim of the framework is to represent spatio-temporal data using abstract concepts to describe motion over time in order to concisely represent high level abstract motion behaviours. The framework is generic and can be applied to any type of spatio-temporal data, but the focus is on the maritime domain.

Acknowledgments

I would like to thank my family, my girlfriend, and my friends for their love, support, and constant encouragement.

I would also like to thank my supervisors, Drs. Glässer and Borwein for their guidance and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Maritime security and disaster response are critical for Canada and the United States, among many other nations, to address the vulnerability of their sea lanes, ports and harbours to a variety of threats and illegal activities as well as to prevent, minimize, and respond to environmental and human disasters. Examples of many such threats and disasters can be seen throughout the world. Activities such as piracy which is especially problematic in the Gulf of Aden, where in 2009 alone, 163 attacks on ships were carried out and 47 ships and their crews were taken hostage [4]. Illegal activities like human [20] and drug trafficking. Environmental disasters such as oil spills, of which there have been 410 large ones (greater than 700 tonnes) since 1960, adding up to a total of over 5.5 million tonnes of oil spilt into the oceans [28].

Surveillance can be challenging in areas with high density and volume of marine traffic. The massive size of the areas that must be monitored only serves to compound the problem; Canada's coastlines alone are over 243,000 Km in length [5]. With increasing volume of spatio-temporal data available from systems like the Automatic Identification System (AIS), satellite, marine radar, and other sources it is ever more problematic to analyze such data in real time so as to enable a swift response to threats, illegal activities, and disasters by border control agencies, coast guard services, and military forces. Eventually the volume of data will exceed the capacity of the human resources available to analyze it. This situation calls for the use of intelligent systems to assist the necessary personnel in this analysis and the decision making that naturally accompanies it.

Systems specifically designed to deal with the complex task of assisting human operators with analyzing large amounts of data and performing timely and informed decisions fall into the category of Decision Support Systems (DSS); for a great overview of DSS technology see [26]. More specifically, the type of system that would be ideal for dealing with this problem is a type of DSS for the purpose of Situation Analysis (SA), a Situation Analysis Decision Support (SADS) system. Decision support systems for situation analysis can be very complex, monitoring numerous resources, distributed computing agents, and events distributed in space and time. Situation analysis is viewed as a process to provide and maintain a state of situation awareness for the decision maker. Situation awareness is essential for decision-making activities; it is about our perception of the elements in the environment, the comprehension of their meaning, and the projection of their status in the near future [8]. Another way to view situation awareness is as a state of mind maintained by the process of situation analysis; it is a product that is maintained and generated by situation analysis. Experienced decision makers rely on situation awareness in order to make informed decisions.

In order to successfully deal with the problem of supporting the analysis of the vast historical and real-time data generated in the maritime domain an SADS system's main role is to provide and maintain a state of situation awareness by automating as much of the task of situation analysis as possible. Realistic situation analysis scenarios routinely deal with convoluted and intricate event patterns and interdependencies to interpret and reason about complex situations, assess risks, and predict how a situation may evolve over time. These features are definitely present when dealing with the maritime domain. There are many aspects of the maritime domain that could benefit from the combination of experienced decision makers and SADS systems. However, the focus of this work will be on the particular area of Anomaly Detection (AD). Anomaly detection can broadly be described as a method for detecting any abnormal behaviour of interest that points to a safety or security threat. AD touches on all of the interesting topics of safety and security discussed at the beginning of this introduction.

Designing and building an SADS system for anomaly detection would be an enormous task in the maritime domain; such systems are usually composed of the following components: 1) situation information collection (e.g., sensors and sensors networks), 2) knowledge base on the situation, 3) reasoning scheme (engine) about the situation, 4) human-computer interface, and 5) controls; see figure 1.1. Because each of these systems in their own right

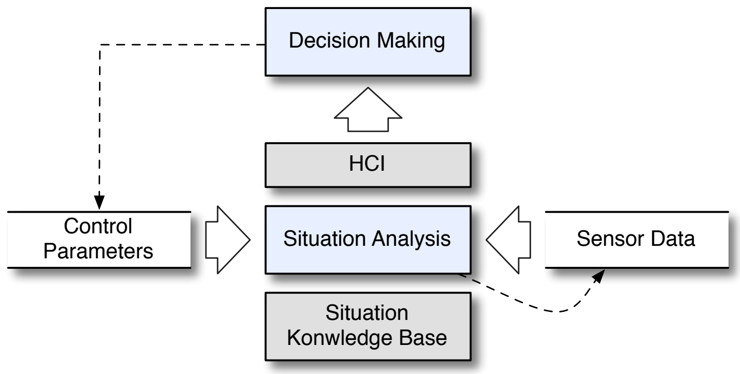


Figure 1.1: Situation Analysis and Decision Support

warrant time and research efforts which are beyond the scope of this work our area of focus will be on component number 3, the reasoning engine, specifically as it applies to analyzing the motion behaviour of vessels as it relates to the problem of AD.

This direction of research was brought about by initial work we undertook on design and analysis of SADS systems.

1.2 Anomaly Detection: Problem Overview

Intuitively, anomalies are outliers in data that do not conform to expected or normal behaviour. A more statistically oriented definition of an anomaly is given by D. Hawkins [12]: " *an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.*" Anomaly detection must be viewed in the context of large spatio-temporal datasets generated from monitoring marine traffic as this is the source of the majority of the data used for analysis.

Given some dataset, the problem of anomaly detection is that of separating a small minority of anomalous data, in our case anomalous behaviour, from the large majority of data, again in our case normal behaviour. Furthermore, there is also the need to separate suspicious anomalous behaviour (any behaviour that points to a potential safety or security threat) as anomalies of interest from the set of anomalies considered irrelevant to maritime safety and security. The main idea in approaching this problem is to separate and characterize the majority of data as normal behaviour and to utilize this as a way to identify anomalies as abnormal behaviour.

The design of reliable algorithms to detect anomalous behaviour indicating a potential threat or illegal activity is a challenging engineering task due to a number of adverse factors: the complex and variable nature of observed behaviours, the enormous volume of marine traffic, and dynamically changing and often unpredictable factors in the physical environment. Additional challenges that any approach commonly faces when attempting to tackle the problem of anomalous behaviour detection include:

- Complex and variable nature of observed behaviours, the enormous volume of marine traffic, and dynamically changing and often unpredictable factors in the physical environment.
- One of the greatest challenges that any anomaly detection system must face is reducing false alarms while maintaining maximum sensitivity to suspicious behaviour.
- *Considerable variety in observed behaviours*: Clearly no two observed motion patterns will be exactly identical in time and space; so the system will necessarily have to be able to deal with fuzzy concepts of similarity.
- *Uninteresting anomalous behaviour*: Often a behavioural pattern will be identified as anomalous, although it may be of no interest to those performing the surveillance and may easily just be a trivial anomaly.

In principle there are two basically different methods to anomaly detection: *data-driven* and *model-driven*; which we will refer to interchangeably as *top-down*, and *bottom-up*. Each one comes with its own pros and cons, thus giving rise to hybrid approaches that combine both data-driven and model-driven methods. Data driven-methods can potentially identify anything anomalous, although most of it may be irrelevant, whereas, model-driven methods focus on specific patterns only but miss everything else.

We have explored existing approaches to anomaly detection in the maritime domain and have identified the following key areas of improvement:

- The *feature set* used when analyzing the data.
- The *representation* of the data.
- The *combination* of model-driven and data-driven approaches.

This work will touch on all of the key areas of improvement. Feature sets for analysis and spatio-temporal data representation are the main areas where a contribution will be made in order to address some of the challenges posed by the AD problem. We propose a novel method for representing spatio-temporal data which leads to an expansion in the feature set available to analyze the data; a necessity, at least in the maritime domain, for improving current AD methodology. This proposed representation applies to any type of spatio-temporal data, however it will be primarily evaluated in the context of ship voyages.

1.3 Contributions

The AD problem is large and multi-faceted and there are no easy solutions to it so we do not claim any such thing. Our goals are more humble and we instead attempt to make some headway in a new direction to dealing with some portions of the AD problem. As briefly mentioned in the introduction the main goals of this thesis are to address some of the main challenges posed by the AD problem. The major way in which we try to address these problems is to propose a representation framework for representing quantitative time series data. In addition to conceptually and formally defining the representation framework we also design and implement it as a software application programming interface (API) in order to facilitate experimentation and exploration. Once the software API is built we attempt to show the robustness of the representation in dealing with the variety of observed patterns in trajectories. Finally, we attempt to demonstrate how the representation framework could be used to detect interesting and complex behaviour in a ship voyage.

As a result of this work and other work on SADS which lead up to this thesis the following research papers have been produced and successfully accepted in a variety of venues:

- Anomaly Detection in Spatiotemporal Data in the Maritime Domain.
  Vladimir Avram, Uwe Glässer, and Hamed Yaghoubi Shahir [3]

- A Formal Engineering Approach to High-Level Design of Situation Analysis Decision Support Systems. Roozbeh Farahbod, Vladimir Avram, Uwe Glässer, Adel Guitouni [10]

- Engineering Situation Analysis Decision Support Systems. Roozbeh Farahbod, Vladimir Avram, Uwe Glässer, Adel Guitouni [9]

In summary the main contributions of this work are:

- A representation framework for representing quantitative time series data, applied more specifically to spatio-temporal data.
- A software implementation of the proposed framework as an API.
- Experimental demonstration of the robustness of the proposed framework.

1.4 Summary of Results and Conclusions

1.5 Overview of Remaining Chapters

First we will discuss related work in chapter 2. We will give an overview of the existing AD methods in the field as well as the main categories they fall under. Moving on to chapter 3 we will spend some time discussing the necessary background information and definitions related to the problem as well as introduce the problem of AD, and discuss some of the details. Chapter 4 will be dedicated to conceptually and formally defining the proposed framework as well as detailing the main algorithms which are then implemented in the AnoDeC API. The chapter is ended with an extensive example of the application of all the presented ideas and algorithms to the detection and characterization of an interesting and complex "shuttling" behaviour in a ship voyage. Chapter 6 presents experiments to show the robustness of the proposed representation, the effectiveness of ϵ-generalization in preserving features and compressing data, and the effectiveness of smoothing coupled with generalization in extracting features from noisy data. Finally the last chapter 7 will conclude the thesis with a short overview of the work and results as well as future directions for this line of research.

Chapter 2

Related Work

This chapter will provide an overview of the different major types of anomaly detection techniques used in the maritime domain. Additionally it will also cover the current state-of-the-art in maritime anomaly detection by summarizing some selected papers for each major category.

2.1 Anomaly Detection in the Maritime Domain

2.1.1 High Level Overview

In this section we will look at a high level overview of the current approaches used for the anomaly detection problem in the maritime domain. An excellent paper [18] that gives a thorough description of this has been published very recently by Defense Research & Development Canada (DRDC)

Existing work on anomaly detection in the maritime security domain currently falls into two main categories, *data-driven* (bottom-up) and *model-driven* (top-down) approaches. Additionally, *hybrid* (mixed) approaches may be considered a third category (see Figure 2.1). You will also notice that after reading the reviews and spending some time going through the extra referenced material in the bibliography that the majority of research focuses exclusively on the detection part and not the characterization or identification part of the problem. Furthermore, you will also see that most of the work also focuses on bottom-up approaches with the ubiquitous four dimensional feature space of position(x, y) and velocity(x, y).

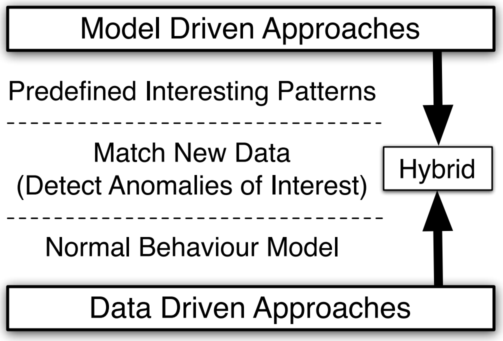


Figure 2.1: Major types of approaches for anomaly detection.

2.1.2 Data-Driven Approaches

The idea behind most data-driven approaches is comprised of two steps: *1)* build a model of the normal behaviour as expressed by the data, *2)* match new data to the existing model of normal behaviour treating any deviations (over some threshold) as anomalies. Three types of methods are included in this category as follows:

- Gaussian Mixture Models (GMM)
- Kernel Density Estimation (KDE)
- Bayesian Networks (BN)

We will give an overview of each method and discuss it in some detail by exploring some of the latest work published in that area.

Gaussian Mixture Models and Kernel Density Estimation

- *Gaussian Mixture Model (GMM)*
  GMMs can be thought of as methods for clustering or unsupervised learning [27]. They are a mixture or a linear combination of multiple multivariate Gaussian Probability Density Functions (PDFs). Stated another way, we can look at a GMM as a probabilistic model with a density function that is just a combination of a number of other PDFs, in this case Gaussian ones. It provides a probabilistic model of the data which is multi-modal and which allows for soft clustering boundaries, meaning that an observation can belong to any class with some probability. In order to build such a model,

an appropriate set of training data and an algorithm (e.g., Expectation-Maximization (EM) algorithm) for fitting the model to the data are required.

- *Kernel Density Estimation*
  A KDE is a non-parametric density estimator, and its aim is to estimate a PDF. Since a KDE does not depend on parameters to describe a PDF, it has no fixed structure and depends on all data observations in order to achieve an estimate [11].

One of the papers taking a statistical approach [15] proposes and implements unsupervised clustering of normal vessel traffic patterns. The cluster model is a Gaussian Mixture Model and the clustering algorithm is a greedy version of Expectation-Maximization. The study makes use of real data to train and evaluate the models. The author points out that if a large amount of data corresponding a model of routine behaviour is available, the model is easier to build than characterizing a priori all anomalies of interest. Once the model of normal behaviour is built the task of finding anomalies is just the act of calculating the likelihood of a new observation belonging to the distribution describing normal behaviour.

With regards to experimental validation, the author points out that there is no well established approach to evaluate a system for anomaly detection in the maritime domain. Also there is no common benchmark, no set of well defined anomalous maritime scenarios. Furthermore, through qualitative analysis, it was found that the anomalies detected in the sea traffic data are vessels crossing sea lanes and vessels traveling close to and in the opposite direction of sea lanes. The author points out that these are not very interesting anomalies and goes on to state that in order to detect more complex anomalies involving multiple vessels, or behaviour that develops over time a more sophisticated feature model is necessary. In conclusion the paper arrives at the point that this work tries to emphasize as well, and that is: “the type of feature model essentially determines the character of the detectable anomalies” [15]

Another paper [16] dealing with statistical approaches to anomaly detection which is co-authored by Rikard Laxhammar who also wrote the first paper discussed in this section [15] compares Gaussian Mixture Models and Kernel Density Estimators. The paper evaluates the GMM and KDE methods for statistical anomaly detection in sea traffic which is characterized using a four dimensional feature space: location(latitude, longitude) and velocity(latitude, longitude). The general approach is similar to that proposed in [15] and is based on modelling normal behaviour as a probability density function and then computing the likelihood that a particular observation was generated by the normal PDF. The authors also propose a novel performance measure for evaluating anomaly detection performance as well as a method for measuring normalcy modelling performance. These measures were then utilized in comparing the two methods of interest.

In conclusion the paper finds that even though the KDE method was initially superior with respect to normalcy modelling, the anomaly detection experiments showed no significant difference between the KDE and the GMM method. The authors even go as far as stating that the anomaly detection results were “disappointing”. Both methods are found to be suboptimal; according to the authors this is partially because the data is artificially divided into cells for the purpose of reducing computational burden.

Another paper written by two of the same authors, Laxhammar R. and Falkman G., who co-authored [16] continues that work on anomaly detection. The paper [17] presents a novel application of conformal prediction for distribution-independent on-line learning and anomaly detection. The authors point out some issues that motivate and steer our work as well. They make the critique that parametric statistical methods assume the existence of a parameterized model which can actually be estimated accurately. Furthermore, they also point out that even though this issue can be avoided by using non-parametric methods like KDE such methods require a larger amount of data in order to produce reliable and accurate estimates of the underlying PDF, and it has already been shown by the authors in the their previous work that both the parametric GMM method and the non-parametric KDE method produce “disappointing” results [16]. They go on to point out that most previously proposed algorithms for anomaly detection operate in a two step off-line mode, where first a model is learned from data and then used for anomaly detection in new data, this property is also true for most of the research reviewed in this chapter.

The proposed approach performs learning on-line and makes no assumptions about underlying distributions other than the assumption that normal behaviour of vessels follows a fixed but unknown probability distribution. It also requires few parameters to be set as well as no requirement for application specific thresholds. The way in which they aim to detect anomalies is by predicting a set of plausible classes (container ship, fishing boat, etc.) for a vessel based on its current velocity and position. If this is not consistent with the reported vessel class (as reported by the vessel itself) the vessel will be flagged as anomalous. In the case that no classes seem plausible this is an indication of unrecognized behaviour, or in the case that only classes other than the observed classes seem plausible this is taken to be an indication of recognized but unexpected behaviour.

The main idea behind conformal prediction is described in the paper with a small example: Given some specified confidence level, 95%, a conformal predictor would produce a prediction set that contains the true label with a probability of at least 95%. The most important feature of these predictors is that successive predictions made will be correct with a probability equal to or larger than the corresponding confidence level even though the subsequent predictions are based on an accumulating dataset (due to on-line learning). I will not go into the details of how the conformal predictors are built and used; all the necessary details can be found in the paper [17].

Looking at the results they seem quite good and are described as superior to the GMM approach and slightly better than the KDE approach. They even go on to say that most anomalous trajectories are very easy to detect. However, they limit the trajectories in their experimental data set to three vessel classes: cargo ship, tanker and passenger ship. These classes of vessels exhibit rather limited behaviour and strictly follow sea lanes. This is probably why detecting anomalies seemed so easy. Finally, as future work the authors identify the need for motion representations that capture behaviour over time and suggest using "sequences of speed and turn rate values represented as high-dimensional data vectors or motion abstractions that compress behaviour over time" which is very much what the work presented in this document tries to focus on.

Another paper [24] applying a statistical approach to the analysis of AIS data for anomalous vessel detection actually presents a solution to motion prediction as opposed to detection of currently observed behaviour. This is all done in the framework of KDE with particle filtering for the motion prediction. There are some interesting ideas in this paper specifically with regard to motion prediction, however the example presented is rather simple and only covers one motion pattern even though it seems that the authors had quite a lot of real data available. Furthermore, the data they pick to experiment on seems much too clean to even begin to test the robustness of the approach. However it is still worth looking at because of the interesting ideas with respect to motion prediction.

Both GMM and KDE are found to be suboptimal in the existing literature. One reason is because the existing methods artificially divide the data into cells by imposing a grid on the geographical area in question. This is done because of computational resource constraints. Furthermore, the two methods are based on a four dimensional feature model composed of momentary location(x and y) as well as velocity(vx and vy).

As indicated by [15] the type of feature model essentially determines the character of the detectable anomalies. A four feature model as employed by the approaches using GMM and KDE is able to detect simple anomalies such as vessels crossing sea lanes or vessels travelling in the opposite direction of sea lane traffic flow, but it can not identify any specific patterns. Consequently because of the four feature model chosen for the two techniques, they are only able to detect anomalies that can be represented by the features, such as simple spatial anomalies and velocity anomalies. For example they can identify if a vessel is travelling too fast or too slow or if it is in a zone that is not normally travelled. These methods can not detect anything more sophisticated than this, definitely no abstract behaviour.

Another issue with using purely statistical approaches for anomaly detection in the maritime domain is that the identified anomalies are never characterized. The result will be a collection of data that represents behaviour different enough from the norm to be deemed anomalous, however, what that behaviour is, is never established. It is possible that behaviour that is not anomalous from a human operators point of view is detected as anomalous simply because the data set used to build a model of normal behaviour did not have this particular behaviour in it. Furthermore it is also possible that behaviour that is not interesting at all is flagged as anomalous. This is a result of the way the techniques work: they can detect anomalies, but not what behaviour those anomalies may represent so they can not differentiate between interesting and trivial anomalies.

2.1.3 Bayesian Network Approaches

There seems to be less research on anomaly detection in the maritime domain focusing on BNs; save for one recent paper [19]. The research deals with anomaly detection using BNs at two different time scales: *1)* moment to moment, and *2)* the entire trajectory. The networks also incorporate additional data like information related to the ship (e.g., type, dimensions, and weight), weather (e.g., temperature, cloud cover, and wind speed), and temporal factors (e.g., hour of day and time since dawn or dusk). The general approach is the common one found in most research concerning anomaly detection in the maritime domain; the two-step approach of a training step to generate a model of normalcy and then the use of that model to estimate the probability that new data belongs to it. There are some deviations from this approach based on the analysis of data at the two mentioned time scales. The first time scale is concerned with moment to moment observations (referred to as “time series”). Each time step in a trajectory has each of its variables (e.g., latitude, longitude, speed, etc.) associated with a node in the BN. The second time scale is concerned with the whole trajectory assessment (referred to as "track summary") and takes into account number of stops, stopping locations, etc. Finally the BNs for both levels of abstraction are trained and evaluated. Findings show that neither model performs better in all cases but there is an improvement to be gained in a variety of cases by combining the models.

2.1.4 Model-Driven Approaches

Model-driven approaches, like data-driven approaches, are also made up of two main steps: *1)* create and specify predefined patterns that you want to detect using rules (predicates or any type of logic), *2)* try to match new data to the predefined rules in order to detect predefined anomalies of interest.

Not a lot of work is focusing on model-driven approaches. Some work from DRDC [25] focuses on knowledge-based approaches. It describes how to categorize anomalies and identifies a taxonomy of kinematic and geospatial concepts, though interesting and useful, as of yet there are no experiments or results described. The authors indicate that the work is still in progress, but the preliminary work does look promising [25].

Even though compared to the diversity of existing *data-driven* anomaly detection methods the number of methods using *model-driven* approaches is limited, this direction appears to be particularly promising for:

- Characterizing anomalies, possibly those already identified by the data-driven approaches.
- Detecting pre-defined anomalies of interest.

2.1.5 Hybrid Approaches

There has been some very interesting work done [14] that combines Bayesian network techniques with specific maritime domain modelling. The authors present some initial work on algorithms for calculating the probability that any one of five specific anomalies is present in ship AIS data. The prediction models of the five anomalous behaviours were tested on real data and the results are quite encouraging. The second main focus of this work was on a general Bayesian network-based method for taking the individual anomalies and determining the probability of a higher-level threat which was demonstrated using simulated data.

The five anomalous behaviours in question are: deviation from standard routes, unexpected AIS activity, unexpected port arrival, close approach, and zone entry.

The analysis of the first anomaly, which is deviation from standard routes, begins by modelling ship tracks. The authors mention GMM and KDE as previously used for this modelling. However they focus on a different method using a network of discrete nodes placed at decision points that are related to constraints and branches to connect the nodes, landmasses are also represented as closed polygons. This generates a sparse network model which is branch rich and node light. This is very important because the speed of optimal route algorithms is dominated by the number of nodes not branches. Once the network is built Dijkstra's algorithm is used to compute optimal routes and costs between ports. The actual detection approach deals with ships journeys which begin and end in ports as well as partial journeys which are broken down into macro (all branches traversed before the previous network node) and micro (route since the previous network node). The algorithm calculates for every possible destination the conditional (on the macro and micro parts) probability that it is in fact the true destination. "These probabilities can be used to assess the following two hypotheses: H_0, the ship is travelling to its stated destination; and H_1, the ship is travelling somewhere else. Implicit in the null hypothesis H_0 is the assumption that if a ship is travelling to its stated destination, then it will do so by a route that does not incur unreasonable cost." [14]. Finally the authors successfully demonstrate the main goal of the algorithm which is to detect ships that do not follow defined routes.

The second anomaly of interest is unexpected AIS activity. The main idea behind this is that if there is a long period of AIS silence in an area that has good AIS coverage there is the possibility that it was turned off deliberately to hide questionable activities. The other scenario is that perhaps a ship is reporting its location via AIS to be in an area with poor AIS coverage thus raising suspicion that the broadcaster might be deliberately reporting the wrong location. The way in which detection of this type of anomaly is approached is by building a coverage map AIS of receivers. The area of interest is divided into a grid and the probability of receiving a signal from each grid square is calculated using ratios of detections and non-detections from historical data. In order to compute if an unexpected AIS reception is anomalous the detection probability in the grid square of interest is subtracted from one. Furthermore to compute the probability of multiple unexpected receptions the product of each of their probabilities of being anomalous is subtracted from one. Finally in order to compute the probability that a period of AIS silence is suspicious the ships position is projected into the future and the probabilities of receiving a signal from each grid square that the ship is projected to be in are combined to yield an overall probability of not receiving a signal.

The third anomaly is unexpected port visitation. The presented model detects unexpected port arrival by mapping vessel types to port facility types and uses a Markov model approach to analyze patterns of port visitation. To detect if a port arrival was anomalous the probability of a certain ship arriving at that port was compared to a threshold.

The work goes on to dealing with close approach (rendezvous) behaviour. For each measurement received from a ship, its location, speed, and heading are used to project a straight line trajectory into the past and future over a defined time period. These trajectories are calculated for every ship and distance is computed as a function of time. The distances are then compared to find the closest point of approach. The behaviour is then flagged as anomalous if any two ships are unusually close together and travelling below certain speeds. However, because there are so many ships active at one time, all pairs can not be considered and areas must be broken down into smaller regions of consideration to reduce computational complexity.

Finally, the paper addresses the zone entry anomaly. Both the issues of whether or not a vessel has entered a zone of interest and the probability that a vessel will do so are considered. Wether or not a vessel has entered a zone can be computed by existing algorithms that determine if a point is within a polygon. Secondly, in order to come up with a probability of whether or not a vessel will enter a zone of interest in the future, the method assumes some distribution for tracks projected from the current position and determines the proportion that intersect the zone in question in order to compute a probability.

In the second part, the paper moves on to dealing with fusion of these five anomalies in order to produce an overall threat probability. This is achieved by constructing a Bayesian network and fusing all the inputs of the different anomaly models into a single probability.

Overall this is a great paper because it takes a fresh and more domain-focused approach to the problem of maritime anomaly detection which departs from the more wholesale generic statistical methods proposed in other works.

Chapter 3

Problem Definition

In this chapter we will give a detailed technical description of the problem that the rest of this work addresses. Before we can get to outlining the details of the actual problem we must first introduce and define a variety of concepts that will be used throughout the remaining chapters and should always be understood as defined in this chapter.

3.1 Motion

Because we are trying to analyze different types of motion or motion behaviours we will need to come up with a way to define motion. We describe motion using the following six variables:

1. Displacement
2. Velocity
3. Acceleration
4. Heading
5. Speed
6. Distance Traveled

For a particular entity, all of these values can be derived given a sequence of positions. We will be dealing with all of these variables as time series; the definition for time series is given in the next section 3.2.

3.2 Spatio-temporal Data

We start by defining the main type of data used in this work because it sets the context for all subsequent ideas. This is the key type of data that we will be dealing with because from this we can derive any of the other variables required to describe motion which were mentioned in section 3.1.

The type of data in question is *spatio-temporal*, or a *time series* of location data which in our case describes locations of mobile objects over water. Time series data refers to a set of n-dimensional data points which are measured at successive time intervals, where the time intervals are equal in length [21, p. 21]. In this case *spatio-temporal* data is a type of *time series* data with two additional properties:

1. The spatio-temporal data we will consider in the maritime domain has n = 3 dimensions: *x coordinates, y coordinates,* and *time.*

2. The time interval elapsed between successive data points is not constant.

Even though the time intervals between data points are not constant where necessary this can be handled via interpolation and re-sampling.

Because we are dealing with the maritime domain the mobile objects considered here are sea vessels, including all types of surface vessels that can be tracked with marine radar, satellite or AIS. Each data point is composed of a location, the *space* component, and a time stamp associated with this location, the *time* component (hence the name spatio-temporal). Together the space and time component represent a data point which is a tuple of three elements: t, x, y and is denoted as $\{t, x, y\}$. Here x and y describe the location of the entity on a Cartesian coordinate plane. The variable t describes the time at which each location observation is made, and can be thought of as a third or z dimension in a three-dimensional Cartesian coordinate plane.

As we have already mentioned in 3.1 there are many other types of data that will be used throughout the system but the idea is that the major type of input data from which all other data is derived is spatio-temporal data.

3.3 Derived Data

Other than the given spatio-temporal location data there is of course the possibility of making other observations such as velocity or heading, however we can calculate both of these as well as any other type of observations concerned with motion as long as we have *x, y, t.* This additional derived information can also be provided as input if it is available. All of the derived data are time series data. We will later discuss in more depth all of the additional derived types of information and how each is derived.

3.4 Displacement

Displacement is a vector, $\vec{d}$ and is computed as the change in position and it is the shortest distance between an initial and final position of some moving entity. If we have an initial position p_i and a final position p_f we can define displacement to be:

$$\vec{d} = \vec{p_f} - \vec{p_i} \tag{3.1}$$

And the magnitude of the vector is of course the distance between the two points given by

$$||d|| = \sqrt{d_1^2, d_2^2, \cdots, d_n^2} \tag{3.2}$$

Displacement is utilized to derive three other types of derived data which are covered in the following section.

3.4.1 Velocity, Speed, and Acceleration

For the variable of velocity we use the formal definition from physics which is speed with direction. We will define average velocity to be the vector representing the ratio of displacement to change in time:

$$\vec{v} = \frac{d}{\triangle t} \tag{3.3}$$

We can also define instantaneous velocity as the first order derivative of displacement:

$$\vec{v} = \lim_{\triangle t \to 0} \frac{d}{\triangle t} = \frac{d\boldsymbol{d}}{d\boldsymbol{t}} \tag{3.4}$$

When we do deal with derivatives and instantaneous velocity, and acceleration we will be using numerical derivatives because we do not have functions defining ship motion; we only have data points describing discrete positions.

Finally when referring to speed we will be referring to the magnitude of velocity which is a scalar value,

$$||\vec{v}|| = \sqrt{v_1^2, v_2^2, \cdots, v_n^2} \tag{3.5}$$

Acceleration is defined similarly to velocity. It is the change in velocity and is defined as the first order derivative of $\vec{v}$ or the second order derivative of $\vec{d}$ with respect to time.

3.4.2 Heading

For our purposes we will define heading to be *grid bearing* as it is also referred to in navigation. This type of heading can of course be expressed in radians, degrees or polar coordinates, however for explanatory purposes we will use degrees in the definition. Headings are scalar quantities denoted by h.

First of all the line between North and South runs along the y-axis of a Cartesian coordinate system that we will be using. North is set to $0°$. So if an entity is moving in the North direction it is moving up the y-axis towards increasing y values, and it has a heading of $0°$. Once we start moving clockwise from north the headings increase from $0°$ all the way up to $359°$ and back to $0°$; we use $0°$ again instead of $360°$ because the headings represented by these values are in fact the same direction.

3.5 Observations

Observations are denoted as tuples of elements, $o = \{e_1, e_2, \cdots, e_n\}$ where each element e is one of the variables describing motion. One of the elements must always be time. There is a subtle but very important distinction between an observation of some value and the true value itself. For example, an observation of a location at some time point is just a *view* of the true location at that time. The same can be said about any of the other variables we use to describe motion. What we mean by this is that the observation may not necessarily report the exact same data as the real location of a vessel at sea because the observation is provided by sensors which are subject to error. The way in which we allow for errors is to distinguish between a real value of some variable and a *view* of that value such as an observation. Some situations where this distinction is important is when dealing with simulations that include sensors which in turn observe simulated "true" locations of entities. Another situation where this distinction is necessary is when there are multiple sources reporting on the same entities. However, for our purposes observations and input data will almost exclusively, be the same because we only have one source reporting data for any given data point.

3.6 Paths and Trajectories

First of all observations can be grouped into any time series representing heading, speed, distance traveled etc. A time series S is then defined as a series of observations $\{o_1, o_2, \ldots, o_j, \ldots, o_{n-1}, o_n\}$. The observations $o_1 \ldots o_n$ are also ordered by time t which is strictly increasing, i.e. $o_1(t) < o_2(t) < o_3(t) < \ldots < o_n(t)$. The following restriction also always holds for any two distinct observations: $o_i(t) \neq o_j(t)$. Additionally time t can also represent time intervals in which case for every observation there will be a start time t_s and an end time t_f. The two restrictions above also hold when intervals are concerned. So in the interval the time restriction is no longer strict;if an observation i comes before an observation j then $o_i(t_f) \leq o_j(t_s)$. With regards to the second restriction, the intervals can not overlap however an interval can begin at the same time the previous one finished.

A path P is just a time series S of observations made up of (x, y) location data and is subject to the same restrictions as any other time series described above.

Time series are grouped into trajectories where at least one of the time series is a path. A trajectory, $T_i \in T^u$ where T^u is the set of all trajectories under consideration describes the motion of an entity in time and space. The remaining time series are of data derived from the path of the particular trajectory in question. So a trajectory T_i is then a set of time series $S_{i1}, S_{i2}, \cdots, S_{in}$.

So the main distinction between paths and trajectories is that a path describes location over time whereas a trajectory aims to describe motion over time.

Now with these definitions in place there is the question of how many or how few observations are grouped into a trajectory? In theory one could group all observations belonging to a particular entity into a single trajectory, but this is not very useful so we employ some guidelines for deciding the beginning and end of a trajectory. We are usually not considering the entire world in a scenario but rather an area of interest. For example, some areas of interest in the maritime domain are the Strait of Georgia off of the west coast of British Columbia, the Gulf of Aden in the Middle East, or a group of islands and surrounding waters, etc. So our first two cues for defining the beginning and end of a trajectory are: the observation at which the entity enters the area of interest, and the observation at which it leaves the area. To clarify, an entry or exit from an area can also refer to the first observation in the area of interest and the last observation in that area. These can of course happen at any point in the area. The other two cues are more natural and refer to the origin and destination of the particular entity. In this case a trajectory describes a journey from origin to destination, which in the maritime domain are usually ports. There is also the possibility that maybe only the destination or the origin location of an entity are in the area of interest, in which case we mix the four cues in order to identify the beginning and end observations of a trajectory giving the four possible combinations:

1. entry into area of interest (or first observation in area) & exit from area of interest (or last observation in area)
2. origin & destination
3. entry into area of interest (or first observation in area) & destination
4. origin & exit from area of interest (or last observation in area)

As for the issue of smaller trajectories, these can be composed of any slice of all of the time series belonging to some trajectory T_k whose beginning and end observations are identified as described above. Such trajectories will be referred to as *sub-trajectories*.

3.7 Behaviour

We can loosely define behaviour as: an action or set of actions described by the variables of motion. Whenever behaviour is mentioned it is important to keep in mind that it is in the context of observations that we are defining and analyzing it. That is, we are not referring to the actual behaviour of the entity being observed but just to the behaviour indicated by the observations. A real behaviour is not dependent on it being observed and the only thing we can say is that whatever behaviour we may think we are observing can only be an indication of the real behaviour. So we are really talking about observed behaviour unless we specifically refer to real behaviour.

We can describe behaviour as a collection of subsets of time series belonging to some trajectories. We can denote this as a set B where $B = S_{i1}, S_{i2}, \cdots, S_{in}$. Note that a behaviour must not necessarily contain time series describing all of the variables of motion, it could be defined by a single type of variable, for example speed, or just heading. A speed behaviour could be something a simple as "fast" or "slow" where a behaviour described by heading could be a "left turn" or moving "straight". Conversely a behaviour can also be described by many variables such as both speed and heading; a "fast left turn".

This is quite a general and sweeping definition which really implies that anything can be labeled as a behaviour. However, this is not a problem because we are interested in a finite though probably large and possibly growing number of behaviours (for the maritime domain anyway), so really what we focus on is behaviours of interest built from more basic behaviours. So we then define types of behaviour, ranging from simple to high level abstract behaviours. For example some simple behaviour types are:

1. **turns**: left turns, right turns
2. **starting**
3. **stopping**
4. **maintaining direction**
5. **not moving**

Building on these simple types we can then come up with new and more complex types of behaviour which can be composed of multiple simpler behaviours, for example:

1. **u-turn**: will be made up of multiple simple turns
2. **circling**: can be made up of multiple u-turns and simpler turns
3. **shuttling**: can be composed of multiple maintaining direction behaviours and u-turns
4. **patrolling**: can be composed of many simpler behaviours

So even though the definition allows for any set of ordered observations to be a behaviour we are really interested in a finite number of behaviour types. The intention is that we are not restricted as to what we can define as a behaviour of interest and that we also allow for new behaviours to be identified at any time. Notice also that the few behaviour types given as examples already start to form a hierarchy where more abstract behaviours are composed of simpler behaviours. These compositions lead to a deepening of the hierarchy by creating more levels. Furthermore if we look at say the behaviour type of “left turns” notice that there is no distinction between slow turns or fast turns, long or short turns, sharp or wide turns, etc. This is definitely part of the hierarchy and you can imagine making such distinctions as widening the hierarchy. These are just some basic examples to help introduce the notion of behaviour and will be explored in much more detail in chapter 4.

3.8 Anomaly

Next we need to tackle the definition of anomaly (sometimes referred to as an outlier) which has already been partially addressed in chapter 1. Following are some definitions that give a more general and intuitive feel for what we consider to be an anomaly.

Starting with the simpler and more intuitive definition as related to the problem of detecting anomalous behaviour we have that anomalies are: Patterns in data that do not conform to expected behaviour.

A more statistically oriented definition of an anomaly is given by D. Hawkins [12]: “An observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.”

The major idea which is hopefully communicated by the two definitions above is that an anomaly is some deviation from the norm. We can restate what an anomaly is using the definitions presented in this chapter as:

An anomaly is some **trajectory** or **sub-trajectory** which deviates from established normal **trajectories** or **sub-trajectories**

Because trajectories describe motion, what we are really talking about is anomalous motion. Of course this raises some major issues:

1. How are normal trajectories defined?

2. How can we measure the distance or similarity between any given trajectories in order

to establish if a trajectory is different enough from normal trajectories to be considered an anomaly?

From a statistical perspective we can define normal trajectories as those with the greatest probability of being observed given some data set. Once you can define what normal trajectories are then it is necessary to have a way to measure the similarity or distance between any two trajectories. This is done via distance or similarity measures.

There is also another kind of anomaly that will come up throughout this work that is less statistically motivated, and more domain related. This kind of anomaly is some type of predefined activity that there is interest in detecting and identifying. These predefined anomalies are domain dependent and are defined by domain experts. This also identifies some more obstacles:

1. How do we identify anomalies of interest in a particular domain ?
2. What kind of representation scheme do we use to express these anomalies ?
3. How do we decide if new trajectories match existing defined anomalies ?

The first problem is a matter of having access to domain expertise. The second problem is very important and is a big part of this research. It will be discussed in greater depth and shown that the representation scheme is absolutely crucial when designing a system with the aim of allowing robust and high level expression of interesting behaviours as well as identification of and comparison between such behaviours. Finally, the third problem is similar to the second problem identified in the previous section and is all about defining appropriate similarity and distance measures for whatever representation scheme is selected.

3.9 Anomaly Detection, Characterization, and Problem Discussion

Given some dataset, the problem of anomaly detection is that of separating a **small minority** of anomalous data, in our case anomalous behaviour, from the **large majority** of data, again in our case normal behaviour. The main idea in approaching this problem is to separate and characterize the majority of data as normal behaviour and to utilize this as a way to identify anomalies as abnormal behaviour. This key idea is described in figure 3.1

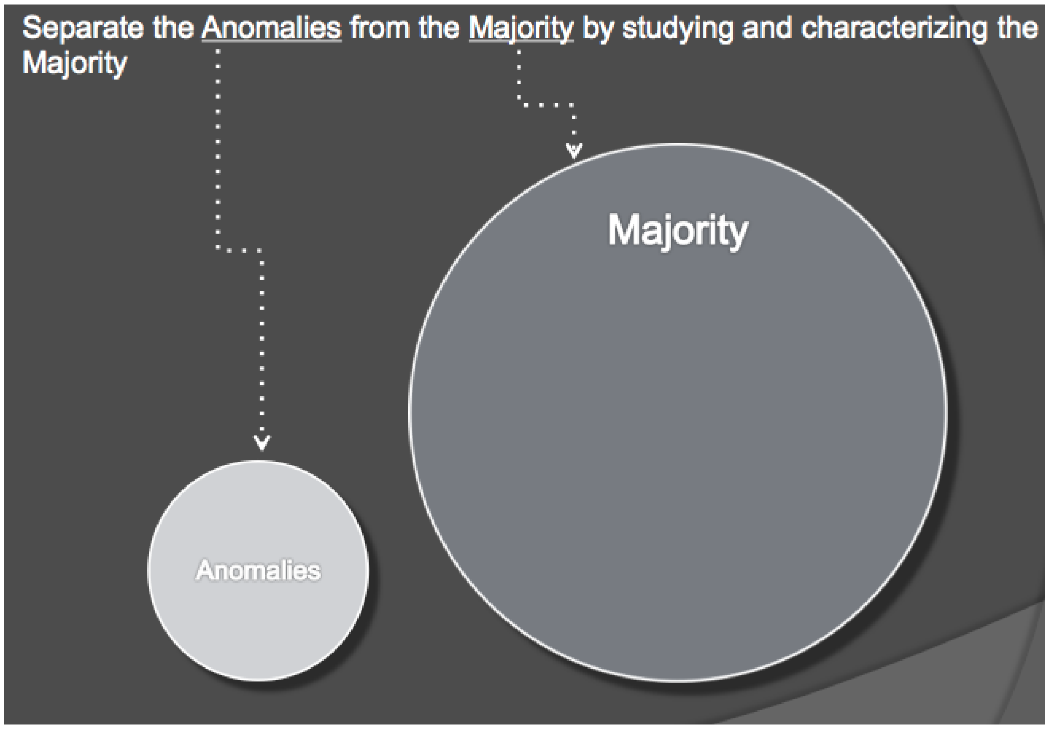

Separate the Anomalies from the Majority by studying and characterizing the Majority
Majority
Anomalies

Figure 3.1: Example ruled figure

Of course this happens to be a very challenging problem because the small minority of anomalous behaviour by its very nature is often made up of data which is inhomogeneous and hard to characterize. Secondly, it is very difficult to define a region that encompasses all possible normal behaviour. Then there is the problem of malicious adversaries adapting to cause their behaviour to appear normal. Furthermore the data itself is very noisy and there is a lack of labeled data for training and validation.

Another problem that naturally arises from anomaly detection is that of anomaly characterization. Anomaly detection refers to detecting *SOMETHING* but not *WHAT*. Often it is not sufficient to simply identify a set of anomalies. It is also necessary to characterize these anomalies in order to answer questions such as:

- Is this an interesting and relevant anomaly?
- What type of anomaly is it, to what kind of interesting behaviour does it correspond?
- Is this a new type of anomaly or has it already been identified?

So there are really two main problems that stand out when looking further into AD:

1. **Anomaly Detection** (AD)

2. **Anomaly Characterization** (AC)

First of all if we look at the AD problem as mentioned in the related work in section 2, there are three main ways to approach it: model-driven (top-down), and data-driven (bottom-up) approaches, with a third being a hybrid of the two. Each of these three methods lends itself differently to the two problems of AD and AC.

Data-driven methods seem to lend themselves better to the problem of AD, using them for characterization could be possible however it would be problematic and require a more significant amount of effort than using model-driven methods for the same purpose. For the task of AD as we have mentioned numerous times the general two step process is to build a model of normal behaviour and then to try to classify new data based on that model. As discussed in the related work section there are many sophisticated statistical methods to achieve this. If we did wish to try to deal with the AC problem one could imagine gathering a large data set containing multiple examples of a specific behaviour of interest and then doing this for all behaviours of interest and then performing multiple classifications of the detected data in order to try to characterize each distinct behaviour. This can pose a variety of problems such as gathering the data in the first place since these anomalous events are rare by their very definition, of course then there is need for adequate coverage of all variations of a particular behaviour to provide some robustness to the characterization algorithms. Even though the data-driven methods lend themselves best to AD the results as pointed out by the authors of some of the papers evaluating these methods are not very promising. One of the reasons given for this in the literature is that the feature set that the data-driven analysis is performed on is not rich enough. Because it is an instantaneous point based four dimensional feature set made up of location and velocity (x, y, vx, vy) the complexity of the behaviour it can describe is limited. So at the basic level what such an analysis would be looking at are differences in location and differences in velocity, nothing much more complex. This is compounded by the fact that these features are taken instantaneously and not considered as evolving or changing over time.

Now if we shift our attention to model-driven approaches they appear to be best suited for AC. This is because their aim is to facilitate the description of patterns; to build a model of some interesting behaviour which can be used to search for a match in new observed data. So the idea is to start with a definition of what it is we are actually looking for and then to perform pattern matching on the data in hopes of finding our anomalies of interest. This may not be as well suited for general AD because you can only find specific behaviours that you are searching for so everything else would be missed. Some of the challenges of this approach stem from the great variety in observed behaviours and the difficulty in actually describing/representing a behaviour of interest. If we are looking for a particular behaviour of interest that we have predefined we want our analysis to match all variations of that behaviour, in other words because not every single time a behaviour is observed will it be exactly the same we want robustness built into the analysis. This is a requirement even for basic behaviours. For example there are many different kinds of left turns that can be observed, obviously they will not look the same for every ship and in every instance so we want a way to capture all of the variations of such a behaviour. Another issue is how to compactly describe these patterns or behaviours that we are looking for, and how to do it in a semantically meaningful way. We would not wish to enumerate a set of all locations that could potentially describe our behaviour of interest and then repeat this task to account for variety of observed behaviours and then perform it for all patterns we wish to detect. This is definitely not compact and the resulting pattern representation would not be very meaningful from a human perspective. It would be most desirable to be able to represent such behavioural models or patterns in a compact and semantically meaningful representation that can also take into account the great variation inherent in any particular behaviour.

In this work we attempt to address some of the challenges posed by the AD problem, and in doing so we inevitably run into the AC problem as well so it becomes apparent that it is often necessary to deal with both. In order to try to do so in the most cohesive way possible we looked at the major themes running through both problems and have identified that the most promising avenue for us is to try to deal with the issue of representation. This is a common theme for both problems and we believe, as has also been pointed out in some of the literature, that this is an area that can provide a stepping stone for further investigation and hopefully improvements in dealing with both problems.

Chapter 4

Approach

This chapter will go into all of the details of our proposed spatio-temporal data representation and all of the pertinent algorithms. We will first start by giving a higher level picture by discussing the conceptual model behind the approach.

4.1 Conceptual Model

At the end of chapter 3 in our discussion we identified that the major theme we wish to focus on that touches both the problem of anomaly detection and anomaly characterization is representation. We propose a representation for spatio-temporal data in order to address the problems posed by the current instantaneous point and velocity feature sets used in a large portion of the existing literature on maritime AD. Additionally we also have the aim of providing a compact, semantically meaningful, and robust way to represent this data as well as methods to automatically generate such a representation.

The data we constantly refer to are vessel paths in the maritime domain but it is noteworthy that the representation proposed here is designed to work with any quantitative variable of interest. However, because we are focusing on analyzing motion, our variables of interest will be those used in describing motion defined in section 3.1.

The key idea of this representation is to allow the description of a sequence of changes over time, in a quantitative variable, or a collection of quantitative variables, by abstract concepts.

An illustrative example of this idea is how a human would go about describing an increase in speed over successive observations. We could describe such a change in speed over time with an abstract concept such as “speeding up”. We could go on to define the behaviour of “stopping” as a succession of decreasing speeds that eventually reach 0. In a similar way we can define what “starting is”. Now that we have these concepts we can piece them together and think of something a little more complex such as alternating intervals where the observed speed of some object switches between the two behaviours of “stopping” and “starting”. As a human observer we could describe this abstractly as “stop and go” behaviour.

The approach taken to generating this representation is a hierarchical one that revolves around the concept of abstraction. The goal is to build up increasingly abstract concepts describing qualitative features of the variable of interest on top of lower level concepts which in turn depend at the lowest level on the concrete observations provided by the raw data. In other words, higher level concepts are abstractions of compositions of lower level concepts. This collection of composable concepts is referred to as an *Abstract Concept Hierarchy.* The process of combining concepts from a particular level in the hierarchy to create the concepts in the next higher level of the hierarchy is that of *Concept Abstraction.* Some similar ideas have been presented in [23] which aim to understand the way that human spatial cognition works with emphasis on the importance of abstraction and qualitative descriptions of spatio-temporal data in human visual perception. Our work takes a different turn and a broader approach to qualitative representation of spatio-temporal data; however, thinking of the way humans perceive such data is a great analogy for how our approach operates. In fact one of our aims is that the automated analysis of spatio-temporal data using our proposed representation framework exhibits a high fidelity with the analysis a human would visually perform on the same data.

The key to generating a representation that can support this kind of abstract description of a time series is to focus on *change* in order to transform the raw data corresponding to that time series into a qualitative representation of that same data.

We will now shift our attention back to the more specific task of describing motion. The primary aim of the proposed representation is to facilitate the capture of features that occur over time. We can see there is a crucial distinction over current approaches which employ four dimensional feature sets focusing on instantaneous measurements of location and velocity because we describe features of motion occurring over time. A four dimensional feature set limits you to describing only simple motion anomalies such as anomalous velocity or entry into a restricted zone. There are already statistical solutions to detect simple anomalies like these. More interesting and complex behaviours can only be observed over time. The main thing to note is that because the features of motion we wish to represent develop over time, they require multiple observations to describe; they can not be described by a single four dimensional observation of velocity and position.

We believe that a representation with the ability to capture motion behaviour that happens over time is a good foundation to detect and characterize complex and interesting motion.

It is desirable for such a representation scheme to be robust and extensible enough to describe any type of motion features, from simple ones (e.g., right and left turns) to more complex ones (e.g., loops and u-turns). Additionally, it should also be simple to combine such features in order to define behaviours of interest. We will show how this is achieved in the remainder of this chapter.

4.2 Abstract Concepts

Abstract concepts are the backbone of the entire representation described in this chapter. At a high level we can view the proposed representation as just a sequence of abstract concepts.

Definition 1 *Abstract concepts are usually linguistically meaningful terms used to describe the change in a quantitative variable of interest over a time interval.*

We say "usually" linguistically meaningful because this is not a hard constraint. It is entirely possible to have an arbitrary name for an abstract concept. However, this will only detract from the value of the approach as it would become more difficult to use and less informative for a human user.

An abstract concept a at its most basic level is just a pair of strings. This is what all concepts share in common.

Definition 2 $a = (tag, shortTag)$ *where both the* tag *and the* shortTag *are unique identifiers.*

In addition to this we also define the following properties that apply to an abstract concept a:

1. A string describing the type of quantitative variable this concept applies to a_t

2. An ordered set of components a_c

3. A set of quantitative constraints a_{qc}

We will now describe each of the properties introduced above.

Quantitative Variable Type: This is rather straightforward and is used to identify the type of quantity this concept applies to. Remember that we wish to use semantically meaningful concepts so we need to have specific concepts for different variable types. For example a concept such as a “left turn” would be nonsensical when applied to speed. So if we have a concept pertaining to speed then we note that in the quantitative variable type and we would use something meaningful for a tag such as “fast” or “slow”.

Components: Really, the purpose of this ordered set of components is to describe the pattern of lower level concepts that make up the particular concept in question. This is an ordered set of concepts paired with modifiers. So each component is a pair $(concept, modifier)$. A modifier can be a positive integer, a positive integer preceded by the “!” modifier or the “*” modifier alone. The purpose of the modifiers is to express the number of times a particular concept occurs in the pattern described by the ordered set of concepts. The modifiers function as follows:

- “n”: where n is an integer and indicates the number of times this concept repeats in the pattern.

- “*”: this modifier occurs alone and means that the pattern will match any number of the concept in question as long as it is ≥ 1.

- “!”: is the negation modifier and must be followed by an integer n. It represents that we want to match n concepts that are any concept BUT the concept the modifier is applied to. So “!3” applied to a “right turn” concept means we are looking for any three concepts that are NOT a “right turn” concept.

Quantitative Constraints: The set of quantitative constraints is a set of boolean functions that take as input a real number value and output either true or false. The real number value describes a quantity of type a_t so it must be the same type of quantity as what the concept applies to. The input value is the value of the quantitative variable of interest over some time interval that this concept is to be assigned to. If the function returns true then the constraint is satisfied, otherwise it is not. There can be any number of constraints and they can be defined in any way as long as they adhere to the function prototype:

$$f : \mathbb{R} \rightarrow \{True, False\} \tag{4.1}$$

All of the properties described above are optional. The first property identifying the quantitative variable type described by this concept may be set to "All" which is just the generic default, and the remaining sets may all be empty for the most basic of abstract concepts.

Using these properties we distinguish between two main types of abstract concepts:

1. Atomic Concept: is just a single abstract concept with no components, $a_c = \emptyset$
2. Composite Concept: is an abstract concept that is composed of multiple atomic concepts, $a_c \neq \emptyset$

One of the reasons that this representation is robust is that abstract concepts do not depend on the magnitude of the change they are describing for their definition. This allows us to describe the quality, or the type of change. An increase in a variable will always be an increase no matter what magnitude. Using a concept like faster to describe an increase in speed always applies no matter what that increase is. Abstract concepts describe the key underlying nature of different types of changes in quantitative variables. This gives us the ability to describe features of time series with a minimal amount of abstract concepts while being unaffected by the huge variation in different time series describing the same phenomena. For example, we can imagine that there is huge variation between the time series describing the velocity of different moving objects, and it would be a huge task to account for and differentiate between all of these variations. In fact it is often not necessary because regardless of quantitative variations observed while comparing changes in speed of different moving objects, these time series often describe the same kind of features. So with a small set of abstract concepts we can describe the changes in velocity for any type of moving object without concerning ourselves with the multitude of quantitative variations.

If however we do wish to distinguish between time series describing the same type of behaviour but exhibiting different quantitative properties then we have the option of doing so by considering quantitative variations in between time series. We can be generic or robust in order to identify all data describing the same behaviour regardless of variation, however we can then distinguish as finely as we wish between the identified data to come up with more specific types of behaviour that are dependent on actual quantitative variation.

4.3 Concept Intervals and Concept Interval Sequences

Close to the idea of abstract concepts is that of a *concept interval*.

Definition 3 *A concept interval is an abstract concept over a time interval starting at time t_i and ending at time t_f which is associated with a time series defined over time t_1 to t_n inclusive. Where $t_1 \leq t_i < t_f \leq t_n$.*

Stated another way, a concept interval is just an abstract concept defined over a time interval that falls within a time series. A concept interval also contains information about the quantitative value and the magnitude of the change it is describing. This value is what the functions in the set of quantitative constraints for the abstract concept defined over the interval (if not empty) would apply to.

4.4 Abstract Concept Hierarchy

The Abstract Concept Hierarchy (ACH) contains all of the relevant abstract concepts that are used to describe change over time in a particular variable of interest. For a quantitative variable of interest to be represented using our representation it must have a concept hierarchy associated with it. Every variable has a different hierarchy associated with it because terms that are meaningful for describing time-dependent features of a certain variable may be irrelevant for another. For example, the concepts used to describe changes in velocity are not relevant for describing changes in position. It would make no sense to say that two positions are "faster", but we can say that positions are becoming "farther" or "closer" together over some time interval.

Concept hierarchies for variables are not fully defined; how many and which concepts are used for the ACH of a particular variable is highly dependent on the domain of application and what the users wish to capture. Concept hierarchies are very flexible and fuzzy by their very nature because they are composed of abstract concepts. There is in fact an infinite number of concepts that can be defined though only a few are interesting for any particular variable.

In this work the abstract concept hierarchies defined contain concepts that can be used in the representation of a trajectory. The idea is that once you can define enough levels of the different ACHs you can very easily describe any type of motion as a combination of lower level concepts. Any ACH starts at level 0, moves upwards, and has no ceiling. All ACHs have at least one level in common, level 0.

4.4.1 Level 0

This is the most important level of the hierarchy which sets the foundation for all other concepts in higher levels. It is also the only level that is shared amongst all of the ACHs for other variables.

Level 0 contains only four basic concepts which are quite easy to understand if we recall that the main idea behind concepts is to describe change. Each level 0 concept describes the change in whatever variable of interest that we are currently analyzing between two time points or between two time intervals. Additionally each level 0 concept is also an atomic concept.

In the diagram below a concept would be assigned to the change between the values of the variable of interest at times t_i and t_j, where $i < j$.

$$\begin{tikzcd} \bullet_{t_i} \arrow[r, bend left, "change"] & \bullet_{t_j} \end{tikzcd}$$

If we are dealing with intervals, the situation is somewhat different. We have to deal with intervals if our variable of interest is defined over intervals as opposed to instantaneously. This happens for values that we derive by using two consecutive values so in the case of heading in order to derive it we need the location at two different time points. Because of this our final result will be the heading over the interval defined by our two initial values. So at any point along that interval the heading will be the value we come up with.

The way we handle intervals is to look at the change in the two values assigned to them and to assign a new interval for the result which begins in the middle of the first interval and ends in the middle of the second interval. We do this to avoid overlapping concept intervals when the analysis is performed for an entire time series. The diagram below shows what the situation looks like with intervals.

$$\begin{tikzcd}\bullet_{t_i} \arrow[r, dotted, dash, "interval"] & \bullet_{t_j} \arrow[r, "change"] & \bullet_{t_k} \arrow[r, dotted, dash, "interval"] & \bullet_{t_l}\end{tikzcd}$$

The following diagram just depicts the resulting interval that the computed change from the previous diagram holds over.

$$\begin{tikzcd}\bullet \arrow[r, dotted, dash, "change\ interval"] & \bullet \\ t_{i+(j-i)/2} \arrow[u] & t_{k+(l-k)/2} \arrow[u]\end{tikzcd}$$

The basic level 0 concepts are listed below.

1. *increase* - an increase in the value of the variable of interest is observed between two time points or intervals. The definition of the "increase" concept as an abstract concept described in 4.2:

$$(tag\{"increase"\},\ shortTag\{"i"\},\ a_t\{"All"\},\ a_c\{\emptyset\},\ a_{qc}\{(f(x) = x > 0)\})$$

2. *decrease* - a decrease in the value of the variable of interest is observed between two time points or intervals. Abstract concept definition:

$$(tag\{"decrease"\},\ shortTag\{"d"\},\ a_t\{"All"\},\ a_c\{\emptyset\},\ a_{qc}\{(f(x) = x < 0)\})$$

3. *constant* - no change is observed in the value of the variable of interest between two time points or intervals. Abstract concept definition:

$$(tag\{"constant"\},\ shortTag\{"c"\},\ a_t\{"All"\},\ a_c\{\emptyset\},\ a_{qc}\{(f(x) = x = 0)\})$$

4. *undefined* - in certain contexts the variable of interest may have undefined values (heading when there is no movement and the orientation of the entity is not apparent is undefined). Abstract concept definition:

$$(tag\{"undefined"\},\ shortTag\{"u"\},\ a_t\{"All"\},\ a_c\{\emptyset\},\ a_{qc}\{(f(x) = x = Infinity)\})$$

This level is complete in the sense that it covers all possible changes that a variable describing quantitative information may exhibit between any two time points t_a, t_b, or intervals. Because *level 0* of the hierarchy is complete it can be shown that any type of of change in a variable can be represented. Furthermore if the representation is complete for any type of quantitative variable then it should also be complete for describing motion because we define motion to be defined by a collection of quantitative variables.

4.4.2 Lower Levels

Moving one level higher in addition to atomic concepts we can now start to have composite concepts. A composite concept is made up of one or more basic concepts found in the lower levels, in this case *level 0*.

For example, we can define a level 1 composite concept for a left-turn made up of a single component. The component is the basic concept *decreasing* applied to heading, with an unlimited number of occurrences (*). This is the general idea behind composite concepts. In this section we will outline some of the level 1 and 2 concepts for the various quantitative variables that are used to describe motion.

Heading: Level 1 Concepts

This level currently has six concepts which are quite generic and would apply to any domain.

1. *right turn* - Any increase in heading greater than h degrees. Abstract concept definition:

$$(tag\{\text{"}rightTurn\text{"}\},\ shortTag\{\text{"}r\text{"}\},\ a_t\{\text{"}heading\text{"}\},\ a_c\{(increasing, *)\},\ a_{qc}\{(f(x) = x > h)\})$$

2. *left turn* - Any decrease in heading less than h degrees. Abstract concept definition:

$$(tag\{\text{"}leftTurn\text{"}\},\ shortTag\{\text{"}l\text{"}\},\ a_t\{\text{"}heading\text{"}\},\ a_c\{(decreasing, *)\},\ a_{qc}\{(f(x) = x < -h)\})$$

3. *straight* - No change in heading is observed. Abstract concept definition:

$$(tag\{\text{"}straight\text{"}\},\ shortTag\{\text{"}s\text{"}\},\ a_t\{\text{"}heading\text{"}\},\ a_c\{(constant, *)\},\ a_{qc}\{\emptyset\})$$

4. *straightI* - This is another definition for the straight concept however it considers an increase in heading $\leq h$ as a straight movement. Abstract concept Definition:

$$(tag\{"straightI"\},\ shortTag\{"s"\},\ a_t\{"heading"\},\ a_c\{(increasing,*)\},\ a_{qc}\{(f(x) = x \leq h\})$$

5. *straightD* - This is the same as the "straightI" concept but with a decrease of less than h degrees. Abstract concept Definition:

$$(tag\{"straightD"\},\ shortTag\{"s"\},\ a_t\{"heading"\},\ a_c\{(decreasing,*)\},\ a_{qc}\{(f(x) = x \geq -h\})$$

6. *notMoving* - An undefined heading is observed. Abstract concept definition:

$$(tag\{"notMoving"\},\ shortTag\{"n"\},\ a_t\{"heading"\},\ a_c\{(undefined,*)\},\ a_{qc}\{\emptyset\})$$

Heading: Level 2 Concepts

1. *stopping* - Any a transition from any type of movement to the "notMoving"concept. Abstract composite concept definition:

$$(tag\{"stopping"\},\ shortTag\{"sp"\},\ a_t\{"heading"\},\ a_c\{(notMoving,!1),\ (notMoving,*)\},\ a_{qc}\{\emptyset\})$$

2. *starting* - Any a transition from no motion to any type of movement. Abstract composite concept definition:

$$(tag\{"starting"\},\ shortTag\{"sr"\},\ a_t\{"heading"\},\ a_c\{(notMoving,*),\ (notMoving,!1)\},\ a_{qc}\{\emptyset\})$$

Displacement: Level 1 Concepts

We use displacement concepts in order to look at the change in position. So for example we would like to know if a moving entity is moving farther from a location or returning to a particular location, perhaps moving back and forth between two locations. Other than the basic level 0 concepts we only use an additional concept for displacement and that is the concept of returning to the same location.

1. *returnedToLocation* - An increase in displacement followed by an almost equal decrease in displacement within some threshold h . Abstract composite concept definition:

$$(tag\{"returnedToLocation"\},\ shortTag\{"rtl"\},\ a_t\{"displacement"\},\ a_c\{(increasing,*),(decreasing,*)\},\ a_{qc}\{(f(x)=-h\leq x\leq h)\})$$

Speed: Level 1 Concepts

For speed we only define three different concepts for this level. One of which is "steady" and has three definitions just to allow us to ignore minor variations in speed, similar to the "straight" concept in the heading concept hierarchy.

1. *steady* - Constant speed. Abstract concept definition:

$$(tag\{"steady"\},\ shortTag\{"sy"\},\ a_t\{"speed"\},\ a_c\{(constant,*)\},\ a_{qc}\{\emptyset\})$$

2. *steadyI* - Constant speed within a threshold. Abstract concept definition:

$$(tag\{"steadyI"\},\ shortTag\{"sy"\},\ a_t\{"speed"\},\ a_c\{(increase,*)\},\ a_{qc}\{(f(x)=x<h\})$$

3. *steadyD* - Constant speed within a threshold. Abstract concept definition:

$$(tag\{"steadyD"\},\ shortTag\{"sy"\},\ a_t\{"speed"\},\ a_c\{(decrease,*)\},\ a_{qc}\{(f(x)=x>-h\})$$

4. *faster* - An increase in speed. Abstract concept definition:

$$(tag\{"faster"\},\ shortTag\{"f"\},\ a_t\{"speed"\},\ a_c\{(increasing,*)\},\ a_{qc}\{\emptyset\})$$

5. *slower* - An decrease in speed. Abstract concept definition:

$$(tag\{"slower"\},\ shortTag\{"sl"\},\ a_t\{"speed"\},\ a_c\{(decreasing,*)\},\ a_{qc}\{\emptyset\})$$

Speed: Level 2 Concepts

We defined three concepts here that describe the speed of a moving entity over some constant velocity interval. Note that the thresholds are dependent on the domain so they would have to be compiled by a domain expert. Also the value constraint functions apply to the actual speed value not the change in speed (this would be 0 for a steady speed interval). Depending on the situation we can of course define any other number of concepts describing really fast speeds or really slow etc.

1. *slow* - Any speed below some threshold h. Abstract concept definition:

$$(tag\{\text{"}slow\text{"}\},\ shortTag\{\text{"}sw\text{"}\},\ a_t\{\text{"}speed\text{"}\},\ a_c\{(steady, *)\},\ a_{qc}\{(f(x) = x < h)\})$$

2. *medium* - Any speed between two thresholds h_1 and h_2. Abstract concept definition:

$$(tag\{\text{"}medium\text{"}\},\ shortTag\{\text{"}m\text{"}\},\ a_t\{\text{"}speed\text{"}\},\ a_c\{(steady, *)\},\ a_{qc}\{(f(x) = h_1 < x < h_2)\})$$

3. *fast* - Any speed over some threshold h. Abstract concept definition:

$$(tag\{\text{"}fast\text{"}\},\ shortTag\{\text{"}fs\text{"}\},\ a_t\{\text{"}speed\text{"}\},\ a_c\{(steady, *)\},\ a_{qc}\{(f(x) = x > h)\})$$

4.4.3 Higher Levels and Inter-Hierarchy Concepts

As we move up to higher levels we have access to concepts defined in lower levels to use for new concept definitions. For example we could define a left loop to be a composition of multiple left turns. We can also define u-turns as compositions of turns and so on. As we continue to move up the hierarchy we are able to define more complex and interesting behaviours.

In this section we also wish to introduce the idea of *inter-hierarchy* concepts. An inter-hierarchy concept is similar to a regular abstract concept except it is a composite concept made up of two or more concepts where at least one concept is from a different abstract concept hierarchy than the others. For example we can come up with an inter-hierarchy concept describing movement in a circular pattern. In order to do this we need a concept from the displacement hierarchy and one from the heading hierarchy. First of all we need to detect that an object has moved away from a point and then back to that same point, so we can use the "returnedToLocation" concept for this, furthermore we need to know through how many degrees the object has turned so we would want either a left or right turn of 360 degrees. Additionally we could also restrict the diameter of the movement by putting a constraint on the distance travelled.

The concepts we have outlined so far are some common ones that are likely to show up for many domains. Beyond these the number of concepts we can define is actually unlimited and what one wishes to define depends on the particular application. We will not be presenting any exhaustive list of concepts. The idea is that the framework presented can facilitate the creation and use of any new concepts one wishes to use.

4.5 Bootstrapping, Merging, and Filtering

In this section we will cover the three basic algorithms used to generate, consolidate, and clean concept interval sequences. These are the *boostrapping, merging,* and *filtering* algorithms.

4.5.1 Bootstrapping

In order to be able to generate a hierarchical abstract concept representation, first, the *bootstrapping* process must be applied, which means making the jump from raw data into a basic qualitative representation. In order to achieve this transition we first start with the raw data which in the generic case is a time series of some quantitative variable of interest and we compute the change between every pair or consecutive data points. So we would compute the change between t_1 and t_2, t_2 and t_3 $\ldots t_{n-1}$ and t_n. From this basic computation we would end up with $n-1$ values representing the different changes in the variable we are analyzing. The next step would then be to take the $n-1$ change values and assign them each a level 0 concept from the appropriate ACH for whatever value we are analyzing. The basic pseudocode for this algorithm can be seen in 4.5.1.

To achieve this, for the heading variable, headings for a path need to be generated for every pair of consecutive observations. For example, if we have a path composed of observations $\{o_1, o_2, \ldots, o_n\}$, we can generate $n-1$ headings for the intervals $\{o_1, o_2\}, \{o_2, o_3\}, \ldots,$ $\{o_{n-1}, o_n\}$. This can be done by taking the derivatives of x and the derivatives of y over the same observation intervals resulting in $n-1$ derivatives for both x and y. These can then be combined using equation 4.2 applied to each observation interval, and will result in headings expressed in radians over the interval $[-180°, 180°]$. To dispose of negative values we can add $360°$ to them and all headings will be on the interval $[0°, 360°]$.

$$heading = arctan(d_x, d_y) \tag{4.2}$$

The next step is to analyze changes in heading between consecutive pairs of headings. To each of those changes we can assign the appropriate *level 0* concepts from the *concept hierarchy.* We can have an *increase, decrease, constant,* or *undefined* change in heading, where an undefined change means that the entity in question was not moving at that time so it has no heading. So the result is that we can turn a sequence of headings $\{h_1, h_2, \ldots, h_{n-1}\}$ into a *concept interval sequence* $\{[1 : c_1 : 2], [2 : c_2 : 3], \ldots, [n-2 : c_n : n-1]\}$ where each

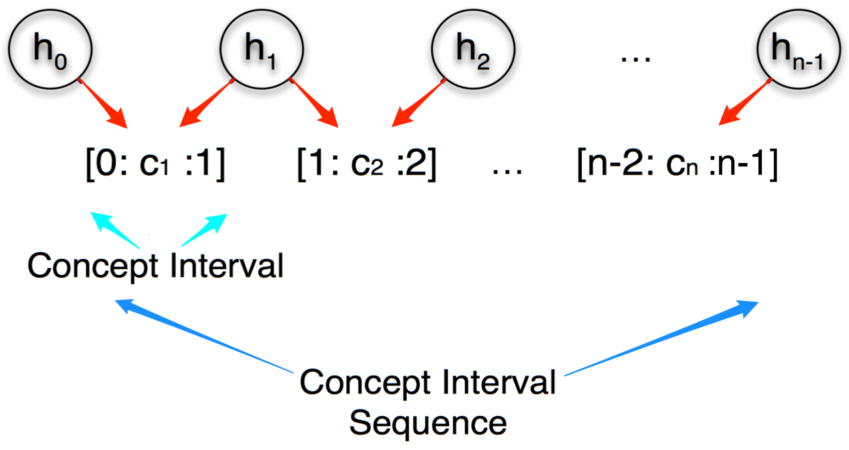


Figure 4.1: Bootstrapping: converting raw data into a concept interval sequence

c_i is a *level 0* concept and each $[a : c_i : b]$ is a *concept interval* (see Figure 4.1). This completes the process of bootstrapping and the shift from a quantitative representation to a qualitative one. It is important to note however that for each concept interval quantitative data is kept track of in order to allow for constraints to be used when defining concepts.

Algorithm 4.1 Bootstrapping algorithm pseudocode.

```
procedure BOOTSTRAP($headings$)
    $conceptSeq \leftarrow List()$
    repeat
        $first \leftarrow headings[0]$
        $second \leftarrow headings[1]$
        if first == Infinity then
            $conceptSeq \leftarrow (first.time, undefined) + conceptSeq$
        else if first == second then
            $conceptSeq \leftarrow (timeInterval(first, second), constant) + conceptSeq$
        else if $first < second$ then
            $conceptSeq \leftarrow (timeInterval(first, second), increase) + conceptSeq$
        else if $first > second$ then
            $conceptSeq \leftarrow (timeInterval(first, second), decrease) + conceptSeq$
        end if
        $headings \leftarrow headings.tail$
    until h.isEmpty
end procedure
```

4.5.2 Merging

The bootstrapping algorithm operates on a point to point basis, that is we only look at the current heading and the next at every step so intervals are never longer than the minimal time between observations. There is of course the possibility that the bootsrapping algorithm will generate sequences that contain runs of the same concept back to back as in {*(1, increasing, 2), (2, increasing, 3), (3, increasing, 4), (4, decreasing, 5)*} in this case what we want to do is merge the run of intervals which are all labeled with the increasing concept to get the following result: {*(1, increasing, 4), (4, decreasing, 5)*}. The numbers represent arbitrary unit beginning and end times of the different concept intervals where in the actual software implementation we use the real dates and times of the actual observations.

This can be achieved using the algorithm described by the pseudocode listed in 4.5.2. The function "mergeConceptIntervals" requires a bit of explanation. It is a concept hierarchy specific function, that is different for different quantitative variables that merges the concepts for that particular variable and the values. Sometimes simple addition is fine other times it may be necessary to express units over time etc. The details are all present in the software implementation.

Algorithm 4.2 Merging algorithm pseudocode.

```
1: procedure MERGE($conceptSeq$)
2:     $mergedSeq \leftarrow List()$
3:     repeat
4:         $toMerge \leftarrow conceptSeq.takeElementWhile(conceptSeq.head = element)$
5:         $mergedSeq \leftarrow toMerge.fold(c_1.mergeConceptIntervals(c_2)) + mergedSeq$
6:         $conceptSeq \leftarrow conceptSeq.drop(toMerge.size)$
7:     until conceptSeq.isEmpty
8: end procedure
```

4.5.3 Filtering

As previously mentioned alongside the concept interval sequences we also keep track of quantitative information depending on the quantitative variable being analyzed. This information is tracked for every concept interval. What the filtering algorithm allows us to do is to remove concept intervals that meet or do not meet some criteria for a given quantitative measure. For example if we aren't interested in concepts that occur over an interval that is less than a second we can filter these out. Perhaps we don't want to take into consideration very small heading changes, we can filter these out as well.

The point of filtering is to try to clean up noise and make the representation more concise. For example if we had a situation with the following concept sequence: {*(1, increasing, 20), (20, decreasing, 21),(21 increasing, 45)*} we might want to get rid of the intermittent decreasing concept if the change in heading was insignificant or if the time was too short etc. in order to consolidate the representation to {*(1, increasing, 45)*} thereby ignoring the noise.

The algorithm takes as input a sequence of concept intervals, a domain which to constrain (time, or quantitative value), and a constraint function with a threshold. Where the constraint function is one of $(<, >, ==, !=)$ and it returns the original sequence with only the concept intervals that satisfy the constraint function. So if the constraint function was specified as $<$ and the threshold was 10.0 with a domain of time then the result would be the input sequence with all concept intervals that did not have duration of less than 10 seconds, removed.

This is a straightforward algorithm and all that it does is find all instances of concept intervals in the input sequence not satisfying the given constraint and removes them.

4.6 Concept Abstraction

Once the initial concept interval sequence expressed in *level 0* concepts is obtained, the next process, called *concept abstraction*, is utilized to move up the hierarchy and reduce the representation further to more interesting and higher level concepts. The *concept abstraction* process must be performed on a level by level basis starting from *level 0* and moving up in order to generate the final representation. This involves two major steps:

1. Identification: Finding the subsequences of concepts from a lower level that correspond to the components making up higher level concepts as well as adherence to any constraints imposed on those components.

2. Substitution: Substituting the higher level concepts for the identified subsequences of lower level concepts.

If we refer to section 4.2 in the *identification* step we are searching for the sequence described by the concepts and modifiers in the component set, a_c, of an abstract concept. The pseudocode in 4.6 describes at a high level the procedure for searching for the components of an abstract concept. This is basically the identification step described above and it returns true for a match as well as the beginning and ending indices for the matched subsequence. This procedure is performed multiple times in order to identify each occurrence of a concept and it is repeated for all concepts in a level. The substitution function and other details are not listed as they are rather straightforward. Again these can be found fully implemented in the software API.

Algorithm 4.3 Basic Concept Abstraction algorithm pseudocode.

```
procedure AbstractConcept($conceptSeq, concept$)
    $components \leftarrow concept.components$
    $toSearch \leftarrow conceptSeq.dropWhileElement(element \neq components[0].concept)$
    $dropped \leftarrow conceptSeq.size - toSearch.size$
    $beginIdx \leftarrow dropped - 1$
    $endIdx \leftarrow beginIdx$
    repeat
        $concept \leftarrow components[0].concept$
        $modifier \leftarrow components[0].modifier$
        if $toSearch.size > 0$ then
            $conceptsMatched \leftarrow toSearch.applyModifier(concept, modifier)$
            if $conceptsMatched > 0$ then
                $components \leftarrow components.tail$
                $toSearch \leftarrow toSearch.drop(conceptsMatched + dropped)$
                $endIdx \leftarrow endIdx + conceptsMatched$
            else
                $returnfalse$                                      ▷ no match
            end if
        else
            $returnfalse$                                          ▷ no match
        end if
    until $components.isEmpty$ $return(true, beginIdx, endIdx)$
end procedure
```

4.7 Smoothing

We utilize two main algorithms for smoothing; though they are refered to as *linear sliding window generalization* and *nested sliding window generalization* we will refer to them simply as linear and nested smoothing to reduce conflicts with the polygonal generalization algorithm presented later in this chapter. Both of these algorithms were introduced in [22] and more details can be found there.

$$w_t = \frac{1}{k}\sum_{i=s}^{s+k-1} v_i,\ w_{t+1} = \frac{1}{k}\sum_{i=s+1}^{s+k} v_i, \dots \tag{4.3}$$

The linear smoothing algorithm can be seen summarized in equation 4.3. This just takes the average of k vectors in a sliding window. Here, t in the resulting vector w_t is $t = s + \frac{k-1}{2}$. Where s is the beginning index of the sliding window and v_i is vector i in the path. The vectors are the locations that make up the path to be smoothed.

The nested smoothing algorithm is described by equations 4.4 and 4.5. In these equations l is the length of some slice of vectors and this is the parameter that must be set for this algorithm; we will refer to it as NS for short.

What the nested smoothing algorithm does given a window size l, is to first compute the sum of all l sized vector windows that cover the vector, v_t, at the time point that we are currently computing a smoothed vector for. There are l such windows so the total number of vectors that get summed is then l^2. The resulting smoothed vector at time t is w_t which is the average of all of the vectors that were summed up.

$$u_{n,n+l-1} = \sum_{i=n}^{n+l-1} v_i,\ u_{n+1,n+l} = \sum_{i=n}^{n+l-1} v_i \tag{4.4}$$

$$w_t = \frac{1}{l^2}\sum_{i=t}^{t+l-1} u_{i-l+1,i} \tag{4.5}$$

4.8 Incremental ϵ-Generalization

The main reason for making use of a generalization algorithm is to be able to have control over the types of features that we ultimately want to represent. Depending on the situation we often wish to suppress small features in a path and just represent the major ones. For this purpose we employ the previously introduced Incremental ϵ-Generalization algorithm.

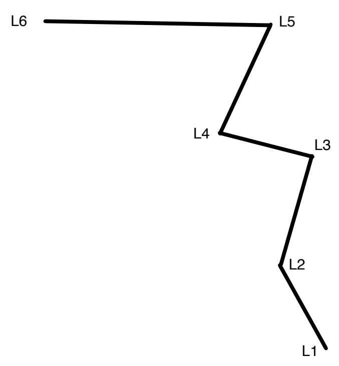
The algorithm described in this section was introduced in [23]. However because it was only briefly described at a high level in that particular paper we had to spend some significant time and effort in figuring out exactly how it should be precisely defined and implemented. So in addition to the information available in the original paper we will also outline some of the more detailed aspects of the algorithm that were not covered but are necessary for actually implementing and making use of it.

The main idea of generalization is to simplify a given path such that the resulting simplified path only contains the general shape of the original path and suppresses deviations and unwanted smaller features. This is achieved by building a simplified polygonal path which is different from the original path by less than some distance ϵ.

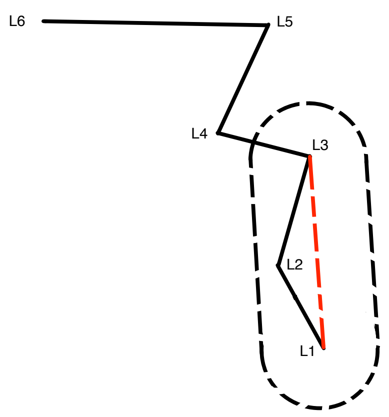
We begin with the path shown in figure 4.2(a) and take the first point L1 and try to build a line segment from here. We include L2 in our segment because the generalization of a line is always a line so we have to have at least two points. If we move on to the next figure, 4.2(b) we can see a dashed boundary and a center line inside this boundary. We will refer to everything inside the dashed boundary as the ϵ-area. Anything inside the ϵ-area is less than ϵ away from the center dashed line segment; the line L1-L3 in this case. In this step we add L3 to the polygonal generalization because does not exclude any other points out of the ϵ-area; L2 is still $< \epsilon$ distance away from the generalized polygon the line L1-L3. Moving on to the next step, figure 4.2(c), we try to add point L4 but this causes L3 to fall out of the ϵ-area so we can not add it at this moment. At this time we check to see if the addition of another point allows us to restore the ϵ requirement. And it so happens that adding point L5, figure 4.2(d) allows us to have a line segment L1-L5 that causes all of the points we have considered so far to fall into the ϵ-area. At this point in the procedure we see that we can no longer add any points to the segment so we use the line L1-L5 (dropping all points in between) as the first line segment of our generalized polygon. We now start the procedure from the beginning but with the end of our last segment, L5, used as the beginning of the next one that we will build. This procedure is then repeated until the entire path is incrementally generalized to a polygon.

Given the procedure we just outlined, the paper [23] goes on to describe an efficient algorithm for obtaining the polygonal approximation of a path. We will go over some of that same information here and then describe some of the additional necessary details that were not elaborated on in the original paper, but that are necessary for implementation.

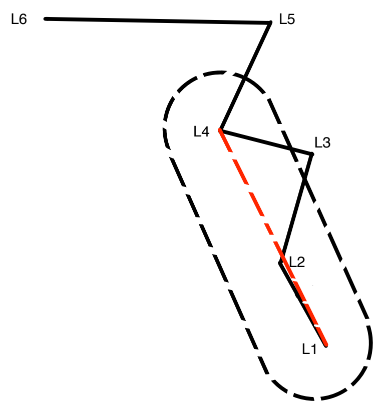
Before presenting the details of the algorithm we need to define some terms. These are visually depicted in figures 4.3(a) and 4.3(b). Every time we build a new line segment S_i to extend the polygonal generalization we start with a single point which we will refer to as C. The point C is the center of a circle with ϵ radius. Given the point C and the ϵ-circle we can proceed to build the ϵ-area. We do this by extending a line out from C and anything



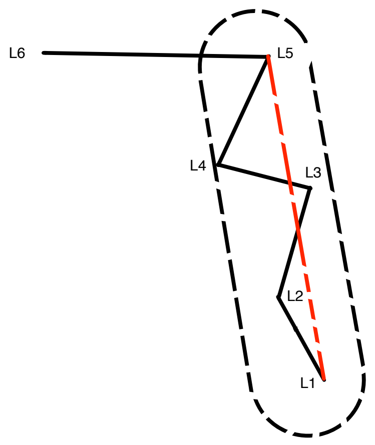
(a) Original path.



(b) First fit.

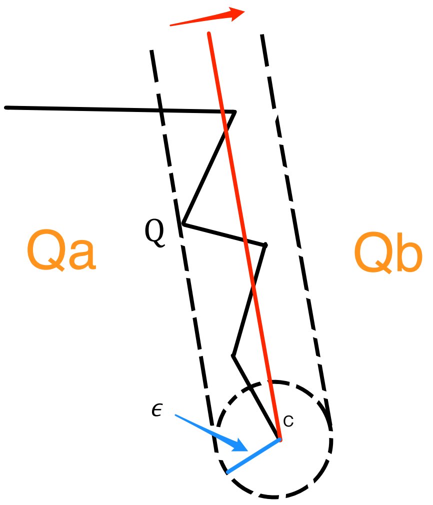


(c) Second fit.

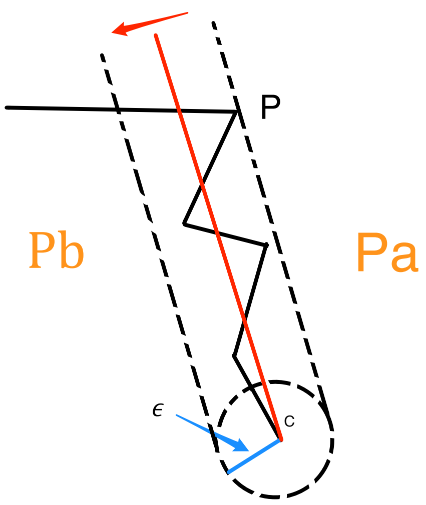


(d) Final fit.

Figure 4.2: Incremental ϵ-Generalization, basic idea.



(a) Q- left border.



(b) P - left border.

Figure 4.3: Incremental ϵ-Generalization.

within ϵ distance away from this line is in the ϵ-area. We can see this boundary depicted by the dashed lines in figures 4.3(a) and 4.3(b). When referring to the left and the right borders of the ϵ-area we are doing so as if we were on the point C and facing down the extended line that runs through the middle of the ϵ-area. We can see in figure 4.3(a) that a line from the border point Q tangent to the ϵ-circle forms the left border of the ϵ-area. In this particular situation when we are using a point Q as the border point the area outside of the border immediately to the left of it is labeled Qa and the area outside of the ϵ-area to the right side is labeled Qb. If we look at figure 4.3(b) we can see the same ideas except we use a point P as the right border point and the names of the areas are prefixed with P instead of Q.

The way in which a segment is built is by finding all of the points in the ϵ-area defined by extending a line from C and rotating it around to try to include new points into the area. As points are added the amount of rotation of the line is restricted until no more points can be added and the segment construction is complete. Given an empty set of points S_i we fill it by collecting all of the points that will be generalized by segment i. The algorithm goes through the following steps:

1. Pick the beginning of a new segment, point C

2. Take all points that are within the ϵ-circle centered at C and add them to S_i. We do this because these points do not restrict the rotation of the segment so we need not perform any further checks on them.

3. Set both the P and Q border points to the first point in the path that falls outside of the ϵ-circle. This is the first point that will restrict the rotation of the segment.

4. Select the next point, x_n and test whether it falls into the Pb or the Qb areas. If either is the case, the point can not be added to the segment and we have reached the end of the current segment. Note that when we check both the Pb and the Qb areas we are effectively rotating the whole ϵ-area trying to capture the additional point. See figures 4.3(a) and 4.3(b). Start at step one using the last point added to S_i as the new C of the next segment.

5. If x_n is neither in Qb or Pb we move on to check whether it is in Qa or Pa.

6. If it is in Pa we set it to be the new border point P and check if the current border point Q is still in the ϵ-area. If this is the case then x_n is added to S_i and P is updated to x_n. Go back to step 4 and consider the next point x_{n+1}.

7. If it is in Qa we set it to be the new border point Q and check if the current border point P is still in the ϵ-area. If this is the case then x_n is added to S_i and Q is updated to x_n. Go back to step 4 and consider the next point x_{n+1}.

8. If x_n is neither in Qb, Pb, Qa or Pa then it must be in the ϵ-area so we add it to S_i and go back to step 4 and consider the next point x_{n+1}.

Now we will go over some of the calculations necessary to detect if a point is in the different areas Pa, Pb, Qa, Qb, and the ϵ-area. In order to to do this the first step is to build the ϵ-area. We do this by taking the point C and building the epsilon circle as in figure 4.3(a). Once we have the circle we can first calculate Qa and Qb by first constructing a tangent to the ϵ-circle through the border point Q. The tangent computations are described in figure 4.8 and equations 4.6 to 4.11. The value r which is the radius is given (ϵ), pc can be computed by a simple distance calculation, and t can be obtained using the pythagorean theorem. The angles θ_1 and θ_2 are the angles between the two tangent lines adn the horizontal at point p. And finally to calculate the actual cartesian coordinates of the two tangent points we use the

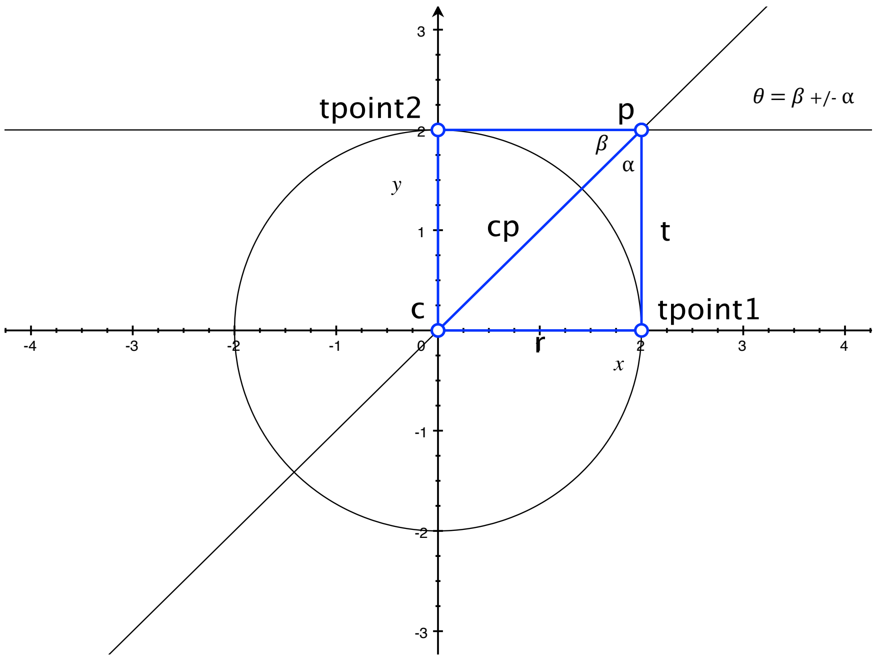


Figure 4.4: Incremental ϵ-Generalization.

length of the tangent line t as a radial coordinate and the angle θ as an angular coordinate and convert these polar coordinates to cartesian ones using equations 4.10 and 4.11. In these equations the "Qoffset" refers to a quadrant offset that is necessary to calculate the proper cartesian coordinates and it is based on the quadrant that the point p is in. The offsets for the different quadrants are: I-π, II-π, III-0, and IV-2π. Finally one must select the right tangent point because there are always two tangent lines from a circle to an external point. The selction depends on if we are using the point P or Q to construct the tangent. If it is Q we want to select the point on the left of the ϵ-area center line, and if it is P we want the tangent on the right of this line.

$$\alpha = arcsin\left(\frac{r}{pc}\right) \tag{4.6}$$

$$\beta = arctan\left(\frac{y_c - y_p}{x_c - x_p}\right) \tag{4.7}$$

$$\theta_1 = \beta + \alpha \tag{4.8}$$

$$\theta_2 = \beta - \alpha \tag{4.9}$$

$$tpoint_x = rcos(\theta + Qoffset) + x_p \tag{4.10}$$

$$tpoint_y = rsin(\theta + Qoffset) + y_p \tag{4.11}$$

Once we have the tangent point on the circle we can compute the slope of the line and using this we can build the center line in the ϵ-area which extends from the point C. We can do this easily because the lines that delineate the ϵ-area are all parallel. Now to test if a point is in Qa we just test whether it is to the left of the tangent line we built through Q; we can use equation 4.12 which will return a negative number if the point lies on the right, a positive number if it is on the left, 0 if it is on the line, and if the line is horizontal points above are considered to be on the left. Note that in equation 4.12 points 1 and 2 define the line and point 3 is the one we wish to test (det means determinant).

$$det \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1) \tag{4.12}$$

In order to determine if a point is in the ϵ-area, we use the center line extending from C (which is just the slope of the tangent to the ϵ-circle through Q) and test that the point is less than ϵ away from the line. To compute the distance from a line to a point we use equation 4.13, details can be found at [30].

$$\frac{|(x_2 - x_1)(y_1 - y_3) - (x_1 - x_3)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{4.13}$$

To test if a point is in Qb we test that it is not in Qa and not in the ϵ-area. We are of course still using Q to define our ϵ-area.

The operation for checking if a point is in Pa or Pb is identical except we use the point P to construct our ϵ-area by computing the tangent and the parallel boundary lines. Also to test if a point is in Pa we check that it lies to the right of the tangent through P, figure 4.3(b).

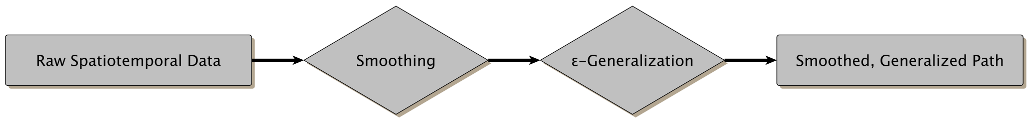


Figure 4.5: Pre-processing steps.

4.9 Example

We currently have AIS vessel location data for the entire year of 2009. It has been collected by the United States Coast Guard and primarily covers coastal U.S. waters. The data covers five areas: the Alaskan, Hawaiian, Pacific, and Atlantic coasts, as well as the Gulf of Mexico. The database is made up of 204 File Geodatabases each representing one month of data for a single UTM zone. The represented UTM zones 1–11 and 14–19 cover the entire United States. In addition to location information, the data also contains extensive information about the vessels themselves, like vessel ID, type, name, physical dimensions, voyage information etc. The data has been obtained from the Multipurpose Marine Cadastre (MMC), an integrated marine information system operated by the National Oceanic and Atmospheric (NOAA) Coastal Services Center and the Bureau of Ocean Energy Management.

In this chapter we will walk through an example using a real ship path from the dataset decribed above. We will apply the different algorithms proposed, present, and discuss the results. The trajectory we are going to use for our example is depicted in figure 4.7(a) in the $xy-plane$ and in three dimensions in figure 4.7(b). We will use the processes described in figures 4.5 and 4.6 to pre-process the data and generate the abstract concept representation.

Here is some summary information about the ship voyage:

- The voyage starts on 2009-06-01@00:00:00.000 and ends on 2009-06-04@23:03:00.000.
- There are 1461 data points reporting observed locations (via AIS).
- The total distance travelled for the journey is 155.76 Km.

As we can see this voyage happens over a period of 4 days and a distance of approximately 160 Km. This does not seem like a long way to travel in four days but notice after some initial movement the ship does not move for almost three days. There is also another interesting

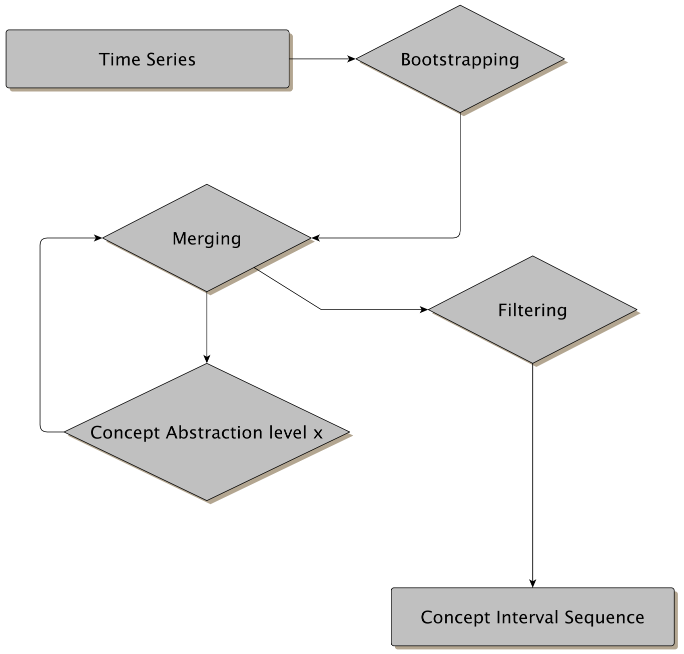


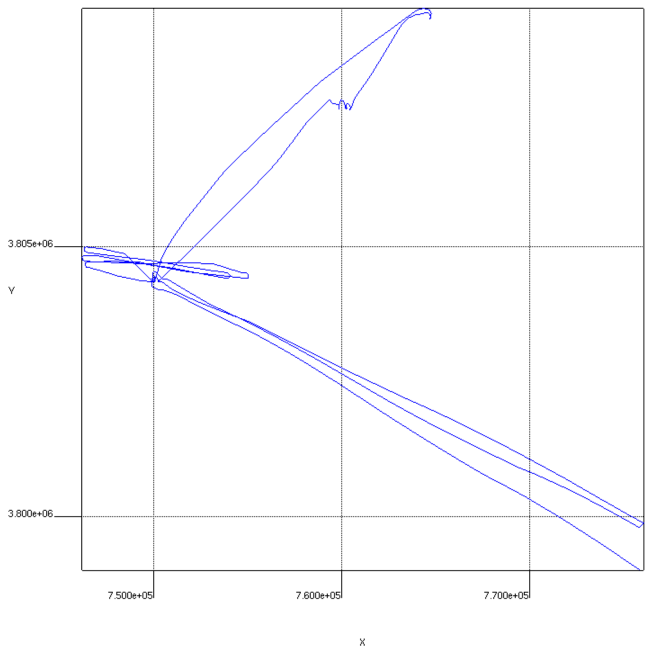
Figure 4.6: Representation generating steps.

feature in the ships' voyage, there is a section in the voyage where it shuttles back and forth between two points a few times before going on its way. Before moving on to performing the analysis and generating the representation let us say we want to detect that shuttling behaviour. To do this we can add some new behaviours. First of all we want to detect the u-turns involved in the shuttling behaviour; right now they would be represented as plain right or left turns. To the level 2 abstract concept hierarchy for the heading variable we add some u-turn abstract concepts that represent turns between 160 and 200 degrees as u-turns:

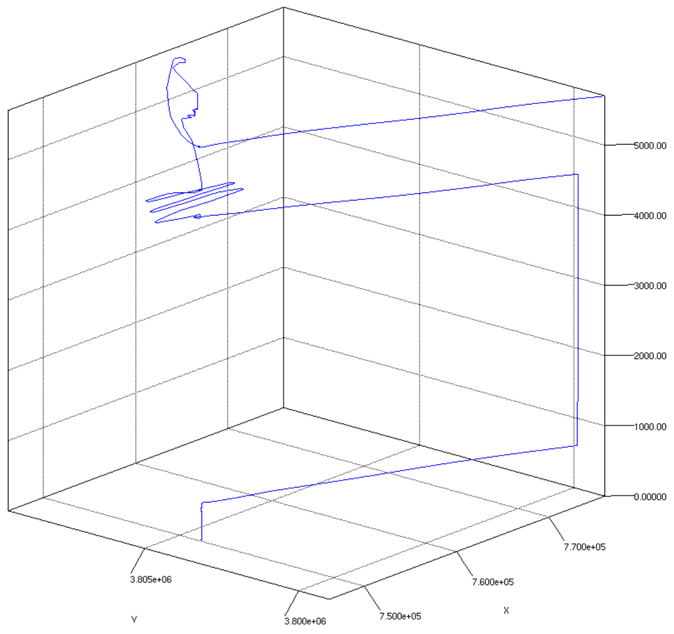
$$(tag\{"rightUTurn"\},\ shortTag\{"ru"\},\ a_t\{"heading"\},\ a_c\{(rightTurn, *)\},\\ a_{qc}\{(f(x) = 160 < x < 200)\})$$

$$(tag\{"leftUTurn"\},\ shortTag\{"lu"\},\ a_t\{"heading"\},\ a_c\{(leftTurn, *)\},\\ a_{qc}\{(f(x) = -160 > x > -200)\})$$

Without use of smoothing or generalization by just applying the automated analysis to the path we get 91 features in the resulting representation. The analysis detects every



(a) 2D view of the example path.



(b) 3D view of the example path.

Figure 4.7: The example ship path.

fluctuation in movement from the smallest occurring in just under a minute to the longest which takes place over two and a half days. The resulting representation can be seen in table A.1 in the appendix.

Moving on, we want to clean up the path and reduce the amount of small feature noise. Next we apply the nested smoothing algorithm with a NS parameter of 20, and we also apply the ϵ-generalization algorithm with an ϵ of 2000. The distances for the ship path are measured in meters so by using 2 Km for the generalization algorithm we get rid of smaller features under 2 Km. We can see the results of the pre-processing steps in figures 4.8(a) and 4.8(b). Smoothing and generalization also reduced the number of features in the resulting representation from the 91 found in table A.1 to the 20 shown in table A.2.

Now we want a way to combine the multiple u-turns and represent them as a single feature, a shuttling behaviour. There are a few ways to do this but we will do it in two steps. First we will add a level 3 “back&Forth” concept that will represent two consecutive u-turns. We will use two definitions for it because we can have a left u-turn followed by a right u-turn or vice versa so we wish to capture both possibilities in one concept.

$$(tag\{"back\&Forth"\},\ shortTag\{"bf"\},\ a_t\{"heading"\},\ a_c\{(rightUTurn, 1),\\ (leftUTurn, 1)\},\ a_{qc}\{\emptyset\})$$

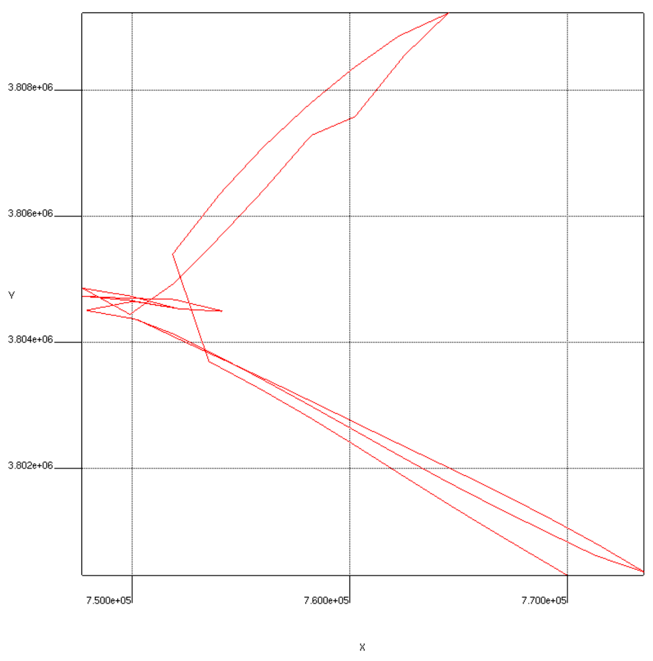
$$(tag\{"back\&Forth"\},\ shortTag\{"bf"\},\ a_t\{"heading"\},\ a_c\{(leftUTurn, 1),\\ (rightUturn, 1)\},\ a_{qc}\{\emptyset\})$$

Finally we can add our “shuttling” concept which we define to be any number of consecutive “back&Forth” behaviours.

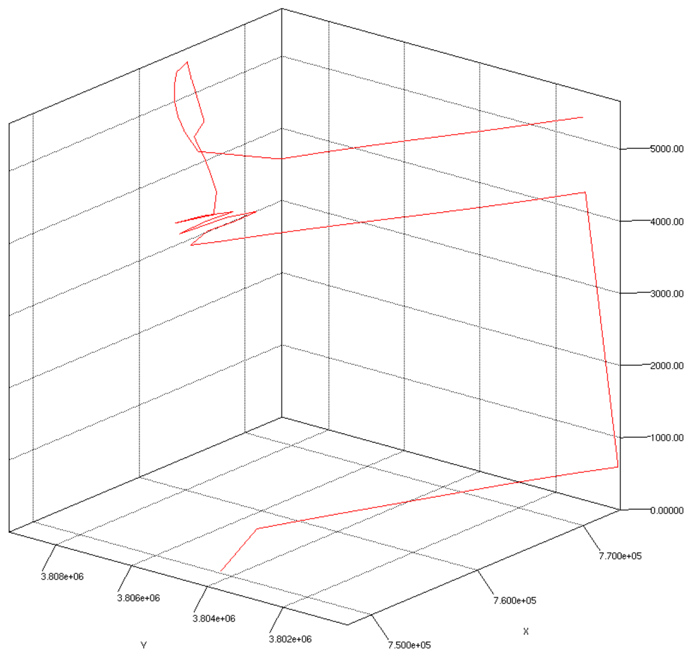
$$(tag\{"shuttling"\},\ shortTag\{"sh"\},\ a_t\{"heading"\},\ a_c\{(back\&Forth, *)\},\ a_{qc}\{\emptyset\})$$

If we generate the representation again with these new concepts we can see that the shuttling behaviour is properly represented. This can be seen visually in figure 4.9 where the section of the original path exhibiting the behaviour in question is bounded by the black box. And in the feature representation instead of the alternating right and left u-turns in table A.2 we have a single feature:

shuttling: 0.455 degrees, time: 4575.49 min to 4893.49 min.



(a) 2D view pre-processed path.



(b) 3D view of pre-processed path.

Figure 4.8: The pre-processed ship path.

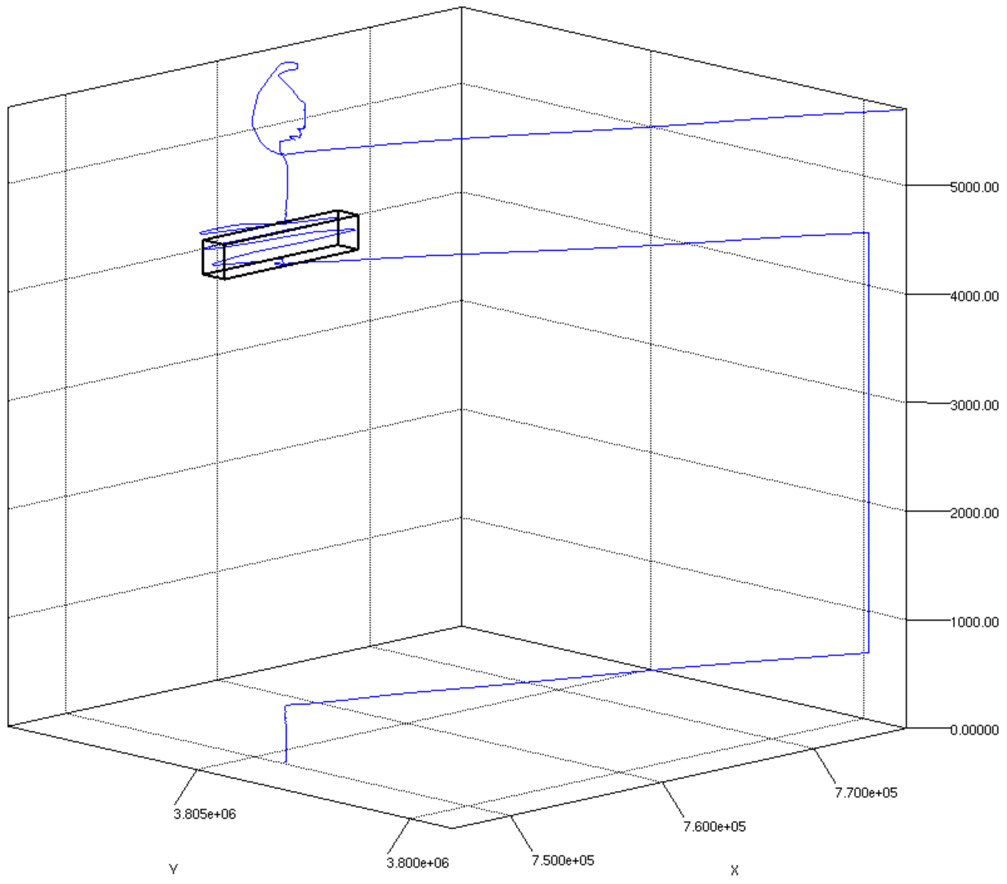


Figure 4.9: Shuttling behaviour.

Chapter 5

AnoDeC API Design

The original research prototypes were written in multiple languages: R, GNU Octave, C, and a little bit of Perl and Bash. However because this would probably be very hard for anyone to use as well as hard to maintain and expand, everything was re-written in a single language and combined into a coherent and hopefully useful API. The language chosen was Scala [2]. The reason for this is that it supports functional and object-oriented programming and it is a modern and advanced programming language providing many useful constructs, it is also very concise and statically typed. Another important feature is that it is a JVM language, the Scala compiler compiles Scala source code into Java bytecode so it is interoperable with Java.

In addition to the main API there are also suites of unit tests for the different parts of the API.

The rest of this chapter will go over some of the main modules and the structure of the Anomaly Detection and Characterization (AnoDeC) API.

5.1 Major Modules

The AnoDeC API is made up of the following major modules:

- **Data I/O**
- **Raw Data Processing**: formatting and extraction of spatio-temporal data.
- **Numerical Analysis**: approximation, smoothing, and interpolation.

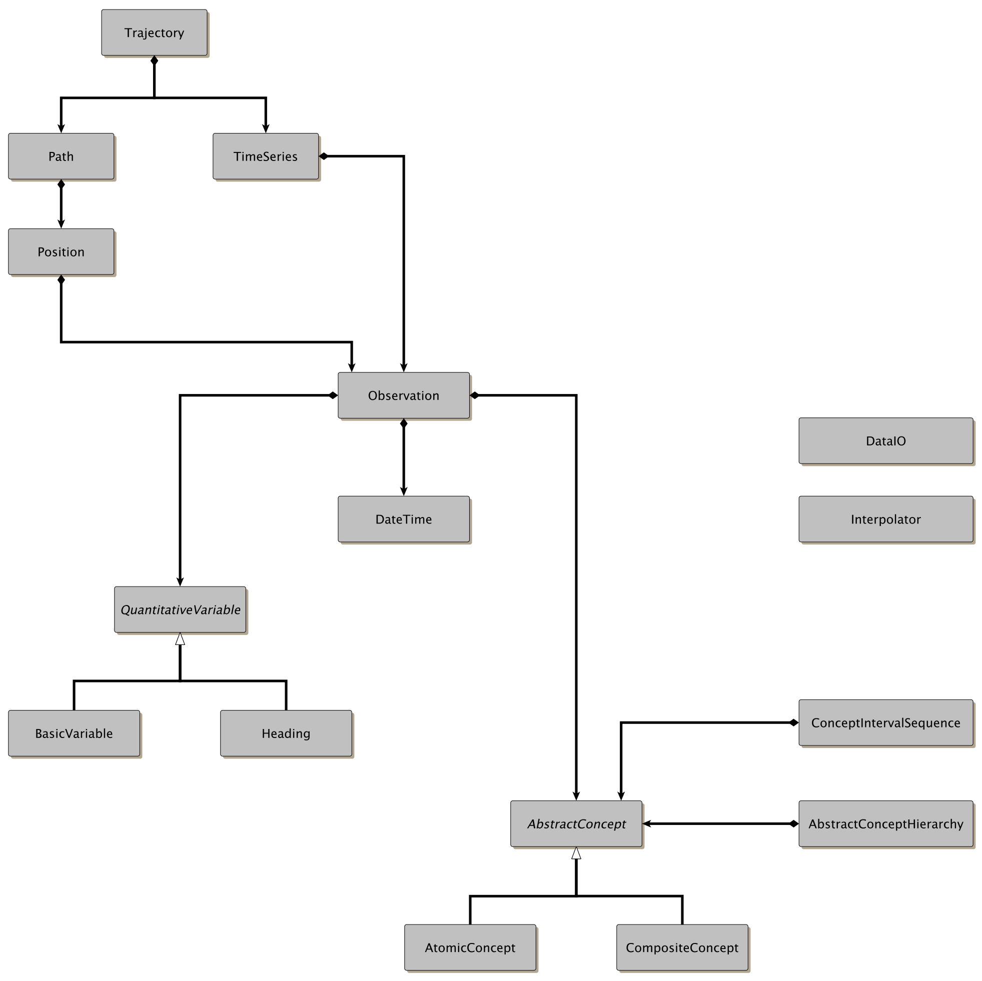


Figure 5.1: Overview of the major classes in the AnoDeC API.

- **Paths and Trajectories**: paths, trajectories, observations, quantitative variables, and time series.
- **Concepts**: abstract concepts, concept interval sequences, and concept hierarchies.
- **Representation Algorithms**: algorithms for detecting, and abstracting concepts from chapter 4.
- **Visuals**: utilities for generating 2 and 3 dimensional plots of paths and time series.
- **Helpers**: supporting auxiliary functions.

- **Convenience**

5.2 Data I/O

This module supports extracting data from various sources as well as writing formatted data to disk. When discussing I/O in the AnoDeC system we need to keep two properties of the data in mind: **source** and **format**. The **source** refers to what generated the data, i.e. the API or some other application, and the **format** refers to how the data is represented, for example, binary files, CSV in text files, XML, etc. We will distinguish the two major sources of data by referring to them as **internal** and **external**. Even though the data generated by the system is referred to as internal it is written out to text files in simple formats so it is available for other uses. Currently there is only one type of data used from external sources, and this is spatio-temporal data describing paths. It may come in many formats depending on the source (XML, CSV, formats specific to certain applications like R or Octave, etc.). All other types of data are generated by the system and so are written to and read from the same formats.

Here is a list of the different formats as well as their respective sources:

- Internal Sources
  - Path Data: formatted as simple columns one each for time x and y space separated and stored as plain text.
  - Time Series Data: Same as paths but with a single data and a single time column.
  - Concept Sequences: For external use these can be represented as string sequences.
- External Sources
  - BST: formatted as XML.
  - Arc-GIS: formatted as plain text space separated columns.
  - Other plain text data sources: formatted as simple columns separated by some delimiter (space, “,”, “:”, etc.).

Other formats can be handled easily by translating them into the basic delimiter separated column format.

5.2.1 Raw Data Processing

This is just a collection of scripts and functions that extract raw data from the various external formats and convert it to the basic space separated column format.

5.3 Numerical Analysis

This module supports approximation, smoothing, and interpolation. There are 4 types of interpolation algorithms available:

1. linear
2. spline
3. loess [6], [7]
4. Neville [29]

There is also a suite of algorithms related to Chebyshev polynomial approximation that facilitate

- Computing any Chebyshev polynomial.
- Computing roots of any Chebyshev Polynomial.
- Generating an approximation by computing the Chebyshev coefficients.
- Generating the n-degree derivative of the approximated function.

5.4 Paths and Trajectories

This module contains all the necessary classes to represent and compute with paths, trajectories, observations, quantitative variables and time series.

5.5 Concepts

This module provides all the necessary code to create abstract, atomic, and composite concepts. It also contains a number of helpful functions for handling concepts such as functions to merge concepts. Also contains the functionality to work with concept interval sequences and to build abstract concept hierarchies. This module also contains all of the definitions of the actual concepts themselves.

5.6 Representation Algorithms

All the algorithms described in chapter 4 are implemented. The bootstrapping algorithm, merging, concept abstraction, the necessary functions to create and manage concept sequences and all their accompanying quantitative data. There are also very many auxiliary functions that go along with these algorithms that make all of this possible.

5.7 Visuals

Everything to do with plotting, visualization, and animation. The main library used to facilitate visualization is the Jzy3D library [1].

5.8 Convenience

There is a Convenience class provided which wraps up a lot of the functionality of the API into a few easy to use functions. It combines a lot of the steps for handling concept sequences and raw data. It also takes care of initialization and so on. Additionally because the API itself is written in Scala the convenience class also ensures that all the necessary function calls work from Java as well. There are also some example classes that demonstrate the general use of the convenience module.

Chapter 6

Experimental

6.1 Data

We will employ families of functions to generate simulated trajectories. The aim is to generate sets of trajectories that describe the exact same kind of motion but with different variations in order to be able to test the robustness or ability of the proposed approach to identify the same features in data with variations. The specific function families will be described in detail in the appropriate experiment section.

6.2 Robustness of Representation

This set of experiments is designed to show that the automatically generated representation proposed in this work is robust in the sense that it can identify a general feature that is defined over time in a time series regardless of variations. The variations themselves would be other similar time series that describe the exact same features but exhibit quantitative variations. To do this we restrict our attention to a single variable of interest, heading. The results are relevant for any variable of interest.

For the data we make use of simulated trajectories that are generated by different function families. We will make use of 9 types of functions:

1. Sine

2. Cosine

3. Square Root
4. Quadratic
5. Cubic
6. Linear
7. Static
8. Absolute Value
9. Logistic Function

The idea for this set of experiments is to use common functions to generate paths. Using functions for this purpose is quite useful because we know exactly what will be generated and what features it will contain so we can easily test against the simulated paths. Additionally with the use of functions it becomes very simple to generate variations of the same paths, allowing us to test the robustness of our representation generator. It is important to keep in mind that even though these generated paths will appear to be identical to the two dimensional functions used to generate them they do in fact contain a third time dimension because they are paths. We will clarify this shortly with some visual examples. Once we have the paths generated for every function we also have a concept interval sequence describing the features in the paths that we will test the automatically generated concept interval sequences against. These expected concept interval sequences were defined by human visual analysis.

We ran the experiment for the 9 different functions listed above and for each function we generated 30 different variations for a total of 270 runs. The automatically generated concept interval sequences matched the expected ones with 100% accuracy. The automatically generated representations contained the correct sequence of concepts at the correct intervals of the function. What this means is that when there was an expected concept change in the path indicated by inflection points in the functions, these were correctly identified.

We will talk about the first experiment and give some details, and for the rest we will briefly outline the functions and the results. All of the functions f_i are of the form

$$a * f(x) \text{ where } a \in [1, 30] \tag{6.1}$$

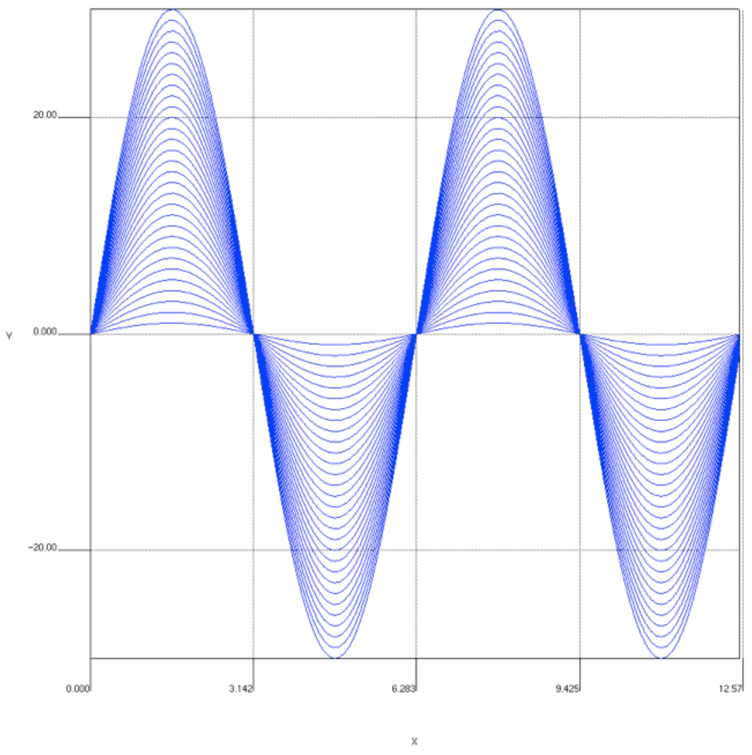
Sine
Function: $f(x) = sin(x)$
Domain: $\{0, 0.1, 0.2, \ldots, 4\pi\}$
Expected Concept Interval Sequence: [rightTurn-leftTurn-rightTurn-leftTurn]

If we look at figure 6.1(a) we can see the four different features in the path as well as the inflection points. We also see the variety of the individual features across the family of paths. What we are really trying to emphasize here is that a feature, the first right turn in the figure, should always be identified as such regardless of how sharp or shallow or long it is, regardless of variation. We have also included a view of the generated paths in three dimensions (6.1(b)) to show how the paths actually appear over time.

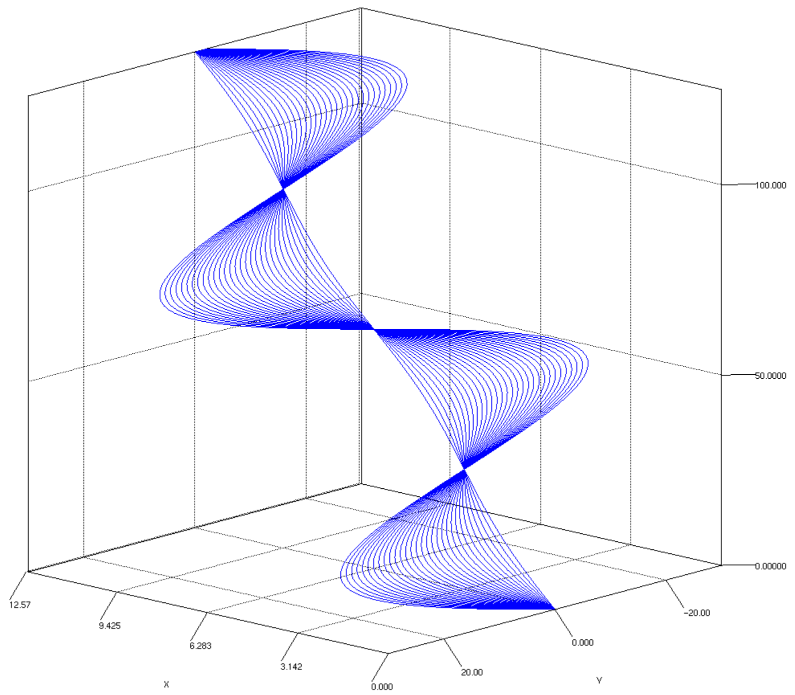
The rest of the information for this set of experiments can be seen in table 6.1 and the corresponding figures are listed in the appendix.

Function $f(x) =$	Domain	Expected CIS
$sin(x)$	$\{0, 0.1, 0.2, \ldots, 4\pi\}$	[rightTurn-leftTurn-rightTurn-leftTurn]
$cos(x)$		[rightTurn-leftTurn-rightTurn-leftTurn-rightTurn]
$\sqrt{x}$	$\{0, 0.1, 0.2, \ldots, 20\}$	[rightTurn]
x^2	$\{-20, -19.9, -19.8, \ldots, 20\}$	[leftTurn]
x^3		[rightTurn-straight-leftTurn]
x		[straight]
$constant$		[notMoving]
$\|x\|$		[straight-leftTurn-straight]
$\frac{1}{1+e^{-x}}$		[straight-leftTurn-straight-rightTurn-straight]
Results		
Automatically generated representations matched the expected ones with 0 errors.		

Table 6.1: Results of robustness experiments



(a) On the xy plane



(b) In three dimensions

Figure 6.1: View of the sine family of paths.

Table 6.1 describes all of the functions that were used to generate paths as well as the expected concept interval sequences and the results. The automatically generated concept interval sequences are not listed because they matched the expected ones with no errors and so they are the same.

6.3 ϵ-Generalization

In this set of experiments we take the same simulated paths as in section 6.2 and generate generalized paths by using the polygonal generalization algorithm.

We again performed 270 runs as in the last experiment. For all but 2 of the 9 functions the generalized paths led to the same expected results as before. The two types of simulated paths whose automatically generated representations were not the same as the expected representations were those generated by the cubic, and logistic functions. In the case of the cubic function the expected concept interval sequence was:

[rightTurn-straight-leftTurn]
the automatically generated representation was:
[rightTurn-leftTurn]

In this case we can see that the middle feature was missed when generating the representation on the polygonal generalization of the original simulated path. It appears that the reason for this is that the feature occurs over the smallest interval of time possible and thus it gets generalized away by the generalization process. The smallest time interval in this case is between two consecutive time points because the data is discrete. The smallest interval size is 0.01 (unit-less) and there are 4001 points in this particular simulated path.

For the logistic function the expected concept interval sequence was:

[straight-leftTurn-straight-rightTurn-straight]
the automatically generated representation was:
[straight-leftTurn-rightTurn-straight]

Again as in the previous case there is a gap, and the middle "straight" feature is not present. The reason for this is exactly the same as that just provided for the case of the cubic function.

The missed features is not a failure of the polygonal generalization algorithm. In fact as mentioned before the point of the generalization algorithm is to generalize away small uninteresting features while preserving the major features of the path. Of course small and major are relative terms and the size of the features preserved as well as the ones removed depends on the ϵ parameter of the generalization algorithm. In this case it just so happens that two out of the 9 function families generate a small feature in the middle of the paths that gets generalized away.

Another benefit of the polygonal generalization algorithm is that it often greatly compresses the paths it generalizes. This makes sense because the resulting generalized paths use fewer points to describe the same data. Table 6.2 lists all of the compression ratios computed as: (generalized number of points / original number of points) $\times$ 100. Additionally the ϵ parameter used for the polygonal generalization algorithm is listed.

Function $f(x) =$	Compression Ratio (%)	ϵ
$sin(x)$	3.25	2
$cos(x)$	16.22	0.5
$\sqrt{x}$	1.01	2
x^2	1.19	
x^3	11.26	
x	0.49	
$constant$	0.07	
$\|x\|$	0.46	
$\frac{1}{1+e^{-x}}$	1.24	1

Table 6.2: Polygonal generalization experiments. Compression levels and parameters.

Because all of the major features were preserved while providing significant compression, the polygonal generalization algorithm is very well suited for pre-processing spatio-temporal data before the main representation analysis is performed.

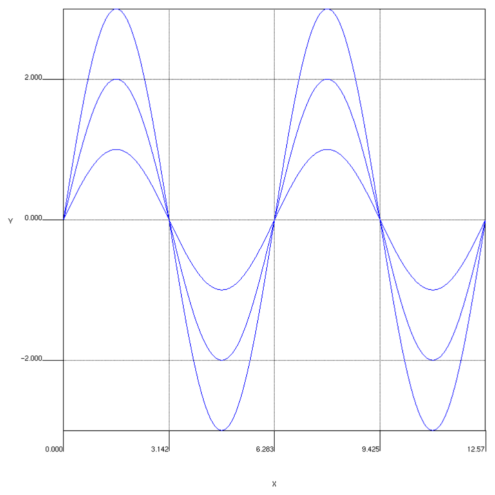
6.4 Noise, Trajectory Generalization, and Smoothing

In this set of experiments we repeat the experiments performed in section 6.2 but with randomly introduced noise. We then apply the same automated analysis as in the previous experiments except with the addition of smoothing and generalization algorithms. We will use both linear and nested smoothing algorithms and the ϵ-generalization algorithm; all of which were introduced in chapter 4. We also apply a filtering function to remove any changes in heading of less than some threshold degrees. The threshold selection is highly dependent on what size features one is interested in detecting.

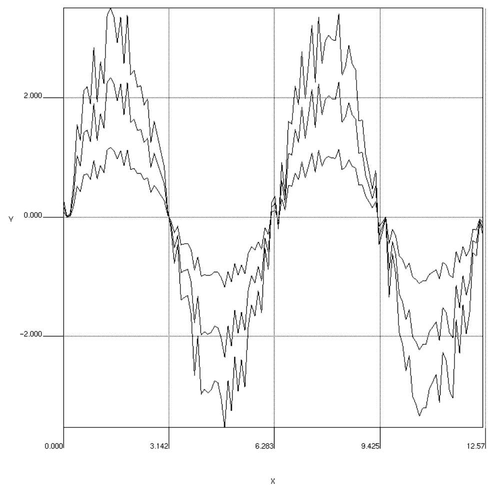
So the objective was to try to detect the same features we did in the first experiment except to see if that can be done when there are significant amounts of noise introduced into the paths. We will first go over the experiments for the sine curves.

Sine
Function: $f(x) = sin(x)$
Domain: $\{0, 0.1, 0.2, \ldots, 4\pi\}$
Expected Concept Interval Sequence: [rightTurn-leftTurn-rightTurn-leftTurn]

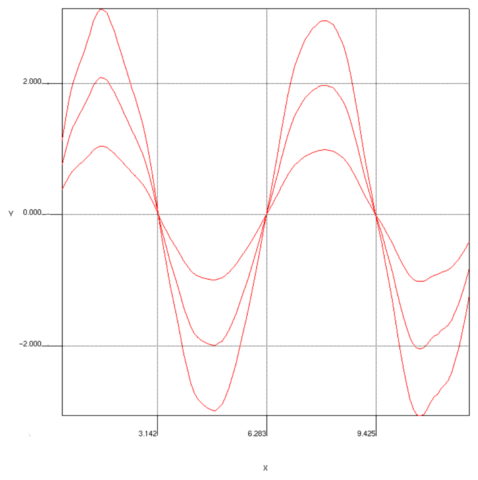
We start with the clean simulated trajectories as shown in figure 6.2(a) and to this we introduce randomly generated noise which then gives us the noisy path shown in figure 6.2(b). The next step is to smooth the data using linear or nested generalization, figure 6.2(c). And finally our last pre-processing step is to perform a polygonal generalization on the result from our smoothing step, figure 6.2(d). We need to set two different parameters, NS which is the nested generalization parameter, and ϵ which is the parameter for the polygonal generalization algorithm. This information is presented in table 6.3 along with compression ratios and errors. The errors are the number of paths out of 30 that were analyzed for each function family, whose automatically generated representation did not match the expected one presented in the first experiment 6.2. We also tried making use of the linear generalization algorithm for smoothing but the results were completely negative so we only list information pertaining to the nested version of the algorithm. This is because of the large noise levels introduced into the paths. We have also omitted the constant function family because the results would never be those expected which is a single "notMoving" feature. This is because no matter how much the noise is smoothed or generalized away we do not end up with all positions of the path being exactly the same so we would never get



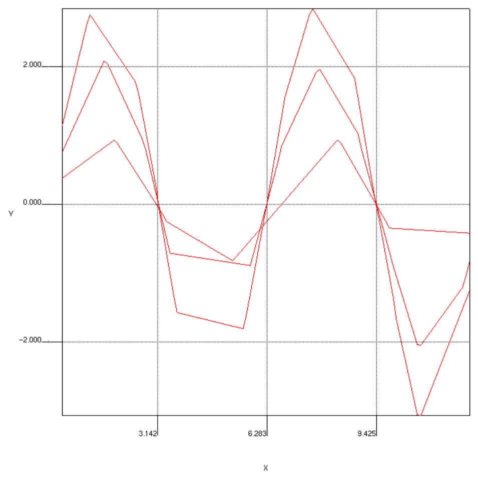
(a) Clean simulated paths.



(b) Noisy paths.



(c) Nested generalization smoothing.



(d) Polygonal approximation.

Figure 6.2: Clean vs noisy generated sine paths.

the representation we expect from a constant function.

For this experiment as explained in section 6.3 we had the same missing middle “straight” features in the cubic and logistic function families. However, we did not count these as errors, for the reasons given in that experiment, so they are not included in the counts presented in table 6.3.

The nature of most of the errors is not that the resulting representations derived from

Function $f(x) =$	Compression Ratio (%)	ϵ	NS	errors out of 30
$sin(x)$	49.47	1.0	5	0
$cos(x)$	54.31	0.8	6	0
$\sqrt{x}$	2.85	4	40	2
x^2	2.99			0
x^3	12.89			0
x	1.01			0
$\|x\|$	8.23	0.7		0
$\frac{1}{1+e^{-x}}$	15.04		45	1

Table 6.3: Polygonal generalization experiments. Compression levels and parameters.

the noisy paths miss important features detected in the clean paths, it is that new features, often small in magnitude and time are inserted. Of course most of these are removed by the smoothing and generalization steps however some remain. Additionally the use of the polygonal generalization algorithm also has a tendency to introduce "straight" segments in paths breaking up larger features. This is an artifact of the way polygonal generalization works. Despite this the major features are still detected and properly represented. We can usually handle these insertions by filtering out features that have a magnitude falling below a certain threshold.

One of the challenges of dealing with noisy inputs is deciding what parameters to utilize for smoothing (NS) and generalization (ϵ). For this particular experiment we ended up choosing those parameters that recaptured the main features of the clean simulated paths. In this case we had an expected set of features for each simulated path and we wanted to show that it is possible even with significant amounts of noise to still identify those features and suppress all of that noise. Parameter selection is highly dependent on the size of the features we want to detect and how many of the smaller features we wish to generalize away. This often depends on the domain. There is a spectrum of options ranging from detecting and representing every single feature of a path to generalizing the path to the point where it is just defined by a line between two points. Often the desired level of smoothing and generalization lies somewhere in between those two ends of the spectrum.

Chapter 7

Conclusion

This work has attempted to explore the area of anomaly detection in spatio-temporal data, particularly in the maritime domain. We have seen that upon further investigation there is more than the problem of anomaly detection to tackle. Whenever we think about anomaly detection in the maritime domain we must now also consider anomaly characterization. Of all of the available techniques applied so far in this field the statistical ones with their two step approach of building a model of normal behaviour and checking new observations against it, are the most widespread. As mentioned in chapter 2 there are other approaches as well, but they all fall into the two major categories of data-driven and model-driven approaches.

The main point we identified as potentially yielding some of the greatest benefits in both the areas of detection, and characterization, was that essentially the complexity and variety of anomalies that one can detect is limited by the representation/feature set, the popular one, being a four dimensional feature set made up of position and velocity. In the case of detection a richer feature set would enable the detection of more complex and interesting behaviour. As for the case of characterization, an improved representation would allow an intuitive and simple way to describe any manner of motion behaviour patterns to search for; whether it be very simple things such as a right turn or a much more complex behaviour like shuttling. We attempted to address this point by proposing a framework for representing quantitative time series data, and specifically targeting it to spatio-temporal data. The focus of the framework was to describe behaviour over time in terms of abstract concepts and compositions of concepts drawn from concept hierarchies. In addition to conceptually and formally defining the representation framework we also designed and implemented a substantial software API that covers all of the presented algorithms and a lot of additional functionality in order to facilitate exploration and experimentation. Using the API we have shown that the proposed representation is in fact very robust, and intuitive for facilitating the concise description of a huge variety of behaviours. Furthermore, we have also taken an existing and vaguely defined path generalization algorithm and described in detail the steps and techniques necessary to actually implement it. We have also shown that when the representation framework is paired with a smoothing algorithm, and the generalization algorithm, it is quite resistant to noise; and furthermore this combined process actually gives the user more control over the detail of the behaviour detected. With this combined approach one can represent every single detail of a ship voyage, or generalize it to a few major features even when the voyage lasts several days, is hundreds of kilometers long and contains thousands of data points.

There is still a lot of work to be done. The necessity to control feature granularity led to the introduction of smoothing and generalization algorithms. However, this brought to light the challenges of parameter selection for these algorithms. It would be interesting to explore the possibility of an automated method for sensible parameter selection. There is also work to be done on the ϵ-generalization algorithm. It would be interesting to develop and test a three(or more) dimensional version of the algorithm. Another considerable improvement would be the use fuzzy logic to turn the abstract concepts that have crisp quantitative constraints into fuzzy sets. This would open the door to a host of possibilities. It would also be very beneficial to collaborate with a domain expert to really work on defining broad and deep concept hierarchies for the maritime domain. Finally, the entire community would benefit from a gold standard data set containing curated real data divided into test and training sets ready for experimentation. We believe this is sorely needed and would really move the field ahead not by just providing real data, but by also acting as a benchmark against which new ideas can be tested. Finally it would be very interesting to use the presented representation approach as the feature set for some of the existing, bottom-up, anomaly detection techniques. Even though this work seemingly opens up a multitude of new questions we see this as a positive side effect that opens up new directions for this line of research.

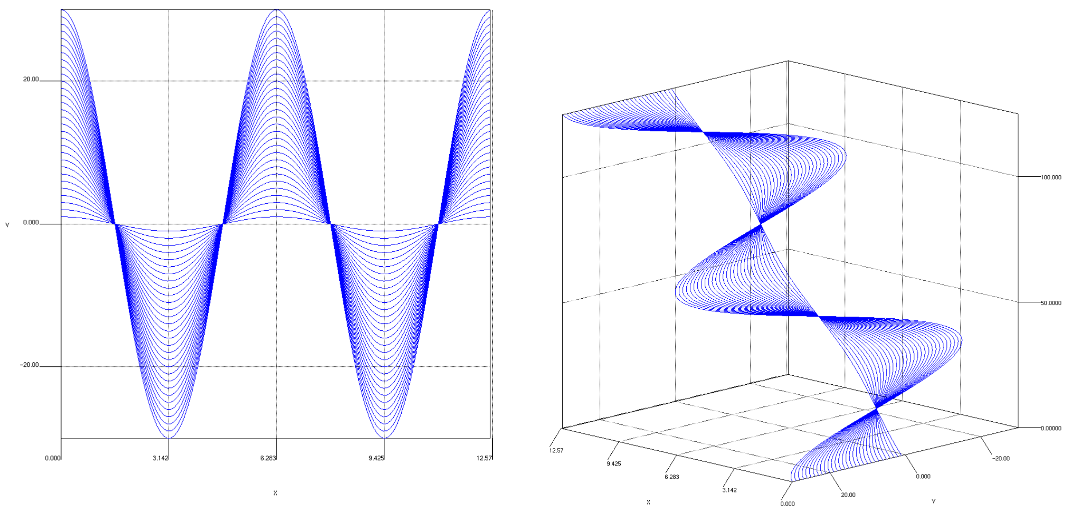
Ultimately the best approach seems to be a hybrid of the bottom-up and top-down approaches. In an ideal situation one would like to be able to detect predefined anomalies, and detect new anomalies and characterize them. However, there may also be great value in identifying what *is* ordinary and using that in order to lower the burden on a human analyst. This is definitely not an easy or a small problem to "solve". There are many interacting factors and ultimately there is a significant human element involved as well; we are not just trying to determine what kind of behaviour ships are taking part in, we are also trying to understand what that behaviour ultimately says about the goals and intent of the humans that are actually in control. There is no neatly package solution to be had, even if there was our efforts would be at best asymptotic. However, there definitely are steps in the right direction and we think we have made some with this work. We've opened up the feature set available to bottom-up approaches, and introduced a new model-driven approach for the maritime anomaly detection field. We feel confident that this work as well as the tools created as a consequence of it will enable further exploration and experimentation necessary for taking some of the next steps in the right direction.

Appendix A

Additional Figures and Tables

A.1 Robustness of Representation: Figures

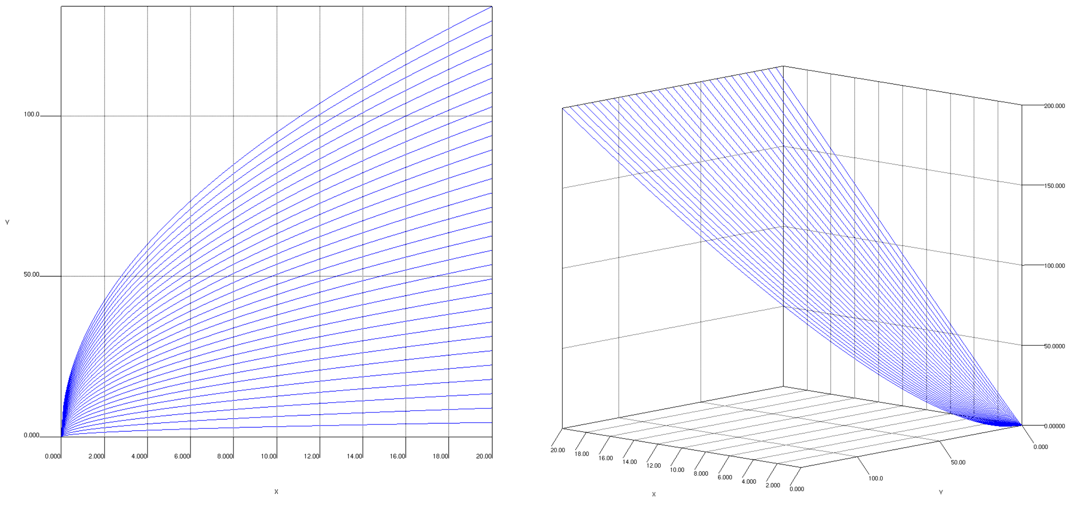
A.2 Example Tables



(a) On the xy plane

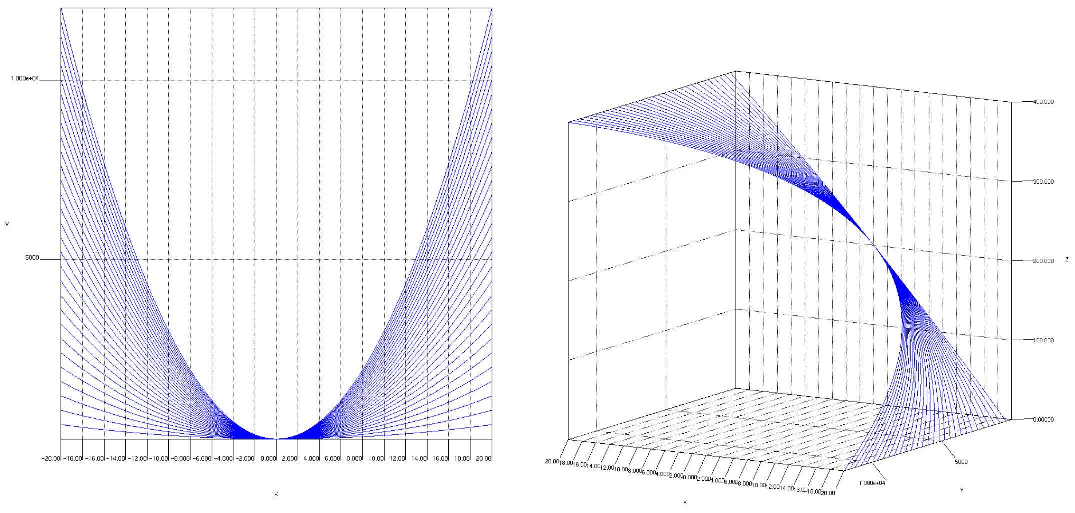
(b) In three dimensions

Figure A.1: Cosine family of paths.



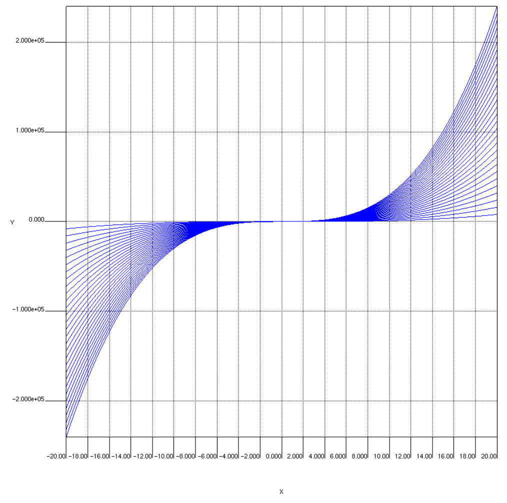
(a) On the xy plane (b) In three dimensions

Figure A.2: Square root family of paths.

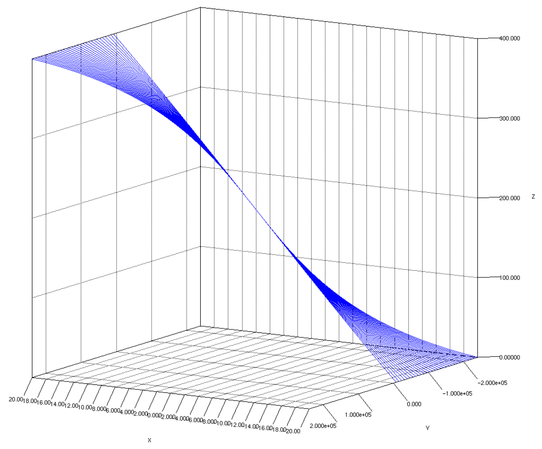


(a) On the xy plane (b) In three dimensions

Figure A.3: Quadratic family of paths.

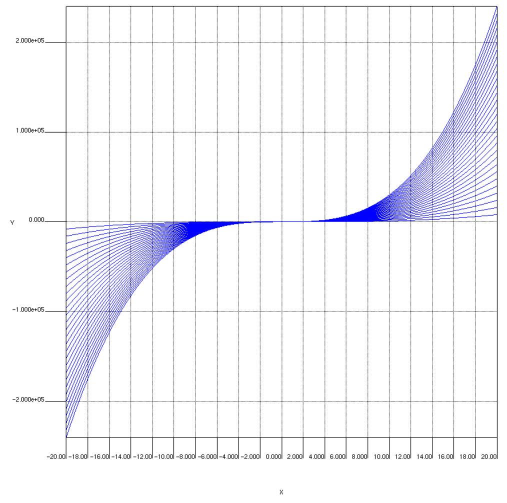


(a) On the xy plane

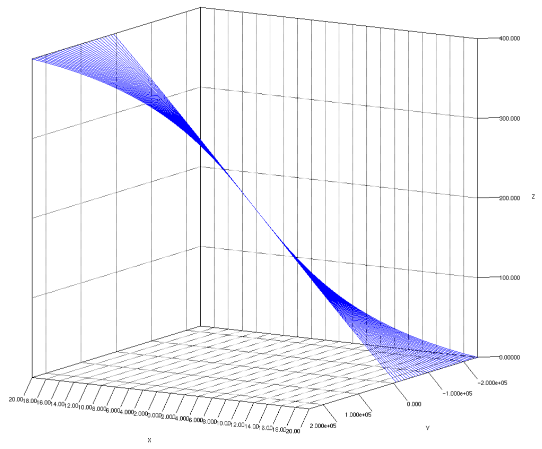


(b) In three dimensions

Figure A.4: Cubic family of paths.



(a) On the xy plane



(b) In three dimensions

Figure A.5: Linear family of paths.

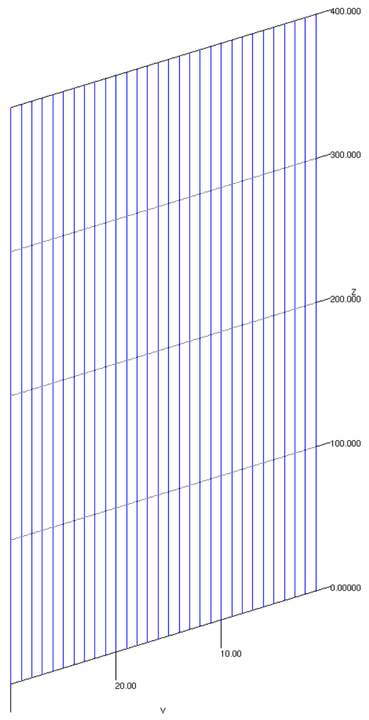
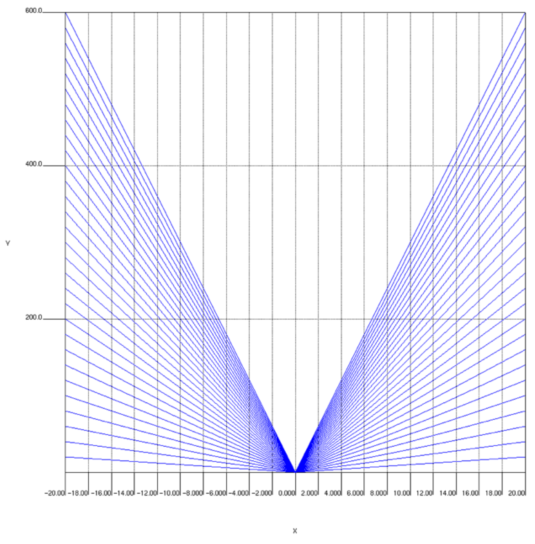
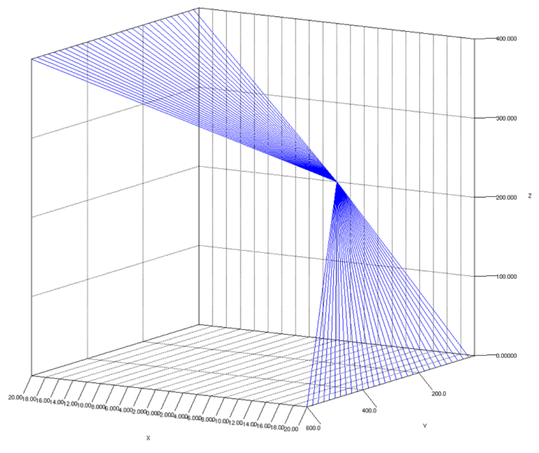


Figure A.6: Constant family of paths in three dimensions. These paths all fall within a plane because they describe a static position over time.

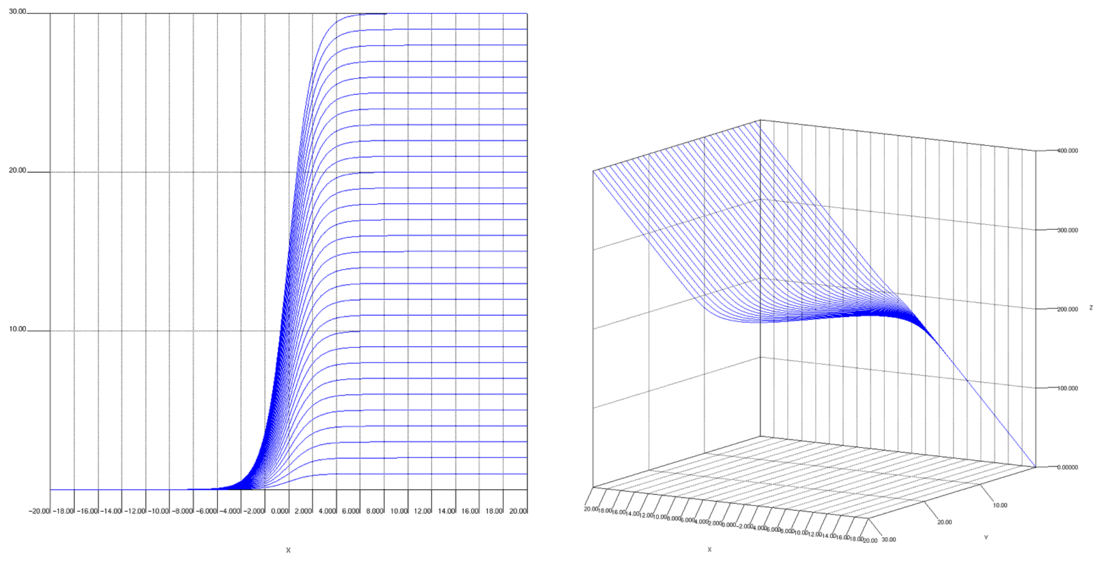


(a) On the xy plane



(b) In three dimensions

Figure A.7: Absolute value family of paths.



(a) On the xy plane (b) In three dimensions

Figure A.8: Logistic family of paths.

Table A.1: Table containing features in example ship voyage representation.

Concept	Heading Change (degrees)	Time Interval (minutes)
notMoving	-Infinity	0.00:522.49
rightTurn	13.63	522.49:524.50
leftTurn	-6.29	524.50:526.48
straightI	0.57	526.48:616.48
notMoving	-Infinity	616.48:4501.49
straightD	0.18	4501.49:4579.48
leftTurn	-8.51	4579.48:4581.49
rightTurn	18.53	4581.49:4582.48
leftTurn	-124.93	4582.48:4586.00
rightTurn	7.38	4586.00:4587.50
leftTurn	-122.18	4587.50:4588.49
rightTurn	28.89	4588.49:4590.50

Continued on next page

Table A.1 – continued from previous page

Concept	Heading Change (degrees)	Time Interval (minutes)
notMoving	-Infinity	4590.50:4646.49
rightTurn	19.93	4646.49:4654.49
straightD	-1.65	4654.49:4660.49
rightTurn	3.42	4660.49:4663.49
straightD	-2.62	4663.49:4672.48
rightUTurn	175.91	4672.48:4677.50
straightD	0.37	4677.50:4701.48
rightTurn	4.41	4701.48:4703.50
straightD	-1.92	4703.50:4750.49
leftUTurn	-183.18	4750.49:4755.49
rightTurn	6.68	4755.49:4757.98
straightD	-0.10	4757.98:4763.49
leftTurn	-6.53	4763.49:4769.49
straightI	1.28	4769.49:4811.49
rightUTurn	178.71	4811.49:4815.50
straightD	-0.66	4815.50:4817.99
rightTurn	3.69	4817.99:4820.48
straightD	-1.26	4820.48:4821.49
rightTurn	3.62	4821.49:4825.50
straightD	-1.43	4825.50:4884.49
leftUTurn	-181.47	4884.49:4892.49
straightI	0.47	4892.49:4935.49
rightUTurn	179.93	4935.49:4942.48
straightD	1.19	4942.48:4959.00
rightTurn	15.80	4959.00:4961.49
straightD	-0.02	4961.49:4963.50
leftTurn	-15.44	4963.50:4966.49
notMoving	-Infinity	4966.49:4969.49
leftTurn	-6.54	4969.49:4970.49
notMoving	-Infinity	4970.49:4974.49

Continued on next page

Table A.1 – continued from previous page

Concept	Heading Change (degrees)	Time Interval (minutes)
rightTurn	395.51	4974.49:4982.49
notMoving	-Infinity	4982.49:5086.49
leftTurn	-5.47	5086.49:5088.98
straightI	-0.89	5088.98:5110.49
leftTurn	-4.36	5110.49:5118.49
straightI	-0.57	5118.49:5127.50
rightTurn	6.90	5127.50:5129.49
straightD	-1.56	5129.49:5132.48
rightTurn	56.81	5132.48:5139.49
leftTurn	-35.57	5139.49:5144.49
rightTurn	303.47	5144.49:5154.49
leftTurn	-15.72	5154.49:5155.49
rightTurn	130.81	5155.49:5160.49
leftTurn	-39.47	5160.49:5161.48
notMoving	-Infinity	5161.48:5173.49
rightTurn	52.76	5173.49:5175.50
notMoving	-Infinity	5175.50:5225.48
rightTurn	206.42	5225.48:5231.49
leftTurn	-15.32	5231.49:5232.99
notMoving	-Infinity	5232.99:5235.48
leftTurn	-27.71	5235.48:5236.49
notMoving	-Infinity	5236.49:5244.49
rightTurn	64.68	5244.49:5245.49
notMoving	-Infinity	5245.49:5472.49
leftTurn	-22.76	5472.49:5473.49
rightTurn	21.61	5473.49:5476.48
leftTurn	-3.83	5476.48:5478.49
straightI	-0.30	5478.49:5482.48
rightTurn	26.92	5482.48:5486.49
leftTurn	-4.67	5486.49:5487.50

Continued on next page

Table A.1 – continued from previous page

Concept	Heading Change (degrees)	Time Interval (minutes)
rightTurn	34.87	5487.50:5489.99
leftTurn	-5.72	5489.99:5491.49
notMoving	-Infinity	5491.49:5493.49
rightTurn	6.61	5493.49:5494.49
notMoving	-Infinity	5494.49:5543.49
leftUTurn	-180.02	5543.49:5551.98
straightI	0.22	5551.98:5563.99
leftTurn	-3.41	5563.99:5569.48
straightI	-0.66	5569.48:5580.50
leftTurn	-5.17	5580.50:5583.98
straightI	0.11	5583.98:5587.50
leftTurn	-26.13	5587.50:5594.49
rightTurn	5.16	5594.49:5595.48
leftTurn	-99.44	5595.48:5596.49
notMoving	-Infinity	5596.49:5613.49
rightTurn	9.87	5613.49:5614.48
leftUTurn	-188.98	5614.48:5618.49
rightTurn	7.85	5618.49:5622.99
straightD	2.37	5622.99:5703.00

Concept	Heading Change (degrees)	Time Interval (minutes)
notMoving	-Infinity	0.00:507.49
straightD	0.31	507.49:546.50
rightTurn	3.00	546.50:586.49
notMoving	-Infinity	586.49:4492.48
straightI	1.71	4492.48:4524.99
leftTurn	-14.08	4524.99:4575.49
rightUTurn	186.28	4575.49:4695.49
leftUTurn	-182.99	4695.49:4766.49
rightUTurn	183.15	4766.49:4835.49
leftUTurn	-180.48	4835.49:4893.49
straightI	1.15	4893.49:4907.49
leftUTurn	-173.26	4907.49:4929.50
rightTurn	2.26	4929.50:4951.49
leftTurn	-34.77	4951.49:5097.49
rightTurn	13.24	5097.49:5128.99
notMoving	-Infinity	5128.99:5457.49
rightTurn	13.58	5457.49:5475.49
leftTurn	-340.17	5475.49:5606.49
rightTurn	2.01	5606.49:5632.00
straightD	-0.74	5632.00:5660.00

Table A.2: Table containing features in example ship voyage representation after smoothing and generalization.

Bibliography

[1] Jzy3d library. `http://jzy3d.org/`.

[2] The scala language. `http://www.scala-lang.org/`.

[3] Vladimir Avram, Uwe Glässer, and Hamed Yaghoubi Shahir. Anomaly detection in spatiotemporal data in the maritime domain. In *ISI*, pages 147–149, 2012.

[4] Santiago Iglesias Baniela. Piracy at sea: Somalia an area of great concern. *The Journal of Navigation*, 63(02):191–206, 2010.

[5] Natural Resources Canada. The atlas of canada – coastline and shoreline [online]., 2011. Last visited, May 2011.

[6] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[7] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, September 1988.

[8] M. R. Endsley. Theoretical underpinnings of situation awareness: A critical review. In M. R. Endsley and D. J. Garland, editors, *Situation Awareness Analysis and Measurement*. LEA, 2000.

[9] Roozbeh Farahbod, Vladimir Avram, Uwe Glässer, and Adel Guitouni. Engineering situation analysis decision support systems. In *EISIC*, pages 10–18, 2011.

[10] Roozbeh Farahbod, Vladimir Avram, Uwe Glässer, and Adel Guitouni. A formal engineering approach to high-level design of situation analysis decision support systems. In *ICFEM*, pages 211–226, 2011.

[11] Robert Fisher (editor). Kernel density estimators. CVonline, `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MISHRA/kde.html`.

[12] Douglas M. Hawkins. *Identification of outliers*. Chapman and Hall, London ; New York :, 1980.

[13] Randall D. Knight. *"Physics for Scientists and Engineers: A Strategic Approach"*, volume 1. Addison-Wesley, 2nd edition, 2008.

[14] R.O. Lane, D.A. Nevell, S.D. Hayward, and T.W. Beaney. Maritime anomaly detection and threat assessment. *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–8, 2008.

[15] R. Laxhammar. Anomaly detection for sea surveillance. *Information Fusion, 2008 11th International Conference on*, pages 1–8, 2008.

[16] R. Laxhammar, G. Falkman, and E. Sviestins. Anomaly detection in sea traffic - A comparison of the Gaussian Mixture Model and the Kernel Density Estimator. *2009. FUSION '09. 12th International Conference on Information Fusion*, pages 756–763, 2009.

[17] Rikard Laxhammar and Göran Falkman. *Conformal prediction for distribution-independent anomaly detection in streaming vessel data.* StreamKDD '10. ACM Press, New York, New York, USA, 2010.

[18] E. Martineau and J. Roy. Maritime anomaly detection: Domain introduction and review of selected literature. Technical report, Defence R & D Canada – Valcartier, 2011.

[19] Steven Mascaro, KB Korb, and A.E. Nicholson. Learning Abnormal Vessel Behaviour from AIS Data with Bayesian Networks at Two Time Scales. *Tracks A Journal Of Artists Writings*, 2010.

[20] P. Monzini. Migrant smuggling via maritime routes. 2004.

[21] David S. Moore and George P. McCabe. *"Introduction to the Practice of Statistics"*. W. H. Freeman, New York, 5th edition, 2006.

[22] Alexandra Musto, Klaus Stein, Andreas Eisenkolb, Thomas Rfer, Wilfried Brauer, and Kerstin Schill. From motion observation to qualitative motion representation. In Christian Freksa, Christopher Habel, Wilfried Brauer, and Karl Wender, editors, *Spatial Cognition II*, volume 1849 of *Lecture Notes in Computer Science*, pages 115–126. Springer Berlin / Heidelberg, 2000.

[23] Alexandra Musto, Klaus Stein, Kerstin Schill, Andreas Eisenkolb, and Wilfried Brauer. Qualitative motion representation in egocentric and allocentric frames of reference. In Christian Freksa and David Mark, editors, *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science*, volume 1661 of *Lecture Notes in Computer Science*, pages 749–749. Springer Berlin / Heidelberg, 1999.

[24] B Ristic, B La Scala, M Morelande, and N Gordon. Statistical Analysis of Motion Patterns in AIS Data : Anomaly Detection and Motion Prediction. pages 40–46.

[25] Jean Roy. Automated reasoning for maritime anomaly detection.

[26] J.P. Shim, Merrill Warkentin, James F. Courtney, Daniel J. Power, Ramesh Sharda, and Christer Carlsson. Past, present, and future of decision support technology. *Decision Support Systems*, 33(2):111 – 126, 2002.

[27] Cesar Souza. Gaussian mixture models and expectation-maximization, October 2010. http://crsouza.blogspot.com/2010/10/gaussian-mixture-models-and-expectation.html.

[28] David Vieites, Sandra Nieto-Romn, Antonio Palanca, Xavier Ferrer, and Miguel Vences. European atlantic: the hottest oil spill hotspot worldwide. *Naturwissenschaften*, 91:535–538, 2004. 10.1007/s00114-004-0572-2.

[29] Eric W. Weisstein. Neville's algorithm. MathWorld–A Wolfram Web Resource, `http://mathworld.wolfram.com/NevillesAlgorithm.html`.

[30] Eric W. Weisstein. Point-line distance–2-dimensional. MathWorld–A Wolfram Web Resource, `http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html`.