Accurate Prediction of Maritime Trajectories From Historical AIS Data Using Grid-Based Methods

ACCURATE PREDICTION OF MARITIME TRAJECTORIES FROM HISTORICAL AIS DATA USING GRID-BASED METHODS

BY

PAUL WILSON, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2017) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Accurate Prediction of Maritime Trajectories From Historical AIS Data Using Grid-Based Methods
AUTHOR:	Paul Wilson B.Eng., (Electrical Engineering) McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. T. Kirubarajan
NUMBER OF PAGES:	xi, 60

To my parents, for all their love and support over the course of my life.

Abstract

In order to aid prediction of future maritime vessel trajectories, it is useful to examine historical vessel information. It is mandatory for large maritime vessels to broadcast, among other fields, spatial, speed, and course information using Automatic Identification System (AIS) transponders. By processing a large historical dataset, it is possible to predict future vessel trajectories. The region of interest is discretized into a grid. Then, using offline computations, the historical data are used to determine second-order transition probabilities and speed information. Predictions will be carried out as an online process. If the destination is known, Dijkstra's Algorithm is used to predict the vessel's path. If the destination is not known, a path can still be determined using transition probabilities, but the prediction will be less accurate. The path is then smoothed using a line of sight algorithm to produce more realistic paths. Finally, the speed information is used to predict travel times. Real data were used to build the graph structure, and predictions were judged against real trajectories.

Acknowledgments

I would first like to thank my supervisor, Dr. Kirubarajan, for his support, guidance, and encouragement. I would also like to thank Dr. Tharmarasa for offering feedback and helping me with my code. I don't think I could have completed this work without his help. I also thank the ECE department's graduate administrative assistant Cherly Gies for all the work she does and her great sense of humour. I would also like to thank Dr. Bruce and Dr. Jeremic for sitting on my defense committee and offering helpful comments on my thesis. Many thanks to all of my friends in the ECE department. Particular thanks go to Krishanth Krishnan for his input on my work.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Maritime transport accounts for roughly 80% of global trade by volume [27]. This high volume of maritime activity makes maritime situational awareness and surveillance important areas of interest. This thesis will leverage historical data to create long-term vessel motion predictions.

Automatic Identification System (AIS) is a monitoring system designed for maritime vessels, for the purposes of surveillance and reducing ship collisions [2]. According to the International Convention for the Safety of Life at Sea (SOLAS) [16], use of AIS is mandatory for ships above 300 gross tonnage on international voyages, cargo ships above 500 gross tonnage not involved in international voyages, and all passenger ships, while smaller ships may optionally use AIS. Because of these regulatory requirements, use of AIS is widespread, creating a wealth of data that can be used to improve long-term predictions.

An AIS transmission contains many pieces of information: the Maritime Mobile Service Identity (MMSI), which serves as a unique identifier for each vessel; position, in latitude and longitude; Speed Over Ground (SOG), in knots (nautical miles per

hour); Course Over Ground (COG), in degrees clockwise from North; destination; Estimated Time of Arrival (ETA); etc. All of these fields are innacurate to some degree. The accuracy of the position, SOG, and COG vary depending on the equipment of the vessel in question. A previous study by Harati-Mokhtari et al.[14] found that, when an AIS field is in error, in 80 to 85% of instances, human error is to blame. This makes ETA particularly challenging, because it is often based on the navigator's best guess. Furthermore, ETA is an optional field, so in many cases it is simply blank or may even be set to the ETA of a previous voyage. Harati-Mokhtari et al. found that 49% of sampled AIS messages contained obviously erroneous ETAs or destination information, such as times in the past, numbers instead of port names, blank fields, etc. In at least one case they found a destination reading "to Hell". Since the destination and ETA fields are entered manually, they can be considered highly unreliable.

1.1 Motivation and Problem Statement

AIS messages can be received by a multitude of devices, including coastal stations, aircraft, and satellites. Only coastal stations provide near-continuous coverage, however, their range is limited. Satellites are the predominant source of AIS messages. Due to the sheer size of maritime regions, satellites still cannot provide full coverage [25]. For the case of tracking vessel movements, reliance on AIS will obviously lead to significant gaps in vessel movements. Therefore, it is desirable to create an algorithm that could predict where vessels will travel throughout these coverage gaps. This is the problem that forms the basis of this thesis.

The assumed input to the problem is an AIS message containing position, speed over ground, course over ground and time information. The destination may or

may not be provided. The output is the predicted vessel track, including position, velocity, and time information. A slightly different method of prediction is used if the destination is not provided. The expectation is that providing the destination will yield better results. Before making actual predictions, traffic patterns must first be learned. In order to accomplish this, a database of historical AIS messages is processed.

1.2 Previous Work

There have been several different approaches to similar problems. [23] proposed a method they named Traffic Route Extraction and Anomaly Detection (TREAD) for classification and anomaly detection. TREAD works by extracting waypoints from each vessel trajectory within a bounding box. These waypoints are stationary points, entry points and exit points. The waypoints are linked to form paths. Then these waypoints are clustered together using Density-Based Spatial Clustering of Applications with Noise (DBSCAN). DBSCAN [9] clusters points that are close together and discards points in low density regions. TREAD is effective, but is computationally expensive.

A grid based approach to predict future behaviour was attempted in [3]. An outstar learning law to determine weights between nodes. The issue with this method is that it does not scale well to larger regions, and it was used for very short prediction windows. Their paper presented results for just 15 minutes in the future.

A grid based approach was also adopted by [37] to determine high density regions. In this work, a region of interest was discretized into a grid to determine traffic patterns and identify high density areas. The scope of the paper did not, however,

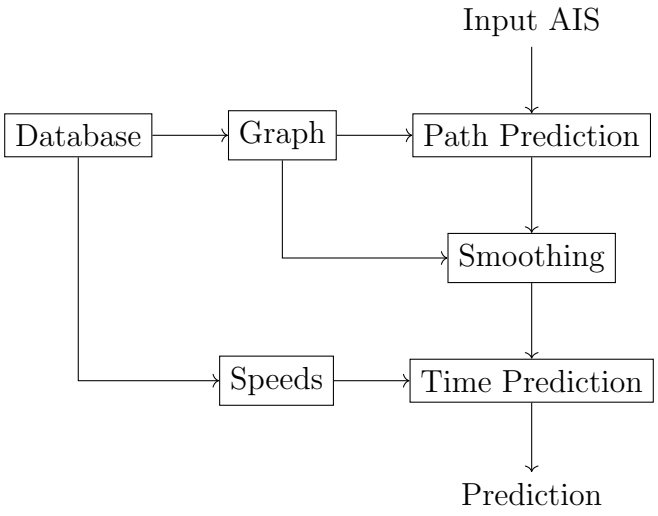

Figure 1.1: Shortest path given destination layout

include vessel prediction.

Ground vehicle prediction using GPS rather than AIS data is a related problem. An algorithm called Sub-Trajectory Synthesis (SubSyn) was created for predicting paths based on historical GPS data [32]. This algorithm makes use of a Markov Model to predict likely paths and destinations vehicles will take. This algorithm was demonstrated using historical GPS data from taxis in Beijing.

1.3 Proposed Approach

This thesis uses grid-based and shortest path planning methods from graph theory to predict the path and arrival time of a specific vessel, based on its starting point, course, and speed as well as its destination, if it is known.

The process is illustrated in Figure 1.1. A database of historical AIS messages is mined, first to determine which regions of an area are frequently traveled. Then, this information is used to create a directed, weighted graph. It is also used to determine

the expected speed in each region. A shortest path algorithm is applied to the graph in order to predict the path the vessel will follow. Then the path is smoothed. Finally, the speed information is used to predict how fast the vessel will traverse this path, and thus the times along the path and the expected time of arrival.

1.4 Contribution and Significance

To compare runtime complexity, O-notation (commonly pronounced as "big-oh" notation) will be used [4]. The purpose of O-notation is to represent an upper bound of the growth rate of the runtime depending on the size of the input. Formally, $f(n) = O(g(n))$ holds if and only if there exists positive constants c and n_0 such that:

$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \tag{1.1}$$

To the best of this author's knowledge, shortest path methods are a novel approach to this problem. For example, [23] performed route predictions by clustering similar vessel routes. Each route was defined by a series of physical locations, and clustered based on location, speed and vessel type using DBSCAN. Unfortunately, DBSCAN has a high runtime of $O(n \log n)$ [9], with n being individual trajectories in this case. On the other hand, the method presented in this thesis requires only a runtime of $O(n)$ in the data processing stage. Also note that this is not a particularly well-studied problem. The authors of [23] focused on anomaly detection, while [3] looked at shorter term predictions.

1.5 Organization of the Thesis

This thesis is divided into five chapters. Chapter 2 discusses graph theory and various methods used to find the shortest path between two nodes. Chapter 3 presents the proposed approach to constructing a graph based on an AIS data-set and using it to predict a vessel's motion and arrival time, both with and without a given destination. Chapter 4 presents results and discussion. Chapter 5 consists of conclusions and future work.

Chapter 2

Graph Theory and Shortest Path Planning Methods

In this thesis, a directed graph is constructed and used to construct the shortest path between the source and destination nodes. Good knowledge of graph theory and shortest path planning is required to understand this process. So in this chapter, some relevant background material pertaining to graph theory and shortest path planning will be presented. There are a variety of potential shortest path planning algorithms that may be used, so a selection of algorithms will be examined with a focus on their respective advantages and disadvantages for this problem.

2.1 Graph Theory

According to the explanation found in [12], a graph is a structure containing two sets: a set of nodes and a set of edges. This is denoted by $G = (N, E)$. Nodes are denoted by n_i. A connection between nodes is called an edge, and if two nodes are connected,

they are called neighbors. An edge between nodes n_i and n_j is denoted by $e_{i,j}$. An unweighted graph is a graph in which the edges are all binary. In other words, the nodes are either neighbors or they are not, and travel between the two nodes is either possible or not. A weighted graph will be considered in this thesis. It is a graph in which each edge has an associated weight. For edge $e_{i,j}$, the associated weight will be $w_{i,j}$. The weights can be considered as the cost of travel. The cost may or may not be related to the physical distance between the two nodes. For example, consider a graph representing a road map. Most cars would tend to prioritize the fastest route, so a freeway would likely have a lower cost than a city street, even if the city street had a shorter distance. Unconnected nodes, or nodes that are not neighbors, may be thought of as having a cost of ∞.

The direction of travel may also be important. An undirected graph is one in which the direction of travel results in the same weight between nodes. This concept is illustrated in Figure 2.1. In an undirected graph, the cost of travel between nodes is considered the same regardless of direction. As its name implies, a directed graph is a graph in which the weight between nodes may be different depending on the direction of travel. This is illustrated in Figure 2.2. In the case of maritime shipping routes, direction turns out to have a big impact, so directed graphs will be used in this thesis.

2.1.1 Second Order Dependencies

The problem with using graph structures, as shown in Figure 2.2, is that no history is preserved. In the case of maritime vessels, at a given node there is information about which node the vessel is likely to transition to but no information about where the

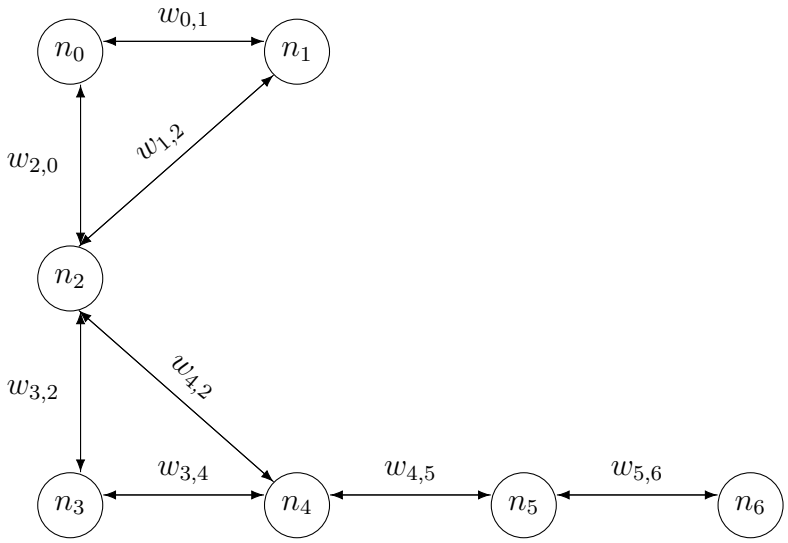

Figure 2.1: Illustration of an undirected graph

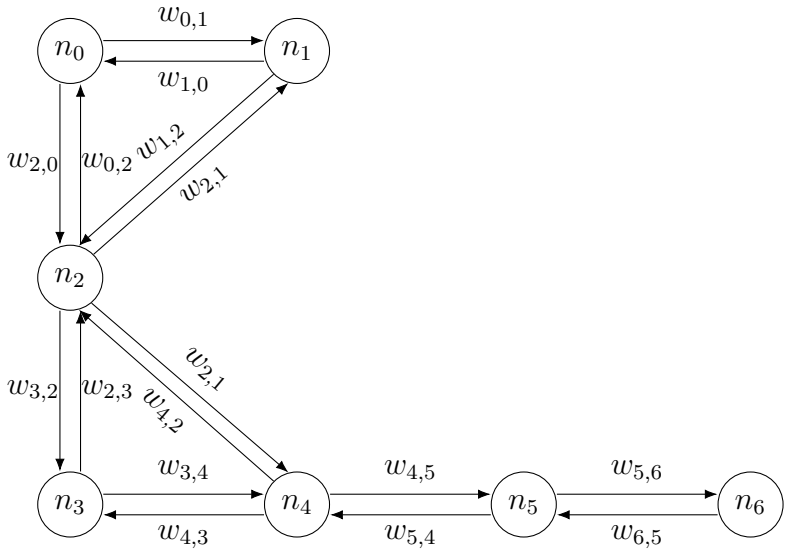

Figure 2.2: Illustration of a directed graph

vessel is coming from. In other words, the system is memoryless. This is problematic because shipping lanes may intersect. So it is necessary to construct a graph in which the nodes do not simply represent an object's current spatial location, but the location it is coming from at a previous time step. Graphs with higher order dependencies has been used to produce better results on various types of network related tasks [31]. Specifically, improvements have been demonstrated on random walk simulations between shipping ports, using clustering to identify shipping ports, and ranking websites.

Therefore, rather than using first order dependencies to only represent a ship's current location, second order dependencies are used in order to also represent a ship's location at the previous time-step. In other words, second order dependencies add memory to the system. Similar to the method used in [31], the difference between first and second order dependencies is illustrated in Figures 2.3 and 2.4. In memory-less structures, as in Figure 2.3, a physical location has a one-to-one relationship with a node in the graph structure. In structures with memory, as in Figure 2.4, one physical location is mapped to multiple nodes.

The main component of the edge weights are the transition probabilities between nodes. If the location of a vessel at time t is represented as a random variable X_t, the probability of transitioning from node i to node j at time step $t+1$ is given as:

$$P(X_{t+1} = i_{t+1} | X_t = i_t) = \frac{W(i_t \rightarrow i_{t+1})}{\sum_j W(i_t \rightarrow j)} \tag{2.1}$$

A second order model does not just consider the current node i, but also considers the previous location h. So the current *state* will be $i|h$. Similar to Equation 2.1, if the current state at time t is $i|h$, the probability of transitioning to node j in time

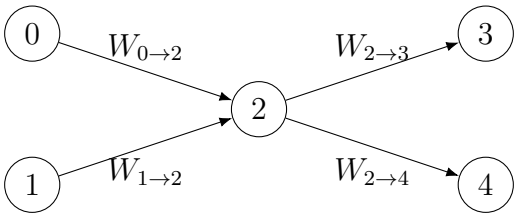

Figure 2.3: Illustration of a memory-less system

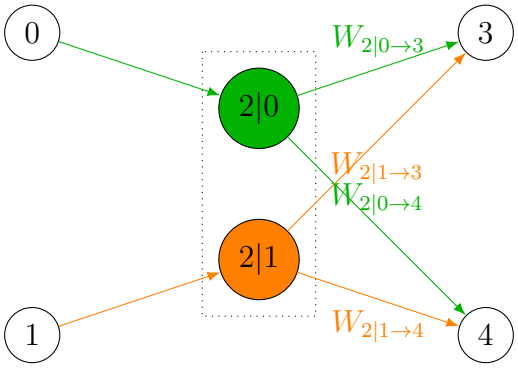

Figure 2.4: Illustration of a system with memory

step $t+1$ is given as:

$$P(X_{i+1} = j | X_t = i | h) = \frac{W(i|h \to j)}{\sum_k W(i|h \to k)} \tag{2.2}$$

2.2 Shortest Path Problem

The shortest path problem is finding the shortest path between two points in a graph. The starting node is called the source, s, and the end node is called the target, t. For the purposes of this thesis, it is somewhat misleading to talk about the shortest path, since the path with the shortest distance is not always the desired path. Rather, the desired path is the one with the lowest cost. The cost is determined based on historical traffic density. However, since the literature uses the term shortest path, this will be the terminology used in this thesis.

In this thesis, a weighted, directional graph is used. Weighted, because historical traffic densities are used. Either directed or undirected graphs could have been used, but weighted graphs were chosen because it was observed that ships travel differently depending on direction of travel. Therefore different routes will be generated based on direction of travel.

2.2.1 Dijkstra's Algorithm

Dijkstra's Algorithm is one of the earliest shortest path algorithms[8], but is still very relevant to this day [17], [26], [33]. Many other shortest path algorithms build upon Dijkstra's Algorithm.

Intuitively, Dijkstra's Algorithm works by traversing the graph, marking nodes as they are visited, starting with s. The nodes that have been visited are kept track of, as well as each node's distance, d, from s. Initially, s is set to the current node and its distance is marked as 0. All other nodes are set to unvisited and set to a distance of ∞. On each iteration, the unvisited node with the smallest distance is set as the current node. Then, looking at the unvisited neighbors of the current node, tentative distances are calculated as the sum of the current distance and the edge weight, e, to the neighbor. Each neighbor's distance is set to the smaller of its current distance and the tentative distance:

$$d_{neighbor} = \min(d_{neighbor}, d_{current} + e_{current \to neighbor}) \tag{2.3}$$

The current node is then set to visited before moving on to the unvisited node with the smallest distance. The algorithm terminates when t has been visited or when all tentative distances have been found to be ∞. In the latter case, this indicates t is

unreachable from s. The pseudocode of this algorithm is detailed in Algorithm 1.

Algorithm 1 Dijkstra's algorithm

Input: s - source node t - target node

```
1: for each node n in graph do
2:     dist[n] ← ∞
3:     prev[n] ← NULL
4:     add n to unvisited
5: end for
6: dist[s] ← 0
7: while TRUE do
8:     v ← n in unvisited with min dist[n]
9:     if v = t then
10:        return prev[v]
11:    end if
12:    remove v from unvisited
13:    for each neighbour n of v do
14:        altDist ← dist[v] + length(n, v)
15:        if altDist < dist[n] then
16:            dist[n] ← altDist
17:            prev[n] ← v
18:        end if
19:    end for
20: end while
```

2.2.2 Bidirectional Dijkstra's Algorithm

It can be shown that the runtime of Dijkstra's Algorithm is $O(n \log n)$ on average, where n is the number of nodes in the generated path. It is possible to significantly speed this up without sacrificing quality of the path, using a bidirectional search method [21]. In essence, this algorithm consists of alternating between forward and backward searches running Dijkstra's Algorithms. So the forward distances, d_f, and backward distances, d_b, must be kept track of. The forward search starts at s and moves towards t, while the backwards search starts at t and moves towards s. Note

that because the graph is directed, the edge weights for the backward search must be reversed. Execution is terminated when the same node, x, has been visited by both searches. However, x may not actually be in the shortest path. In order to find the actual shortest path, the node y that minimizes the total resulting distance will be found:

$$\arg\min_{y}[d_f(y) + d_b(y)] \tag{2.4}$$

Then the forward path $s \to y$ is simply concatenated with the reverse path $y \to t$.

Bidirectional Dijkstra's Algorithm usually, though not always, results in visiting significantly fewer total nodes. On average, this leads to a runtime of $O(\sqrt{n}\log n)$, which is an improvement from the $O(n \log n)$ average runtime of the regular Dijkstra's Algorithm.

2.2.3 A* Algorithm

The A* Algorithm [15] is another extension to Dijkstra's Algorithm. It can be much faster than Dijkstra's Algorithm, although it will not always produce the shortest path. It is widely used in many applications [5], [10], [19], [36]. It functions in much the same way as Dijkstra's Algorithm, the main difference being that A* calculates the cost of a node, n, in a different fashion. Dijkstra's Algorithm merely takes the cost of n as the cost of the path coming from s. A* incorporates a heuristic function to find an estimate of the cost of the path from $n \to t$:

$$f(n) = g(n) + h(n) \tag{2.5}$$

where $f(n)$ is the cost of n, when coming from s and going towards t. $g(n)$ is the cost of the path coming from s. $h(n)$ is the estimate of the cost of the path $n \rightarrow t$. Everything else proceeds the same as Dijkstra's Algorithm. $h(n)$ is a heuristic function that estimates the cost of the remainder of the path. The difficulty lies in the choice of $h(n)$. Note that if $h(n) = 0$, then the algorithm is the same as Dijkstra's Algorithm. In other words, it will find the shortest path, and will not result in a performance increase. The ideal case is when $h(n)$ is the true cost of $n \rightarrow t$. In this case, the shortest path will be found, and it will be found much faster than with naive Dijkstra. In practice, however, it is difficult to choose $h(n)$ in such a way. Thankfully, one can still get significant improvements even when $h(n)$ is not optimal. [15] proved that as long as $h(n)$ remains less than the actual cost of the shortest path from $n \rightarrow t$, A* will produce the shortest path. In this case, $h(n)$ is called admissible. Furthermore, in order to guarantee that no node will be expanded twice, $h(n)$ should be monotonic. That is, for all adjacent nodes i and j, $h(i) \leq d(i, j) + h(j)$. If $h(n)$ is allowed to be greater than the actual cost of the shortest path, then it will result in a significant speed-up but may not always result in the shortest path.

A suitable heuristic function is possible in many cases, but it is not very useful if it does not have a constant runtime. In video game pathfinding, the A* may use a euclidean heuristic [5], that is a heuristic based on the euclidean distance between points. The edge weights in this application are more complicated, so a simple euclidean distance heuristic will do little good.

2.2.4 Bellman-Ford Algorithm

This section will refer to the description of the Bellman-Ford Algorithm found in [4]. Unlike Dijkstra's Algorithm, the Bellman-Ford Algorithm can deal with negative edge weights, and will detect if there exists a path with a negative edge cyle. A negative edge cycle indicates no solution. Allowing negative edge weights means that it is a much more flexible algorithm, in the sense that it can be used for a greater variety of applications with less modification. The downside is that Bellman-Ford is much slower than Dijkstra's Algorithm. Dijkstra's Algorithm has a runtime of $O(n \log n)$, compared to Bellman-Ford's runtime of $O(e \cdot n)$, where n is the number of nodes and e is the number of edges.

Bellman-Ford is also a much simpler algorithm to implement. First, the source distance is set to 0, and all other distances to ∞. Then, each node is scanned and updated if its neighbour's distance to the source is smaller. Each update corresponds to incrementing the shortest path length. Therefore, the loop must be repeated $n-1$ times, since the longest possible shortest path will go through every node. Finally, all edges are scanned once more and if an update occurs, this means a negative edge cycle has been found. This process is detailed in Algorithm 2.

2.2.5 Genetic Algorithm

Genetic algorithms are inspired by biological evolution [30]. The data structures involved are made to resemble chromosomes. These algorithms are initialized with a series of randomized solutions to the problem. Then the solutions are typically recombined, evaluated with a fitness function where the solutions with the lowest fitness are removed. The remaining solutions are randomly crossed with each to create

Algorithm 2 Bellman-Ford algorithm

Input: N - nodes W - edge weights s - source

for each vertex $v \in N$ **do**
    distance$[v] = \infty$
    predecessor$[v] = NULL$
end for
distance$[s] = 0$
for $i = 1$ to N.size $- 1$ **do**
    for each edge$(u, v) \in E$ **do**
        if distance$[v] >$ distance$[u] + E(u, v)$ **then**
            distance$[v] =$ distance$[u] + E(u, v)$
            predecessor$[v] = u$
        end if
    end for
end for
for each edge$(u, v) \in E$ **do**
    if distance$[v] >$ distance$[u] + E(u, v)$ **then**
        return FALSE
    end if
end for
return TRUE

new solution. This continues for a set number of iterations or until an acceptable solution has been found. One problem with genetic algorithms is the solution(s) found can be biased if one is not careful implementing the initialization step. They also use few assumptions about the problem, which means they can often be outperformed by other optimization methods.

Genetic algorithms have been previously adapted for use in the shortest path problem [38]. First a set of initial solutions are found. This stage cannot be fully random. It must begin with node s, end at node t, with all intermediate nodes sharing edges. So, the path initially contains only node s. Then the path is populated with a random neighbor of the end-node until t is reached. Note that a node cannot be repeated in a path. If a repeated path is found, or if all of the neighboring nodes have been visited before reaching t, the path is re-initialized.

To decide which solutions to propagate, genetic algorithms determine the fitness of a solution. To accomplish this, a fitness function is used. The fitness function for an individual *ind*, representing the path P is given as:

$$\text{fitness(ind)} = [\sum_{(u,v)\in P} E(u,v)]^{-1} \tag{2.6}$$

This is calculated for all paths, and the paths with the lowest fitness are eliminated.

This step is the core of genetic algorithms. Genetic algorithms produce new solutions by mixing two solutions together. This process is called crossover and simulates crossing over chromosomes in biological reproduction. For the purposes of shortest path applications, two paths containing common nodes are chosen randomly as parent nodes for crossover. These nodes are represented by the sum of sub-paths:

$$\begin{aligned} P_1(s,t) &= P_1(s, n_{\text{common}}) + P_1(n_{\text{common}} + t) \\ P_2(s,t) &= P_2(s, n_{\text{common}}) + P_2(n_{\text{common}} + t) \end{aligned} \tag{2.7}$$

Then two child paths are generated by mixing the sub-paths of the parent paths:

$$\begin{aligned} P_3(s,t) &= P_1(s, n_{\text{common}}) + P_2(n_{\text{common}}, t) \\ P_4(s,t) &= P_2(s, n_{\text{common}}) + P_1(n_{\text{common}}, t) \end{aligned} \tag{2.8}$$

2.3 Algorithm Selection

Any of these methods can be suitable choices. Genetic algorithms have proven to be computationally expensive [18], but they may provide multiple optimal paths, since a different result may be generated at each iteration. However, this will be a large increase in computational complexity, as well as being much more difficult to implement than other shortest path algorithms. For the purposes of this thesis, one path will suffice, so these are not good trade-offs. The Bellman-Ford algorithm is computationally expensive, but can handle negative edge weights. This could be advantageous for this thesis, since the goal is to prioritize high edge weights, while using shortest path algorithms. But as it will be seen in Section 3.2, there is a simpler way to handle negative edge weights in this case. Since probabilities are being used, $1 - p$ can be used in order to treat high probabilities as low costs. The A* algorithm results in a significant speed-up, but a poor choice of heuristic can result in a sub-optimal path, so this choice will be avoided. For bidirectional Dijkstra, the difficulty

comes in determining which node to merge the two paths. If an incorrect node is chosen, a sub-optimal path may be returned.

Dijkstra's Algorithm, however, is a greedy algorithm that has been proven to find an optimal path [4]. So this algorithm will be chosen, but note that other shortest path algorithms, such as those mentioned in the previous sections, could be substituted, if sufficient attention is paid to the trade-offs. There is not much that can be done to improve the computational complexity costs of the genetic algorithm or Bellman-Ford. However, the sub-optimal concerns in A* and bidirectional Dijkstra could be overcome, given enough care. This thesis will focus on the end results more than the computational complexity. Dijkstra's Algorithm will therefore be chosen to guarantee optimal paths.

Chapter 3

Methodology

Now that the background knowledge for graph theory and the shortest path problem has been established, it can be put into practice to predict maritime vessel trajectories and arrival times. First, the graph structure is constructed from historical AIS and landmass data in Sections 3.1 and 3.2. Then the speed graph is constructed in Section 3.3. Path Prediction is performed in Section 3.4, and the path is smoothed in Section 3.5. Finally, the Speed Graph is used to generate speed and arrival time predictions in Section 3.6.

3.1 AIS Processing

The available AIS data are contained in a PostgreSQL database. First, the database is queried for AIS messages within the region of interest. Then, the messages are separated based on the MMSI values, and ordered by the timestamp in ascending order. This allows the AIS messages to be represented as vessel tracks. As in [23], a vessel track, $\mathbf{V}$, can be represented as a series of state vectors $\mathbf{v}_i$ across T time steps:

$$\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_T\} \tag{3.1}$$

Where $i = 1, 2, ..., T$ is the index of each track at each time step up to T. The state vector, $\mathbf{v}_i$ is defined by its position component, (φ, λ), and velocity component, (s, θ) to form a four-dimensional state vector:

$$\mathbf{v}_i = [\varphi_i, \lambda_i, s, \theta]^T \tag{3.2}$$

Where φ and λ are latitude and longitude, respectively. s is speed, in knots (nautical miles / hour). θ is the course, in degrees clockwise from North.

3.2 Graph Structure

In preparation to construct the graph, the region of interest is discretized into a grid of $L \times H$ cells. Each node in the graph represents a cell in the grid, and nodes are considered neighbors if they are spatially adjacent. The region of interest is denoted by R. For each cell m in region R, let $N = \{n_1, n_2, ..., n_k\}$ denote the neighborhood of m. That is, the surrounding cells that are reachable from m. For the purposes of this thesis, each cell is considered to have eight neighbors, in order to simplify implementation. This is illustrated in Figure 3.1. In principle, there could be more neighbors for each cell, and the cells are not required to be uniform in size, though this is not the case in this thesis. The more cells there are, the higher the resolution of the grid and the more accurate the predicted paths should be. However, more cells will obviously increase computation time.

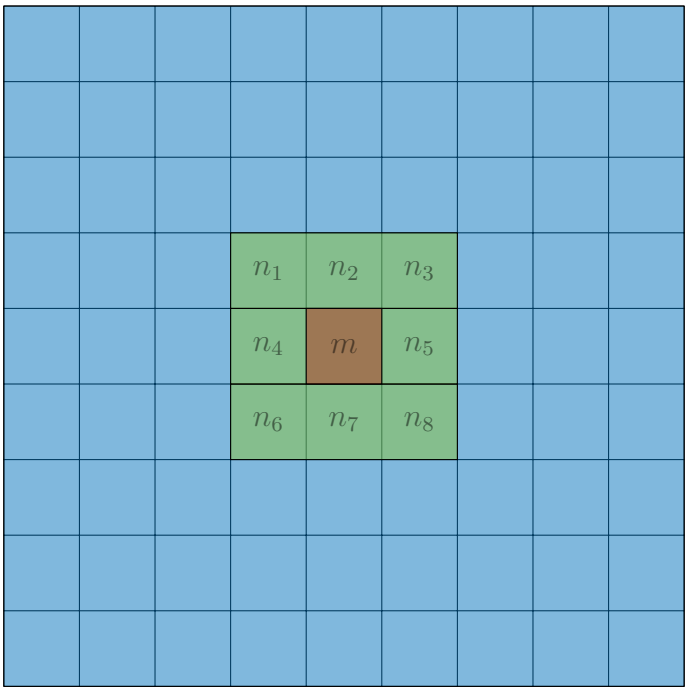

Figure 3.1: Neighborhood of cell m

3.2.1 Transition Probabilities

If only first order transitions are considered, each node in the graph would directly map to a cell in the grid mentioned above, and edges would be constructed between neighboring cells. An edge weight from position i to position j would be denoted as $W(i \to j)$. However, this work considers second order transitions are considered, in a method similar to [31]. This will be more complicated than the first order case. In the second order case, each node in the graph is denoted as $n_{i|j}$, that is position i given that the trajectory is coming from position j. So, the edge weights are given by $W(i|j \to k)$.

Based on the vessel tracks found in Section 3.1, transition counts, $c_{i|j \to k}$ can be determined. j can be found directly if there are two consecutive messages in neighboring cells.

Otherwise, it is found based on the course information for the first message found in i. Given the initial position $i = (\varphi_i, \lambda_i)$, the course from North θ, and the Radius of the Earth R, then the previous position, $j = (\varphi_j, \lambda_j)$, can be estimated using a variation of the Haversine formula [7].

$$\varphi_j = \arcsin(\sin(\varphi_i \cdot \cos\delta + \cos\varphi_i \cdot \sin\delta \cdot \cos(\theta - \pi)) \tag{3.3}$$

$$\lambda_j = \lambda_i + \arctan2(\sin(\theta - \pi) \cdot \sin\delta \cdot \cos\varphi_i, \cos\delta - \sin\varphi_i \cdot \sin\varphi_j) \tag{3.4}$$

$\theta - \pi$ is the reverse course, since the location the vessel is coming from is desired. δ is the *angular distance* found using d/R. d is the distance traveled, which is taken as the distance between the respective centers of two grid cells.

From there, it is straightforward enough to detect a transition between two cells if there are two consecutive messages in neighboring cells, and increment $c_{i|j\to k}$ accordingly. Unfortunately, due to the variable update rate of AIS transponders[14], updates may occur in non-neighboring cells. For example, some vessels send updates an hour apart. For cells with diagonal lengths of 5 nautical miles and a vessel traveling in a straight line at 20 knots, the updates would come 4 cells apart. This a common enough scenario that a substantial amount of information would be thrown away if only transitions between adjacent cells were accepted. Therefore, linear interpolation is used to determine intermediate transitions when non-neighboring updates are observed. This is not a completely accurate solution, as a vessel may perform some maneuvers between the updates. However, it is far worse to have a sparse amount of data in the graph. The amount of total vessels in each node, $c_{i|j}$, is also determined.

Once transition counts have been obtained, transition probabilities are found as:

$$P(i|j \rightarrow k) = \frac{c_{i|j \rightarrow k}}{\sum_t c_{i|j \rightarrow t}} \tag{3.5}$$

3.2.2 Cost Function

Once the transition probabilities have been found, they can be used to determine the costs to be used as the edge weights. The cost is meant to be unit-less. It is a weighted sum of two components: the probability cost and the time cost. The motivation for the probability cost is fairly obvious: the less likely path should have a higher cost. The time cost is meant to give some penalty to paths that are possible but do not make physical sense, for example, a path that traverses a long distance when a short one is available. This is rare, but it can occasionally occur when the time cost is not incorporated. The cost is calculated as:

$$C_{i|j \rightarrow k} = \alpha \cdot [1 - P(i|j \rightarrow k)] + (1 - \alpha) \frac{d_{i,k}}{v_i} \frac{v_{\min}}{d_{\max}} \tag{3.6}$$

where α is a weighting factor, $d_{i,k}$ is the distance between i and k, $d_{\max}$ is the maximum distance between two cells, v_i is the average speed in i, and $v_{\min}$ is the minimum average speed in a cell. $1 - P(i|j \rightarrow k)$ is used because the goal is to have a higher cost for a less likely path. $P(i|j \rightarrow k)$ is in the range $[0, 1]$, so it can be safely subtracted from 1.

In order to determine the time cost, time is determined through dividing distance by speed. In order to find a unit-less value, a ratio is formed between $\frac{d_{i,k}}{v_i}$ and a maximum time value. The maximum time value is found by taking the maximum possible distance between two cells, $d_{\max}$, and a minimum speed, $v_{\min}$, that is the

minimum speed for a vessel to be considered to be in a cruising motion. In this thesis, the minimum speed is set to 5 knots. This is somewhat arbitrary. The goal is to make the quantity unit-less, so the exact value is unimportant.

α is used to determine how to weigh the probability and time costs, and will be in the range $[0, 1]$. It makes sense to weigh the probability cost higher, since there are many situations where the physically shorter path may not be the most likely path. For example, a hazardous region may be a faster route that is uncommon, so α should normally be above 0.5. For this thesis, 0.8 was used so as to give most of the weight to the probability.

3.2.3 Landmass Avoidance

For the most part, this graph will avoid landmasses without further modification, since vessels will not send out AIS updates from land. Therefore cells enclosed by a landmass should have zero density. However, due to AIS errors the graph may sometimes contain non-zero densities within landmasses. Also, due to AIS data sparsity in some regions, some actual routes may contain cells with zero density values. So zero density cells should not be excluded from the path planning algorithm. Otherwise unsolvable paths may be found. With this scheme, it is possible to have paths passing through land.

In order to solve this, shapefiles are used. Shapefiles were designed by the Environmental Systems Research Institute. Their detailed specifications can be found in [1]. The shapefiles were obtained courtesy of the GSHHG database [29]. Landmasses are extracted from the shapefiles. These landmasses are treated as polygons defined by points, which form boundaries of the landmasses. In order to determine if a cell

is within a landmass, the ray-crossing test described in [13] and [11] is used to determine if a cell lies within a polygon. The implementation used in this thesis is given in Appendix A. If a cell is fully contained in a landmass, it is marked as being within land, and will not be considered to be a neighbor of any other cell. In this way, cells within landmasses will be excluded from the path-planning algorithm.

3.3 Speed Graph

Next, a structure describing the speed of ships in each region is constructed. Similar to Section 3.2, the region of interest is discretized into a grid of M cells, and the time into T intervals. Note that because the density grid and the speed grids are separate data structures, they do not have to be the same size or discretize the region in the same fashion. In other words, the M value in this section is not necessarily the same as the M in Section 3.2.

The speed distribution of a sample cell is shown in Figure 3.2. The distribution is assumed to be Gaussian, even though it is clearly not exactly Gaussian. Therefore, the mean, μ, and the variance, σ^2 in each grid cell will be calculated as in [34]. Given the number of vessels, n, and vessel speeds, v_i:

$$\mu = \frac{\sum_{i \in R} v_i}{n} \tag{3.7}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} (v_i - \mu)^2}{n - 1} \tag{3.8}$$

However, it is preferable to have an online method of calculating variance to provide a system that can adapt to new information. So, the following iteratives

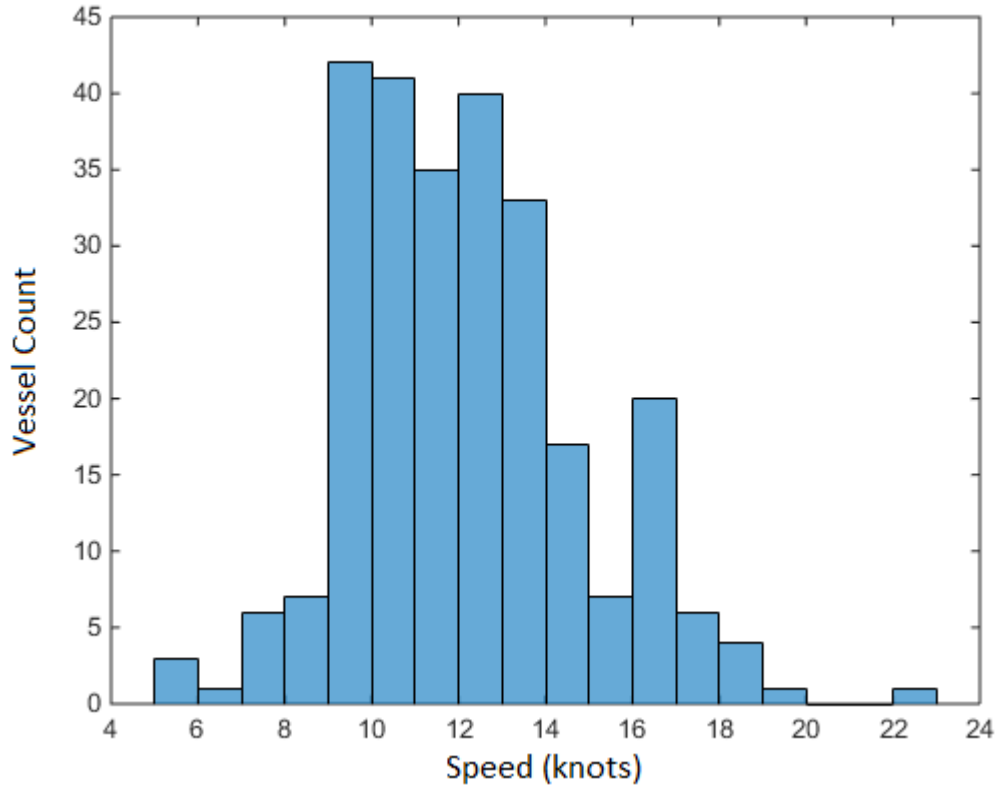

Figure 3.2: Speed distribution in a single grid cell

estimates from [20] are used:

$$\mu_i = \mu_{i-1} + \frac{v_i - \mu_{i-1}}{i} \tag{3.9}$$

$$\sigma_i^2 = \frac{(i-1)\sigma_{i-1}^2 + (v_i - \mu_{i-1})(v_i - \mu_i}{i} \tag{3.10}$$

3.4 Path Prediction

Now that the transition and speed information has been extracted from the historical AIS data, a prediction can be generated given an AIS message. The required fields for the AIS message are the position, speed, and course over ground. The initial course and speed are taken to be more reliable than the predicted path, so, initially, the track is propagated forward in a straight line, depending on the starting course and speed. This is done for an initial propagation time $t_{\text{propogate}}$, or until land is encountered, based on the landmass information found in Section 3.2.3. Note that since cells determined to fall within landmasses were previously marked, there is no need to once again access the shapefiles. This is a big advantage, since iterating through all the polygons within the shapefiles is computationally expensive.

3.4.1 Path Prediction Using Dijkstra's Algorithm

If a destination is not provided, Dijkstra's Algorithm, as found in Section 2.2.1, can be used to generate a path. As described, Dijkstra's Algorithm should work with only a minor modification. Dijkstra's Algorithm normally exits when the target node has been reached. In this case, there is no single target node. For target position t, without direction considerations, Dijkstra's Algorithm would exit when t is reached. Because second order transitions are considered instead, it will exit when $t|i$ is reached, for any node i.

As a note for implementation, in line 9, finding the node with the minimum distance can be a significant source of inefficiency depending on implementation. If the distances are simply stored in an unsorted array, this leads to a worse case runtime of $O(N^2+E)$, since the entire array has to be examined to find the minimum

value. However, if a min-heap is used instead, the runtime can be further reduced to $O(N \log N + E)$ [4].

It should be noted that Dijkstra's Algorithm could be substituted with any of the other shortest path algorithms outlined in Chapter 2, with similar results. The other algorithms should give optimal or worse results. Note that there is not necessarily a single optimal path. The best substitutes would be Bidirectional Dijkstra and A* Search, because they would be computationally cheaper. The main concern is producing lower quality paths. This can be minimized with careful modifications to the algorithms, but since this thesis is more focused on the end result and less on efficiency, Dijkstra's Algorithm was used. In this way, optimal paths are guaranteed.

Algorithm 3 Path prediction with destination

Input: start, end

1: path ← dijkstra(start, end)
2: smooth(path)
3: track ← predictTime(path) **return** track

3.4.2 Path Prediction with Unknown Destination

If a destination is not provided, a prediction can still be made. However, the destination is a valuable piece of information, so the expectation should be that predictions without the destination will have significantly higher error. Dijkstra, and other shortest path algorithms, cannot be used here because they require the destination as an input. An alternate algorithm has been designed to find the most likely path. The algorithm is given in Algorithm 4.

Algorithm 4 Path prediction without destination

Input: start, time_to_predict
1: current ← start
2: track.add(current)
3: **while** time < time_to_predict **do**
4: next ← n in current.neighbors with min cost[n]
5: next.time+ = timeToTraverse(current, next)
6: tracks.add(next)
7: current ← next
8: **end while**
9: smooth(tracks) **return** track

This algorithm is essentially a Markovian process, conducted similar to [31]. Without a destination, the next node is determined by examining the neighbors and selecting the one with the highest weight. The timeToTraverse function determines the time to travel between the two nodes, in the same fashion as Section 3.6. Then the smoothing algorithm, detailed in Section 3.5, is applied to make the path appear more natural.

3.5 Path Smoothing

Dijkstra's Algorithm, as well as Algorithm 4, can produce unnatural looking paths in a staircase shape. Such a prediction is shown in Figure 3.3. Jagged paths like these should be smoothed, as maritime vessels would not be making such sharp turns in reality. One might think that this would produce sub-optimal paths. This could be the case, but often jagged paths have very similar costs, so it is best to have a slightly less optimal path if it better aligns with the obvious fact that ships prefer to travel in straight lines.

To smooth the path, it is post-processed with a smoothing algorithm [6]. This

Figure 3.3: Unsmoothed path prediction

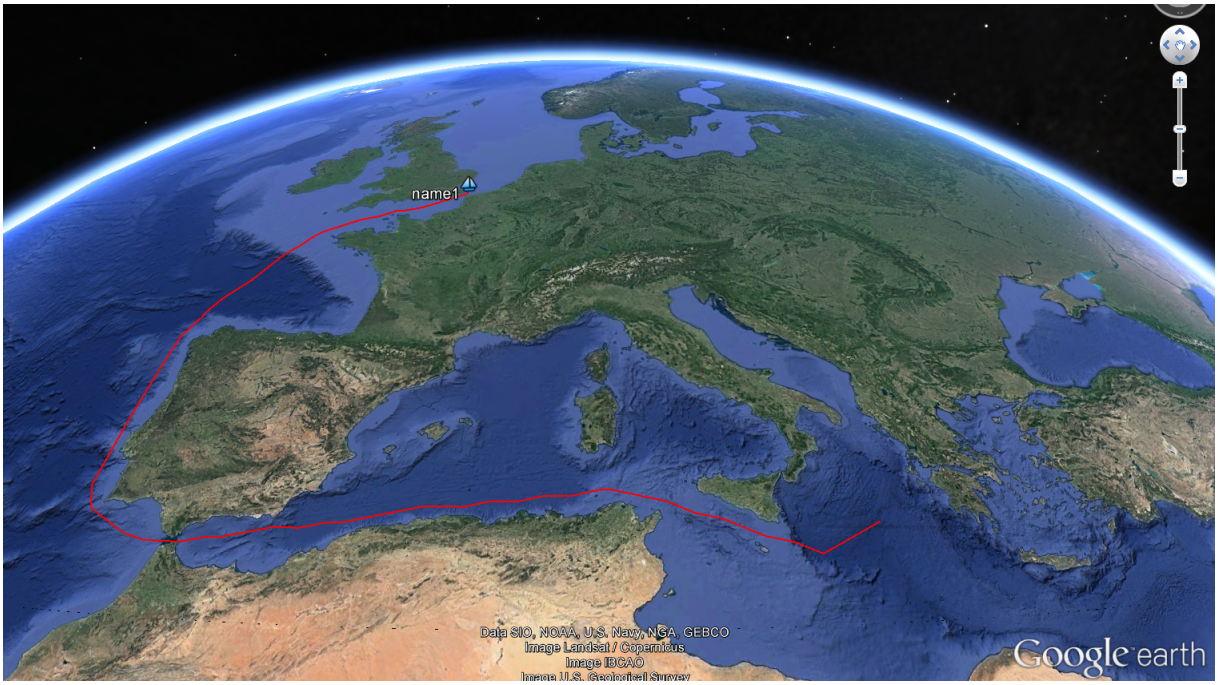

Figure 3.4: Path prediction after smoothing

algorithm is detailed in Algorithm 5. In essence, the smoothing algorithm iterates through each point along the path, and deletes any points that are unnecessary. It determines this by checking if the differences between weights of the points between the the current and next point are within a threshold, which will be referred to as the Smoothing Threshold T_{sm}. If the points fall within the threshold, the endpoint is considered unnecessary and will be deleted from the path. These checks will be accomplished using the *walkable* function given in Algorithm 6. The *walkable* function checks all points between the current and next point to see if their costs are similar enough to be considered walkable. Linear interpolation is used to determine which points fall between the current and next point.

Figure 3.4 shows the result of the smoothing algorithm after having been run on the path in Figure 3.3 with a smoothing threshold of 0.2. As can be seen, most of the perturbations have been eliminated from the path. The choice of a smoothing threshold is a balance between smoothing the path and preserving meaningful turns. If the smoothing threshold is too low, there will be too many unnatural perturbations in the path. If the smoothing threshold is too high, the path will simply be a series of straight lines, avoiding only land. 0.2 was found to be a good compromise for the test dataset.

Algorithm 5 Smoothing algorithm

```
1: current ← path.begin
2: while Point.next ≠ NULL do
3:     if walkable(current, current.next) then
4:         delete current.next
5:     else
6:         current ← current.next
7:     end if
8: end while
```

Algorithm 6 walkable Function used by the Smoothing Algorithm

course $\leftarrow \tan^{-1}$ slope
for distance $= 0 :$ distanceBetween(P_1, P_2) **do**
    $x \leftarrow P_1.x +$ distance $\times \cos$ course
    $y \leftarrow P_1.y +$ distance $\times \sin$ course
    if $|w_{x,y} - w_{P1}| \geq T_{sm}$ **OR** inLand(x, y) **then return** false
    end if
end for
return true

3.6 Speed Prediction

In order to evaluate arrival times from paths generated in Section 3.4, the structure generated in Section 3.3 is used. Essentially, the vessel's movement is simulated through the path prediction generated in Section 3.5 to generate the predicted arrival time. First, waypoints along the vessel's path are generated using interpolation, making sure that the waypoints do not skip cells. This way more speed information is generated.

To determine the arrival time, it then becomes a simple matter of the elementary formula $time = speed \times distance$, for each pair of waypoints, i and j, then summing them. The only wrinkle is finding the distance, since we are dealing with longitude, λ, and latitude, φ, rather than Cartesian coordinates. Locations i and j will have latitude-longitude coordinates (φ_i, λ_i) and (φ_j, λ_j), respecitvely. In geospatial analysis [7], the Cosine formula may be used:

$$d_{ij} = R \cdot \arccos[\sin\varphi_i \cdot q \sin\varphi_j + q\cos\varphi_i \cdot q\cos\varphi_j \cdot q\cos(\lambda_i - \lambda_j)] \tag{3.11}$$

where R is the radius of the Earth. However, in practice, the Haversine formula is

more commonly used to reduce computational errors:

$$d_{ij} = 2R \cdot \arcsin(\sqrt{\sin^2(A) + \sin^2(B) \cdot \cos\varphi_i \cdot \cos\varphi_j}) \tag{3.12}$$

where:

$$A = \frac{\varphi_i - \varphi_j}{2}, B = \frac{\lambda_i - \lambda_j}{2} \tag{3.13}$$

Unfortunately, the stored mean speeds turn out to have a wide variance with an unclear distribution type, as seen in Figure 3.2. This means potentially inaccurate predictions of the vessels' speeds will be generated. To partially compensate for this, the initial speed of the vessel will be used for earlier points in time. Then, the longer the prediction is carried out, the more the predicted speed will tend towards the mean speed. At position i and time t, the speed is calculated as:

$$v_i(t) = \begin{cases} v_{\text{initial}} & t \leq t_1 \\ v_{\text{initial}} + \frac{(v_{i,\text{mean}} - v_{\text{initial}})(t - t_1}{t_2 - t_1} & t_1 < t \leq t_2 \\ v_{\text{mean},i} & t > t_2 \end{cases} \tag{3.14}$$

where t_1 is the time to use the initial speed v_{initial} until, and t_2 is the time after which to use the previously calculated mean speed v_{mean}. When $t_1 < t \leq t_2$, linear interpolation is used to find a value between v_{initial} and v_{mean}. This is illustrated in Figure 3.5.

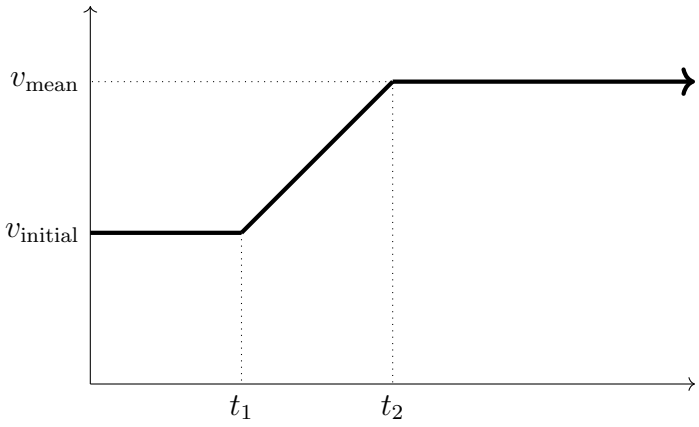

Figure 3.5: Speed transition

3.7 Discussion

Because the runtime when processing the AIS data is relatively long, it was stored in a separate PostgreSQL database, so it could be loaded when predictions needed to be performed. This allows for more refined predictions based on different factors. The database could contain multiple entries depending on desired factors to be controlled for, if such data was available. Weather, time of year, and vessel type, for example, may be used as different factors. The provided dataset was relatively small, so only the vessel type was tested for.

3.7.1 Computational Complexity

In terms of computational complexity, several factors are at play: the amount of AIS messages to be processed, n_{ais}; the number of grid nodes, N; and the number of edges, E.

For building the graph in Sections 3.1 - 3.3, n_{ais} will be the most important variable that affects runtime. For one of the tested regions, over a three month period 476,541

AIS messages had to be processed. However, each message will only need to be queried once, so the computation time will be only $O(n_{\text{ais}})$. Additionally, the AIS messages were queried from a PostgreSQL database. Due to the size of the database, the query can take a significant amount of time. But speeding up this query time is not the focus of this algorithm. Another computationally expensive aspect is the shapefile processing in Section 3.2.3. Even using low resolution shapefiles, there will be hundreds of polygons to process, and each location will have to be compared against each polygon. Fortunately, this process is considered a pre-processing, offline stage. In a real system, it would not be needed for real-time predictions, so the computational cost is not a core concern compared with the online portion.

For prediction in Section 3.4, N and E will be important. As was mentioned in Section 3.4.1, Dijkstra's algorithm has a runtime of $O((N + E) \log N)$ in the worst case with an efficient implementation. In practice, the closer the destination is to the starting point, the faster the algorithm will perform. The smoothing algorithm in Section 3.5 will depend on the number of points along the predicted path. This will be significantly smaller than the number of edges accessed, so it will be overall negligible. Similarly, the arrival time prediction in Section 3.6 will only depend on the size of the smoothed path. The average speed in each cell has been pre-computed, so the computational cost of finding arrival time grows linear with the size of the path.

Chapter 4

Results

4.1 Procedure

The first step in the procedure was to build a density map as outlined in Section 3.2. This was performed using C++ from a historical dataset that is detailed below. Once the graph construction was completed, it was exported to a PostgreSQL database for use in the prediction phase.

A dataset consisting of AIS messages over a three month period was used. This dataset was provided courtesy of help from the exactEarth company. The dataset was stored in a PostgreSQL database. The database is queried to return all AIS messages within the area of interest. The messages are then sorted by MMSI and timestamp to allow for the grid construction procedure. A region off the coast of Mexico was selected. Most of the destinations given in the AIS messages are vague, completely incorrect, or simply not present, and the amount of data in the database was limited in size. Therefore, for prediction with known destination, the destination was taken as the next zero-speed point in the track. The density map for the messages in this

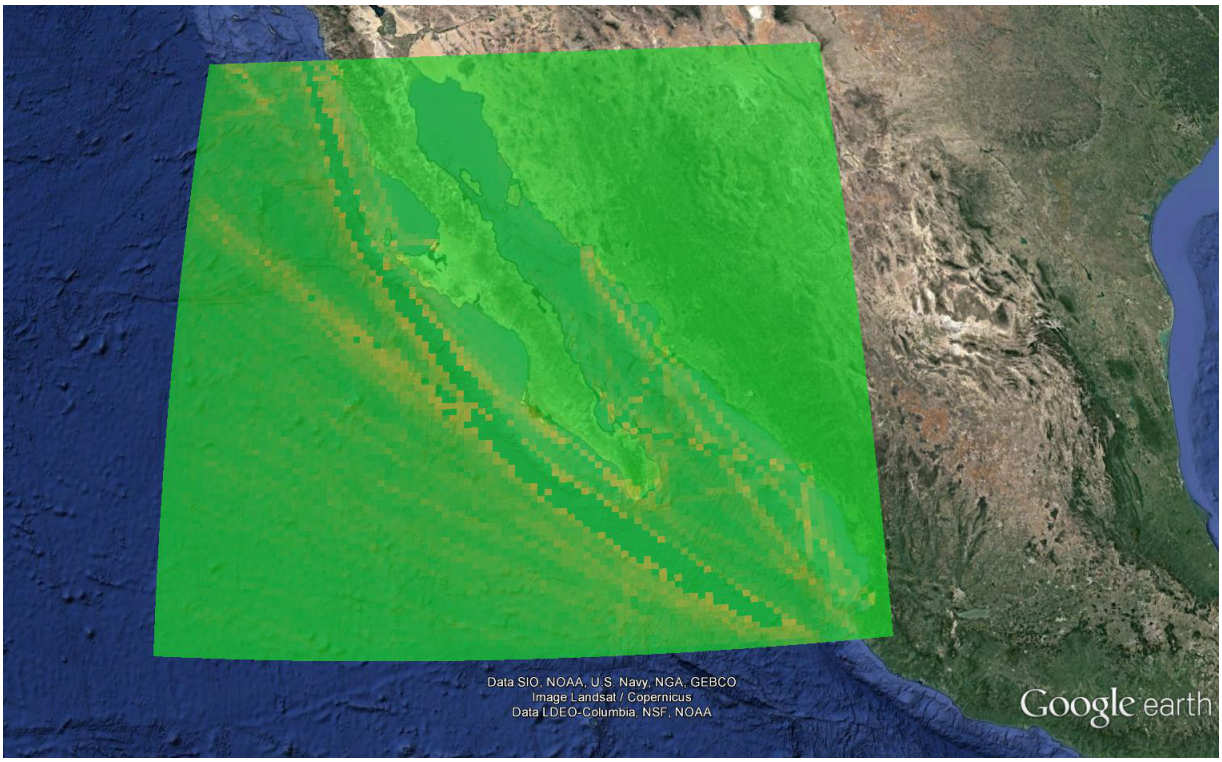

Figure 4.1: Density map of the coast of Mexico region

region is shown in Figure 4.1.

In this region, the most common vessel type in the dataset is type 70, which corresponds to general cargo ships [2]. Figure 4.2 shows the density map for ships of just class 70. Clearly, the traffic patterns are very different if the class is filtered out. In this case, the patterns are more distinct. Therefore, it makes sense to separately examine the effect when only one vessel type is considered. To do this, when the graph structure is being built, only messages from vessels of type 70 were used, and tested against vessels of type 70.

The relevant model parameters are given in Table 4.1.

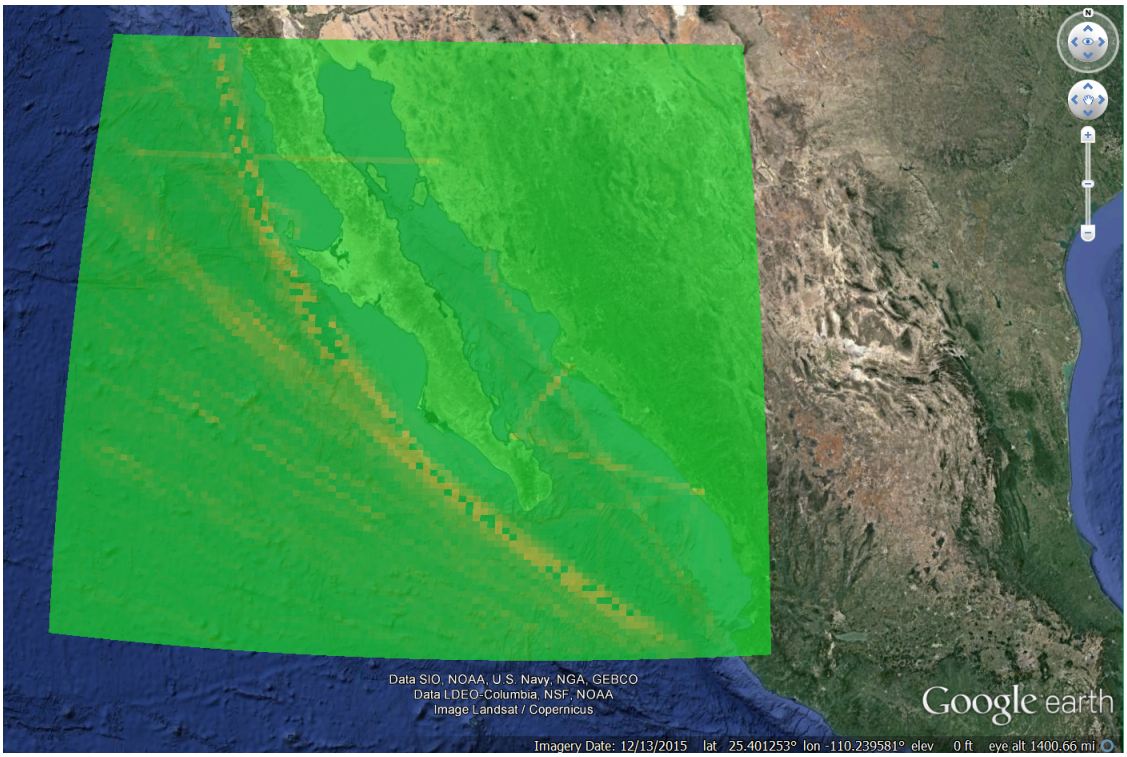

Figure 4.2: Density map of vessels of class 70

Parameter	Value
Longitude Range	$[-120°, -105°]$
Latitude Range	$[20°, 32°]$
Diagonal Cell Width	10 nautical miles
Smoothing Threshold	0.2
Initial Velocity Time, t_1	2 hours
Mean Velocity Time, $t2$	3 hours

Table 4.1: Experiment Parameters

4.1.1 Error Evaluation

To evaluate the prediction, the distance between the predicted and actual position was found at every time step. The frequency of the predicted track can be controlled. The prediction model assumes linear paths, so given predicted positions p_1 and p_2, at times t_1 and t_2, it is easy to find p_x, where $t_1 \leq t_x < t_2$. The difficulty comes with the actual data. Some of the tracks have updates of an hour or more apart. A similar linear interpolation scheme could be performed, but the data is infrequent enough that it could produce misleading results. For example, the Figure 4.3 shows a vessel passing straight through land. This is almost certainly not the case. In reality, it likely maneuvered around the land between updates.

4.2 Results

Figure 4.3 shows a prediction of a single MMSI, along with the corresponding truth. This is a long-term prediction over 80 hours. In this long-term prediction, the vessel successfully maneuvering around the landmass is clearly demonstrated. By inspection it can be seen that the path is reasonably close to the true path. However, when the corresponding error over time is plotted in Figure 4.4, it can be seen that the distance error grows significantly over time. This is due mostly to the error in the speed prediction portion of the prediction. There is no better alternative to deal with this error without more information about the scenario. There are too many factors affecting a vessel's speed to compute an accurate offline prediction over the long-term, such as weather and avoiding other vessels. This is also a good example to illustrate the peculiarities in many of the AIS messages. Note that, although the AIS

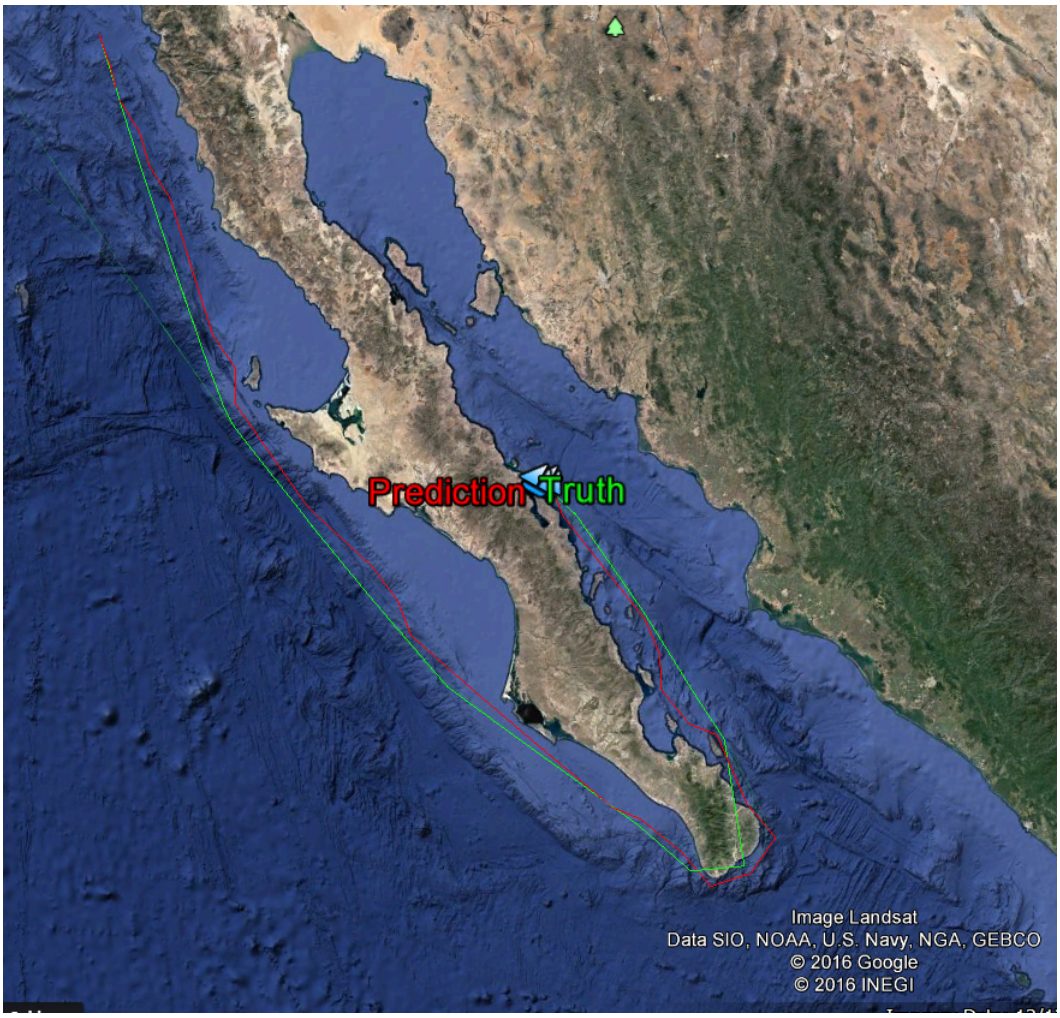

Figure 4.3: Predicted and actual ship path

message appears to cross over land, this is not actually the case. The update rate of AIS messages is variable, and Google Earth attempts to fill in the gaps. Most likely, the vessel sent out an update, traveled around the landmass, then sent out another update. Google Earth displays this as a straight line through the landmass.

Next, the average error for 100 predictions was found. 100 tracks were randomly selected from the database, and predictions were conducted both for known and unknown destinations. The average error in both cases is shown in Figure 4.5. They

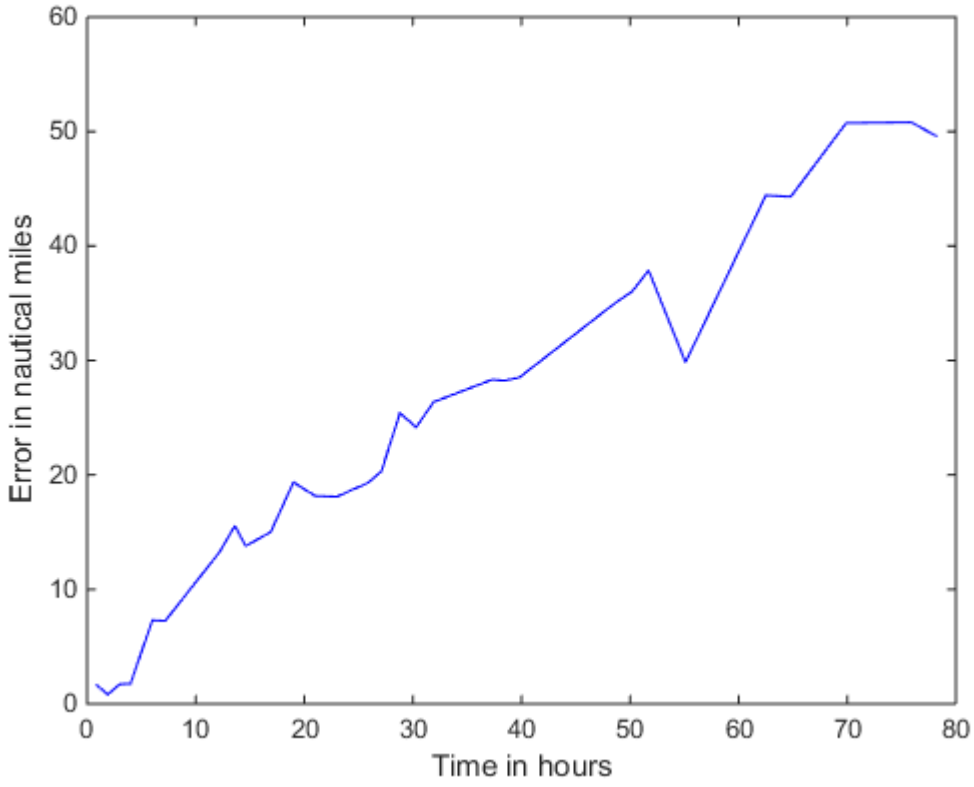

Figure 4.4: Over time, distance between actual and predicted path

are very close early in the run, but the unknown destination error increases for longer journeys. The error for the unknown destination case is much higher, particularly as time increases. This should be expected, because the destination is a valuable piece of extra information. Near the beginning of a journey, the vessel is likely to travel in a straight line, and may maneuver further into the journey. It is difficult to predict such maneuvers without knowing the vessel's endpoint.

Also notice the spikes in the graphs. This is largely due to erroneous tracks. Some tracks were noted to do very strange things, such as moving back and forth, seemingly at random. Others displayed high spikes in speed, beyond what is reasonable. Still others would jump around, clearly in error. So this leads to average errors higher than should be reasonably expected. This can be seen in the histogram for t = 22 hours, shown in Figure 4.6. In Figure 4.5, there is a spike at $t = 22h$. However, in the histogram, it can be seen that there is an error of 115 skewing the mean. When the offending MMSI was examined, the vessel was apparently teleporting across large distances. Tracks such as this are clearly in error and should not be used for comparison purposes. When the most inaccurate 10% of tracks are excluded from consideration, the results in Figure 4.7 are generated. As can be seen, the average error drops significantly.

Figure 4.8 shows the average error when only vessels of class 70 are considered, both for the training and prediction phase. As can be seen, a small improvement is demonstrated when class information is included.

Moreover, the Florida region was also examined to see if different traffic patterns had an effect on the results. In Figure 4.9, different traffic patterns are apparent. The same grid size was used. When experiments were run, the error results were

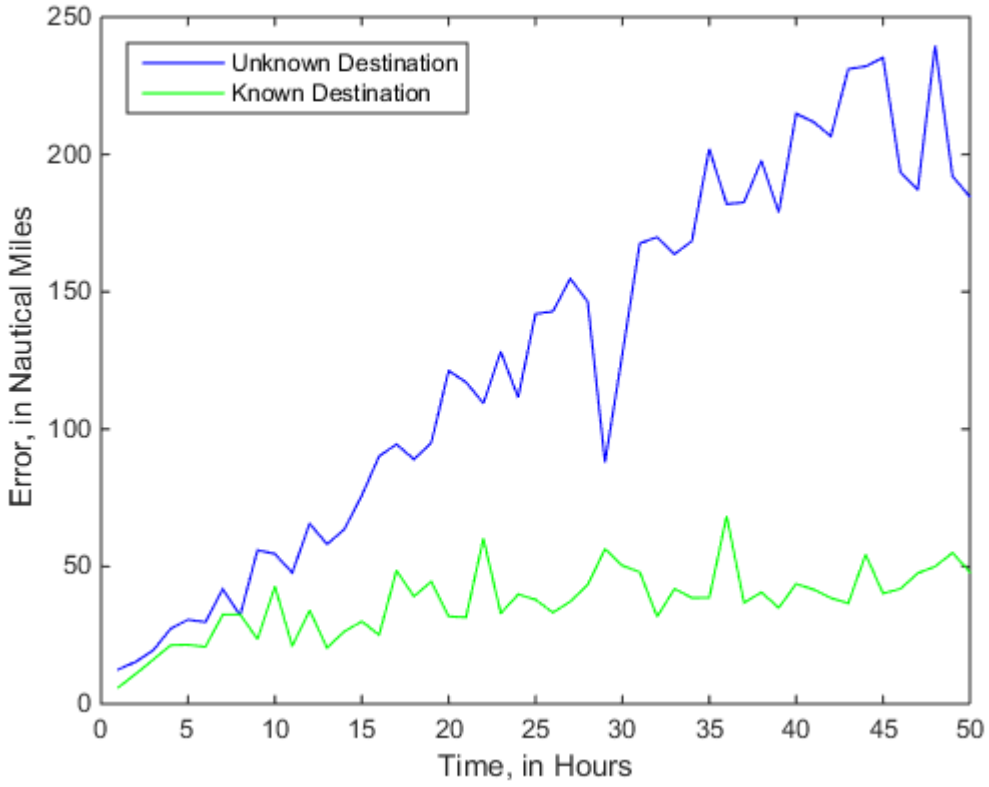

Figure 4.5: The averaged error for 100 runs

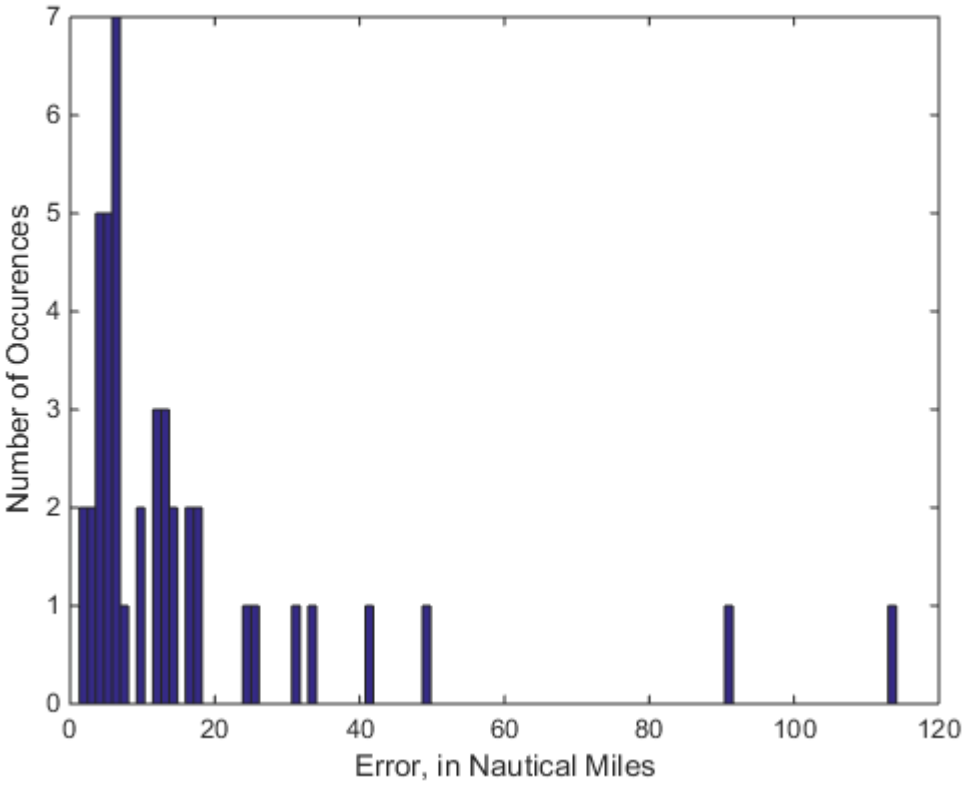

Figure 4.6: Error histogram for t = 22 hours

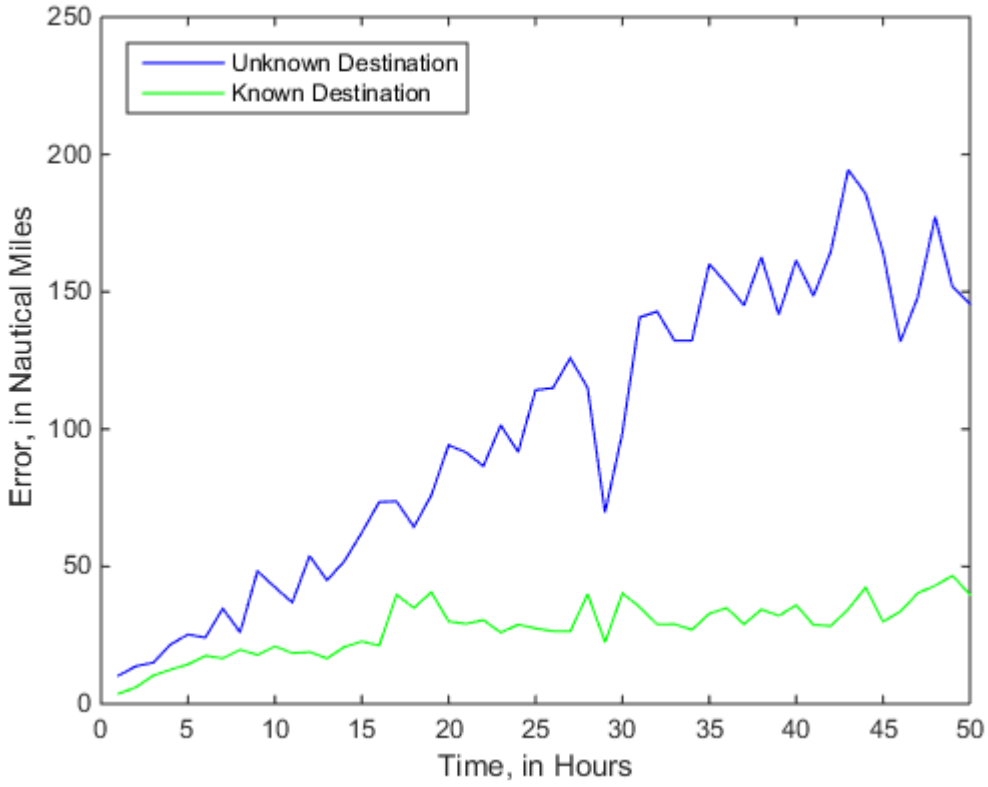

Figure 4.7: Average error for 100 runs, with outliers filtered out

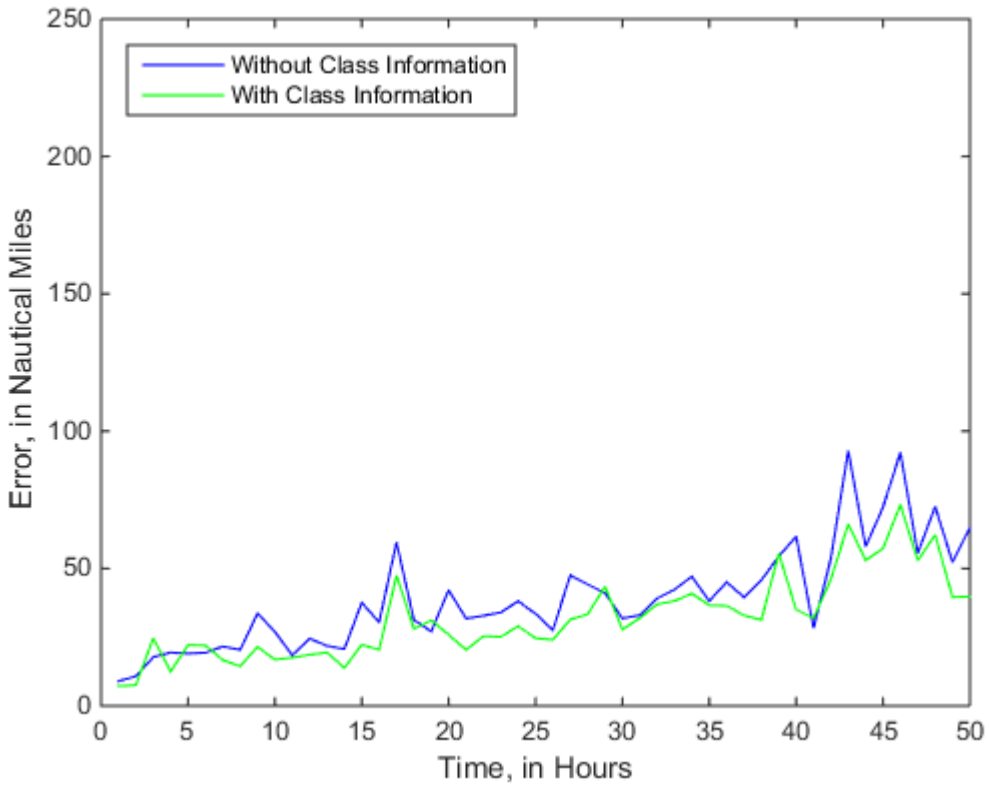

Figure 4.8: Comparison of error when considering vessel class

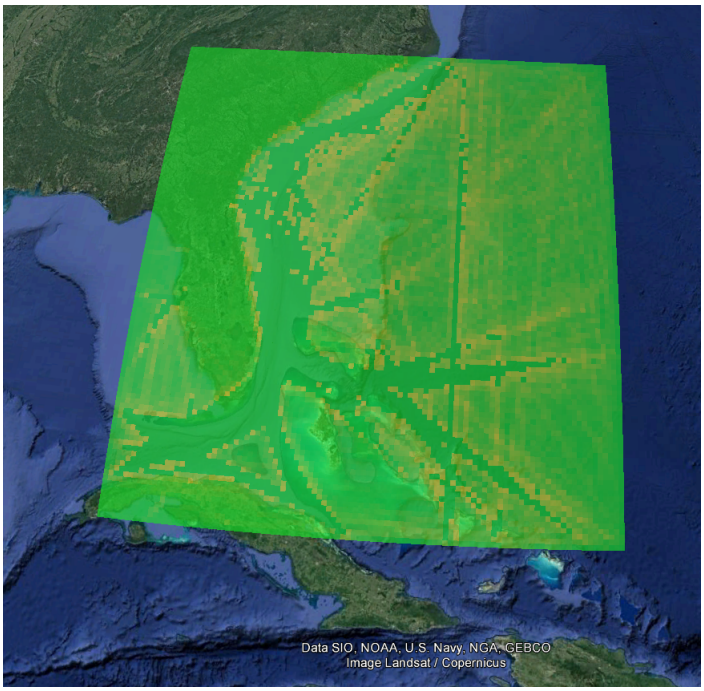

Figure 4.9: Density map of Florida region

demonstrated in Figure 4.10. It can be seen that the error is higher in this region. Different regions may have different traffic rules [23], so it makes sense that different error rates would be observed.

4.3 Discussion

Evidently, the prediction with a known destination performs better over the long term. It is interesting to note that the unknown destination prediction performs on par with the known destination prediction in the short term. In fact, it appears to perform slightly better in some cases. This is likely because it is more sensitive to the initial course. So it might give better results at first, but as the vessel performs maneuvers, according to shipping lanes, a prediction with a known destination will

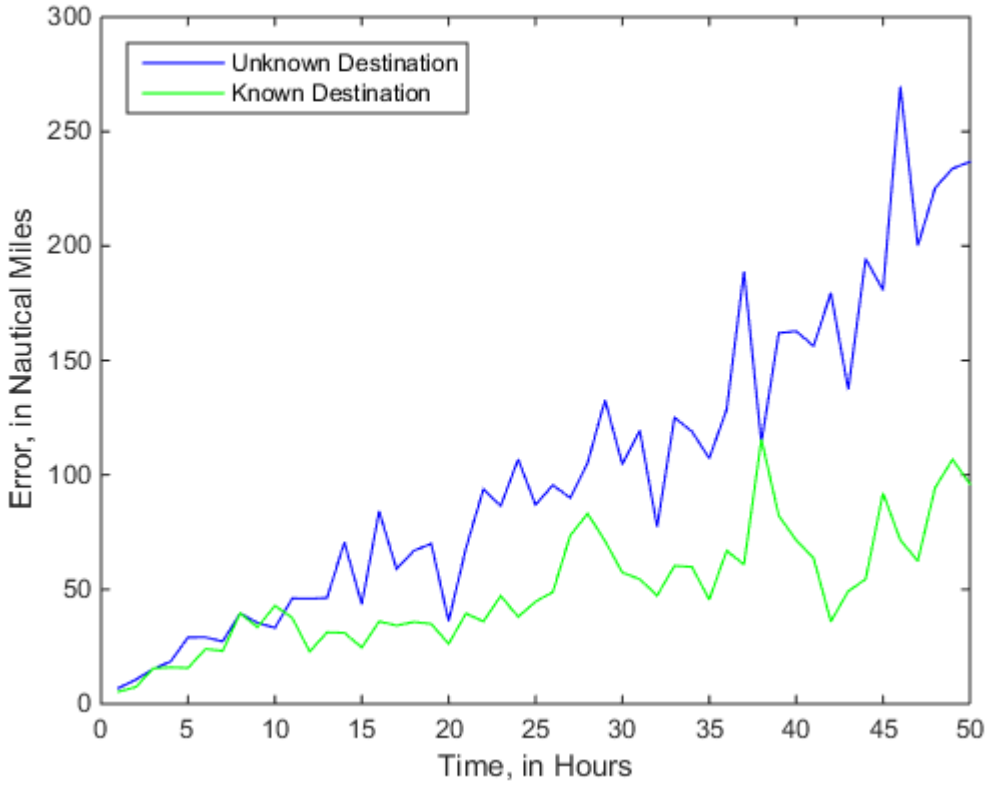

Figure 4.10: Average error in the Florida region, with outliers filtered out

produce better results.

The error ends ups being quite large, at 50 nautical miles after 50 hours. There are two main sources of error: the error in the path prediction and the error in the speed prediction. It is difficult to quantify the two separately. For the path prediction, the main drawback of the grid-based method is that the predicted path will travel along the grid cells. In some cases, it was observed that the prediction traveled parallel to the actual path, but introduced a large error because it was at the edge of the cell. For this experiment, with grid cell sizes of 10 nautical miles, this could mean an error of 5 nautical miles right, even if the predicted cell was correct.

The second source of error is the speed prediction. The main cause of this is the variability in the average speed. As discussed in Section 3.3, there is a high amount of variability in the average speed of each cell. Assuming a near-constant velocity model also creates a high amount of variability [22]. So there will be a limit on how accurate the speed prediction can be.

A significant drawback for this thesis was the lack of data available to construct the graph. Ideally, more data would be available with which to analyze. With more data, it should be possible to further refine predictions based on information such as time of year. As is, with a relatively small dataset, it was difficult to get enough data to meaningfully refine and test the impact of such factors.

Chapter 5

Conclusions

This thesis has presented a novel approach to predicting maritime ship paths using a large amount of historical AIS data. This was accomplished by discretizing a region of interest into a grid, then determining the density in each grid cell offline. The novelty of this work comes in the online prediction phase through the use of shortest-path planning algorithms, such as Dijkstra's Algorithm to determine the optimal path to use for the prediction. This method is much more simple and computationally cheap than other methods, such as [23]. Shapefile data was used to ensure landmass avoidance, and a good distance was maintained from such landmasses. Using vessel class information was shown to somewhat improve prediction accuracy. The two sources of error were the path prediction and speed prediction. The path prediction error was tied to the resolution of the grid. The speed prediction error was tied to the wide uncertainty in the historical data.

The potential for this method was demonstrated by predicting the path and arrival time of vessels with reasonable accuracy. However, it should be noted that it is difficult to find a benchmark to evaluate the accuracy of these predictions. This is not a

conventional estimator, so it is difficult to determine a theoretical error bound, such as a Cramer Rao Lower Bound. Additionally, other papers do not present average error for their predictions or are focused on other aspects such as anomaly detection [23] [24] [28]. Long-term prediction based on AIS is not a well studied problem, so these results could serve as a point of comparison for future work.

5.1 Future Work

Recently, a method was published detailing long-term prediction of vessel speeds using mean-reverting processes in [22]. They found that maritime vessels tend to travel around a particular mean speed. This could be one avenue to improve the speed prediction used in this thesis. For previously observed vessels, the speed prediction could be adjusted based on their previously determined mean speeds.

It would also be beneficial if the grid size was not fixed. There are sparse amounts of data in some regions, and dense amounts of data in other regions. The issue is that, with grid cells too small, sparse data leads to cells with little or no density data. Meanwhile, if grid cells are too large, some of the path information may be lost as different trajectories are grouped together in the same cell. Ideally, large grid cells would be used for sparse regions and fine cells would be used in the dense regions.

Due to the amount of data involved, computation time becomes an issue, particularly as the problem is scaled to larger regions or a grid with finer resolution. So it would be beneficial if this problem could be tackled from a big data and cloud computing perspective. It could also be useful to determine alternate paths, using k-shortest path algorithms such as Yen's Algorithm [35].

Appendix A

In Polygon Test

The C++ function, based on [11], to determine if a point lies within a polygon is given below. This function was used in Section 3.2.3 to determine if a point lies within a landmass.

```
int inLandMass(std::vector<point>& landmass, double longitude, double latitude)
{
        int i, j, c = 0;
        for (i = 0, j = landmass.size() - 1; i < landmass.size(); j = i++) {
                if (((landmass[i].latitude>latitude) != (landmass[j].latitude>latitude)) &&
                        (longitude < (landmass[j].longitude - landmass[i].longitude) *
                        (latitude - landmass[i].latitude) /
                        landmass[j].latitude - landmass[i].latitude)
                        + landmass[i].longitude))
                {
                        c = !c;
                }
        }
        return c;
}
```

Bibliography

[1] ESRI shapefile technical description. Technical report, Environmental Systems Research Institute, July 1998. URL: `https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf` (accessed on Sept 8, 2016).

[2] Recommendation m.1371-5: Technical characteristics for an automatic identification system using time-division multiple access in the VHF maritime mobile band. Technical report, International Telecommunication Union (ITU), February 2014. URL: `https://www.itu.int/rec/R-REC-M.1371/en` (accessed on Jan 29, 2017).

[3] N. A. Bomberger, B. J. Rhodes, M. Seibert, and A. M. Waxman. Associative learning of vessel motion patterns for maritime situation awareness. In *Proceedings of the 9th International Conference on Information Fusion*, pages 1–8, July 2006.

[4] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*, chapter 24. McGraw-Hill Higher Education, 2nd edition, 2001.

[5] X. Cui and H. Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[6] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[7] M. J. de Smith, M. F Goodchild, and P. Longley. *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*. Troubador Publishing Ltd, 2007.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[9] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[10] E. Fernandes, P. Costa, J. Lima, and G. Veiga. Towards an orientation enhanced astar algorithm for robotic navigation. In *Proceedings of 2015 IEEE International Conference on Industrial Technology*, pages 3320–3325, March 2015.

[11] W. R. Franklin. PNPOLY - point inclusion in polygon test. URL: `https://www.ecse.rpi.edu/~wrf/Research/Short_Notes/pnpoly.html` (accessed on Sept 12, 2016).

[12] R. P. Grimaldi. *Discrete and Combinatorial Mathematics*. Pearson Education, 5 edition, 2006.

[13] E. Haines. Point in polygon strategies. In *Graphics Gems IV*, pages 24–46. Academic Press, 1994.

[14] A. Harati-Mokhtari, A. Wall, P. Brooks, and J. Wang. Automatic Identification System (AIS): Data Reliability and Human Error Implications. *Journal of Navigation*, 60:373, August 2007.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[16] International Maritime Organization (IMO). Chapter V: safety of navigation, regulation 19. *International Convention for the Safety of Life At Sea*, December 2002.

[17] J. R. Jiang, H. W. Huang, J. H. Liao, and S. Y. Chen. Extending Dijkstra's shortest path algorithm for software defined networking. In *Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium*, pages 1–4. IEEE, 2014.

[18] Y. Kambayashi, H. Yamachi, Y. Tsujimura, and H. Yamamoto. Dijkstra beats genetic algorithm: Integrating uncomfortable intersection-turns to subjectively optimal route selection. In *Proceedings of the 2009 IEEE International Conference on Computational Cybernetics*, pages 45–50, Jan 2009.

[19] D. Klein and C. D. Manning. A* parsing: fast exact Viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, volume 1, pages 40–47. Association for Computational Linguistics, 2003.

[20] D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[21] M. Luby and P. Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(1):551–567, 1989.

[22] L. M. Millefiori, P. Braca, K. Bryan, and P. Willett. Long-term vessel kinematics prediction exploiting mean-reverting processes. In *Proceedings of the 19th International Conference on Information Fusion*, pages 232–239, July 2016.

[23] G. Pallotta, M. Vespe, and K. Bryan. Traffic knowledge discovery from AIS data. In *Proceedings of the 16th International Conference on Information Fusion*, pages 1996–2003, July 2013.

[24] B. Ristic, B. la Scala, M. Morelande, and N. Gordon. Statistical analysis of motion patterns in AIS data: Anomaly detection and motion prediction. In *Proceedings of the 11th International Conference on Information Fusion*, pages 1–7, June 2008.

[25] A. N. Skauen. Quantifying the tracking capability of space-based AIS systems. *Advances in Space Research*, 57(2):527 – 542, 2016.

[26] K. Uchida. Optical ray tracing based on Dijkstra algorithm in inhomogeneous medium. In *Proceedings of the 9th International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 371–376, Nov 2014.

[27] UNCTAD. Review of maritime transport 2015. URL: `http://unctad.org/en/pages/PublicationWebflyer.aspx?publicationid=1374` (accessed on Jan 29, 2017).

[28] M. Vespe, I. Visentini, K. Bryan, and P. Braca. Unsupervised learning of maritime traffic patterns for anomaly detection. In *Proceedings of the 9th IET Data Fusion Target Tracking Conference*, pages 1–5, May 2012.

[29] P. Wessel and W. H. F. Smith. A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research*, 101(B4):8741–8743, April 1996.

[30] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[31] J. Xu, T. L. Wickramarathne, and N. V. Chawla. Representing higher-order dependencies in networks. *Science Advances*, 2(5), 2016.

[32] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *Proceedings of the 2013 IEEE International Conference on Data Engineering*, ICDE '13, pages 254–265, Washington, DC, USA, 2013. IEEE Computer Society.

[33] S. Yang and C. Li. An enhanced routing method with Dijkstra algorithm and AHP analysis in GIS-based emergency plan. In *Proceedings of the 18th International Conference on Geoinformatics*, pages 1–6, June 2010.

[34] R. D. Yates and D. J. Goodman. *Probability and stochastic processes: a friendly introduction for electrical and computer engineers*. John Wiley & Sons, 2005.

[35] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[36] W. Zeng and R. L. Church. Finding shortest paths on real road networks: The case for A*. *International Journal of Geographical Information Science*, 23(4):531–543, April 2009.

[37] F. Zhu. Mining ship spatial trajectory patterns from AIS database for maritime surveillance. In *Proceedings of the 2nd IEEE International Conference on Emergency Management and Management Sciences*, pages 772–775, Aug 2011.

[38] X. Zhu, W. Luo, and T. Zhu. An improved genetic algorithm for dynamic shortest path problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2093–2100, July 2014.