NTNU – Trondheim
Norwegian University of
Science and Technology

Motion Prediction by Optimal Paths Through Disordered Landscapes

Mads Fromreide

Physics
Submission date: May 2014
Supervisor: Alex Hansen, IFY
Co-supervisor: Peter Ellevseth, Safetec Nordic

Norwegian University of Science and Technology
Department of Physics

Motion Prediction by Optimal Paths Through Disordered Landscapes

Mads Fromreide

August 2013 - May 2014

Acknowledgments

This Master's thesis was performed at the Department of Physics at NTNU, in the period August 2013 - May 2014.

I would like to thank my supervisor Professor Alex Hansen, for ideas and guidance through both the practical and written work with this thesis, and for introducing me to the algorithm which the work is based upon.

Trondheim, May 2014

Mads Fromreide

Abstract

The ability to navigate safely and efficiently through a given landscape is relevant for any intelligent moving object. Examples range from robotic science and traffic analysis to the behavior within an ecosystem. Through this thesis, methods for finding traffic patterns and predicting future motion, have been constructed based on theory of optimal paths. The algorithms are applied to maritime traffic, in terms of recorded vessel coordinates.
By considering the structure of a given traffic situation as a disordered energy landscape, one can define optimal routes within the area. An algorithm for finding hierarchies of optimal paths in a disordered energy landscape is implemented. The algorithms are used in two settings, one for detecting patterns of motion within a given area, and a method for estimating single vessel trajectories. The results found in the thesis, show that the methods have great potential for analyzing traffic patterns and predict future motion.

Abbreviations

- AIS: Automatic Identification System
- EM: Expectation-Maximization
- IMO: International Maritime Organization
- MLE: Maximum-Likelihood Estimate
- SOLAS: Safety Of Life At Sea

Contents

1. Background

1.1 Motivation

The understanding of the regular behavior of a dynamic system, gives opportunities to predict future events within it. Objects tend to move in patterns, and are often influenced by other surrounding objects. By applying theory of optimal paths, moving through a disordered energy landscape, one may gain knowledge about the regular movement within an area. Applications range from behavior of ecosystems, to robotic navigation and large traffic systems [15].

Maritime safety and security are important concerns in todays society. Maritime transportation is the most important way of transport, measured by volume [11]. Gaining knowledge about the regular patterns of vessel traffic gives opportunities to explain the future behavior of the system. Being able to predict future motion gives the opportunity to predict future events, for instance vessel collisions. Accidents involving large vessels may risk the health and life of people, as well as having big environmental impacts and economical costs. Other concerns that may be enlightened by the understanding of the typical behavior of vessel motion, are illegal import/export, maritime pollution, piracy and maritime terrorism [12].

Through the *Safety Of Life At Sea* (SOLAS) convention, the *International Maritime Organization* (IMO) has introduced a reporting system known as the *Automatic Identification System* (AIS). The SOLAS convention originated in 1914 and was initiated by the United Kingdom government after the Titanic accident in 1912 [10]. Through the years, the convention has been edited and new information has been added. The AIS system was introduced to establish the position of vessels lying in radar shadows [1], helping nearby vessels to avoid accidents. By using satellite navigation along with AIS signals, vessels may locate other nearby vessels more accurately. The AIS system is based on transponders on the ships and land-based stations. By the SOLAS convention, all domestic ships with a gross-tone weight above 500, international ships with weight over 300 tones and all passenger ships are required to carry AIS transponders [1, 9]. These exact requirements became efficient for all ships by the end of 2004 [9]. The AIS messages contain different types of information. The information is classified into two

groups, dynamic and static information. The static information includes a unique identification number, name, type and size of the vessel. Position (in geodesic coordinates), speed, course, heading, port of destination and estimated arrival time belong in the dynamic information class [12]. The different types of information are broadcasted in their own specified time intervals and not all types of information are compulsory. The AIS system provides ships with a large amount of near real time information, which enables the use of automatic control systems [11]. In addition, storing AIS data gives easy access to large detailed datasets which may be used for analysis.

In this thesis, an algorithm for finding optimal paths in a disordered energy landscape is applied to analyze the motion of moving objects. By different implementations of the algorithm one may find regular patterns of motion and predict the future behavior of vessels. The theory used, is applicable to a general setting, and are tested on real AIS datasets.

1.2 Structure of thesis

1. **Background**
   Motivation behind the thesis

2. **Introduction**
   An introduction to the theory for which this thesis is built upon. Includes pre-existing methods.

3. **Method**
   Description of the different algorithms used in the thesis.

4. **Results**
   Presentation of the simulations and calculations performed.

5. **Discussion**
   Discussion of the results.

6. **Conclusion**
   Conclusion and ideas for future work.

7. **Appendix**
   Draft for an article based on the work in the thesis.
   The programs created for the analysis. All code is written in C, except for the plot scripts created for gnuplot.

2. Introduction

2.1 Motion prediction in a dynamic environment

When navigating through a dynamic environment, a moving object must adjust to the changes of its surroundings. The change of the environment may be caused by other moving objects populating the same area. In order to adjust to changes, a moving object must in some way be able to predict the future motion and behavior of its surroundings. For most animals and intelligent beings the ability to efficiently navigate and interact with its surroundings is crucial for surviving [15]. For instance, a hunting predator must predict the future motion of its prey in order to catch it. If the area is populated by other predators, both equal or above itself in the food chain, the predator must take into a count these motions as well.

Motion prediction may be applied to a wide range of fields from robot navigation and video surveillance to collision avoidance and behavior of an ecosystem [15]. Motion prediction may be performed on both microscopic and macroscopic level. That is, for a microscopic view of the system one predicts the future motion of every single object in the area of interest. This is based on the future behavior of the system upon a prior motion model, for instance differential equations. One may group the different models of motion into three groups, or as combinations of these: (i) Constant velocity model, (ii) Random motion model and (iii) Intentional motion model [17].

The constant velocity model (i) assumes that all objects move with a constant velocity, that is with zero acceleration. This simple model may seem modest and unrealistic in many cases, however it may give a good description when applied to small time scales. That is, for a small time interval Δt it is fair to approximate the motion with constant velocity. Model (ii) shows the acceleration as a stochastic function, hence the velocity and position vectors will be stochastic functions as well. By assigning different probability distributions to the stochastic variables one may create a realistic environment. In the Intentional model (iii), the objects move in a scheduled way. Different objects may move in different ways according to their needs and goals. In this case the acceleration vector will be a function of time. This function may or may not be unique for each object. All

these groups may be further specified, adjusting the environment of interest. For instance in the example of predators hunting preys, the equations will be coupled, taking into consideration the motion of the surrounding animals.

It soon becomes evident that if the environment consists of a large number of moving objects, these kinds of calculations will be too time demanding. Therefore, in order to handle large areas with complex traffic, another approach has emerged [15]. This approach operates on a macroscopic level. The method is based on the assumption that objects tend to move in patterns. These patterns are determined both by the object's nature and its surrounding environment [15]. Patterns are found everywhere in the nature and is therefore important in many branches of science. The next section gives an introduction to the field and terms of pattern recognition, and highlights the most used methods. Further, an introduction to optimal paths are given, before section 3.1 describes how optimal paths can be used to predict future behavior.

2.2 Pattern recognition

The term pattern recognition means to extract data that has certain similarities, from the rest of its environment [7]. In the research field of pattern recognition the goal is to learn machines the ability to recognize patterns. In *Pattern recognition: human and mechanical* Watanabe defines a pattern as "the opposite of chaos" [16]. By this rather vague definition, many things can be said to be patterns. For instance, letters are formed in certain patterns. The patterns that are formed, are independent of the handwriting, machine writing and cluttered backgrounds. The letters still read the same [7]. Pattern recognition applies to many branches of science, such as biology, psychology, medicine, linguistics, computer vision, artificial intelligence, etc.

When applying theory of patterns for predicting future motion, the process is divided into two stages. First the *learning stage*, which consists of the actual pattern recognition. In this step, the goal is to learn the typical motion patterns within a given area [15]. The next stage is the *prediction stage* which predicts future trajectories of an object using the learned patterns.

2.2.1 Learning stage

When a pattern is recognized or classified, it is either considered to be a part of an already known class, or it is assigned to a new unknown class. These two categories are referred to as supervised and unsupervised classifications, respectively [7]. When dealing with motion patterns one would typically need to learn a new pattern for every situation studied. Pattern recognition can be approached in different ways. The four categories: template matching, statistical classification, syntactic or structural matching and neural networks are considered to be the best known [7].

The idea behind template matching is that there exists a template or original pattern. The actual recognition process is performed by matching the new structure with the template. This process is performed, taking into consideration all possible orientations, by rotating and translating the structure. For instance one could imagine a machine trying to detect a square among many different geometrical figures. It would then compare all the figures with a template square until the best match is found. In syntactic (or structural matching) one considers each pattern to consist of several sub patterns,

which again consists of even smaller and simpler sub patterns. The smallest pieces are called primitives. In order to construct the original pattern from the primitives, one needs a set of rules or a recipe. Hence patterns may be classified by their primitives and their rules. Consider a sentence as a pattern. The sentence may be grouped as individual words (sub patterns), and further as individual letters (primitives). The associated rules would then be the spelling and grammar of the relevant language [7].

When looking at patterns of motion, the statistical approach to pattern recognition is the most applicable method. The statistical approach characterizes a pattern by d measurements/features and represent each unique pattern as a point in d-dimensional space [7]. Statistical decision rules are then used to separate different patterns from each other in a set of training data. Each pattern is then assigned decision boundaries. One example of this process is the process of detecting motion patterns by pairwise clustering. Starting with N paths in a given area, one seeks to minimize the number of paths by introducing representative trajectories which consists of one or several paths, belonging to the same pattern [15]. When using pairwise clustering one seeks out the two paths that are most likely to belong to the same pattern. These two paths are then grouped in a cluster represented by their mean trajectory and the associated standard deviation. The process is now repeated with the $N - 1$ paths (including the newly constructed cluster) until some boundary is met. As new data (paths) are added to the area, one considers the likelihood of the new path belonging to each existing cluster. If a match is found, the path is added to the cluster. If not, a new cluster consisting only of the new path is added to the area [15].

Another approach is the use of the Expectation-Maximization algorithm (EM algorithm) for clustering whole trajectories. The EM approach is considered to be state of art within cluster-based techniques [15]. The EM algorithm is an iterative technique for calculating maximum-likelihood estimates (MLEs) when the observations are considered as incomplete [5]. Incomplete data implies existence of two sample spaces X and Y. The incomplete data is the observation $\boldsymbol{y}$, which are realizations of Y. The variables $\boldsymbol{x}$ are not directly observable, they may only be calculated by mapping from $\boldsymbol{y}$. That is, in general there exists a mapping $\boldsymbol{x} \rightarrow \boldsymbol{y}(\boldsymbol{x})$, and x is only known to be a realization of the subset $X(\boldsymbol{y})$ of X. Further, letting $\boldsymbol{y}$ be incomplete data, leads to the term complete data for $\boldsymbol{x}$ [5]. If $f(\boldsymbol{x} \mid \boldsymbol{\phi})$ is the family

of densities associated with x and $g(\boldsymbol{y} \mid \boldsymbol{\phi})$ is the corresponding family of densities associated with y, the relation between f and g is given by

$$g(\boldsymbol{y} \mid \boldsymbol{\phi}) = \int_{X(\boldsymbol{y})} f(\boldsymbol{x} \mid \boldsymbol{\phi}) dx. \tag{1}$$

In general, the EM algorithm aims at finding a value $\boldsymbol{\phi}$ that maximizes the function $g(\boldsymbol{y} \mid \boldsymbol{\phi})$ for an observed $\boldsymbol{y}$ [5]. As the name implies, the algorithm consists of two steps. An expectation step and a maximization step. Moreover it consists of maximizing the expected likelihood function. Consider a dataset consisting of a collection of N trajectories $d = \{d_1, ..., d_N\}$. Each trajectory consists of a sequence of positions, $d_i = \{\boldsymbol{x_i^1}, ..., \boldsymbol{x_i^T}\}$, where $\boldsymbol{x_i^t}$ is the positions after t steps. Further it is assumed that there exists M unique patterns of motions, i.e clusters of trajectories, and let θ_m denote a pattern, with $1 \leq m \leq M$. A pattern θ_m is represented by a probability distribution $p(x \mid \theta_m^t)$, which gives the probability of being at position x after t steps given that the trajectory belongs to θ_m. The likelihood of a trajectory d_i belonging to a pattern θ_m may then be written

$$p(d_i \mid \theta_m) = \prod_{t=1}^{T} p(x_i^t \mid \theta_m^t). \tag{2}$$

The EM algorithm aims to maximize Eq. (2) subject to θ [4]. In order to find the likelihood of data belonging to a certain pattern θ, a set of correspondence variables are introduced. These variables are denoted c_{im}, where i specifies the trajectory d_i and m specifies the pattern θ_m. The variables are binary, if the trajectory d_i belong to θ_m $c_{im} = 1$, if not $c_{im} = 0$. One trajectory d_i can only belong to one pattern on motion θ_m, hence for any trajectory d_i

$$\sum_{m=1}^{M} c_{im} = 1. \tag{3}$$

The probability density functions $p(x \mid \theta_m^t)$ are typically assumed to be Gaussian [15, 4, 12]. As the Gaussian distribution is part of the exponential family it is convenient to consider the log likelihood rather than the

likelihood itself [4]. Since the logarithm is a monotonic function, a maximization of the log likelihood will be equivalent to maximizing the likelihood itself. Instead of maximizing the expected likelihood, one maximizes the expected log likelihood. That is, one aims at optimizing the function given by $E_c[\ln p(d, c \mid \theta) \mid \theta, d]$, where $p(d, c \mid \theta)$ is the total likelihood function for all trajectories d_i. The EM algorithm generates iteratively a sequence of patterns of increasing log likelihood. The ordered sequence of such patterns are denoted $\theta^{[1]}$, $\theta^{[2]}$, $\theta^{[3]}$,... The elements of this sequence may be found by introducing an additional function, the Q-function, defined as follows [4]

$$Q(\theta' \mid \theta) = E_c[\ln p(d, c \mid \theta') \mid \theta, d]. \tag{4}$$

The Q-function is a function of two different patterns, θ and θ'. The next element of the sequence of ordered patterns may then be found by

$$\theta^{[j+1]} = \underset{\theta'}{\operatorname{argmax}}\, Q(\theta' \mid \theta^{[j]}). \tag{5}$$

Hence, from a chosen initial model the remaining are found by iterating Eq. (5) until it converges. If the Q-function is continuous, the EM algorithm converges at least to a local minimum [4].

2.2.2 Prediction stage

As regular patterns of motion have been established, they may be used to predict motion within the area. By starting of with a partially observed trajectory, one may predict its future behavior. For each of the clusters, which represents regular paths, one calculates the likelihood for the partial trajectory to belong to it. That is, given a cluster C_k and a partially observed trajectory d_p, one calculates the probability that d_p *belongs to* C_k [15]. In order to calculate the likelihood, the dissimilarity between the partial trajectory and the cluster is calculated. If $d_p(t)$ is the position of the partial trajectory at time t, $d_i(t)$ is the position of a longer trajectory at time t and T_p is the duration of the partial trajectory, then the dissimilarity may be expressed as

$$\delta_p(d_p, d_i) = \left(\frac{1}{T_p} \int_{t=0}^{T_p} (d_p(t) - d_i(t))^2 dt \right)^{1/2} \tag{6}$$

When the distance/dissimilarity is known, one may calculate the likelihood for d_p belonging to a cluster C_k [15]. Under the assumption that a cluster C_k is Gaussian distributed with mean μ_k and variance σ_k^2, the likelihood is expressed

$$P(d_p \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k}} \exp\left(-\frac{\delta_p(d_p, \mu_k)^2}{2\sigma_k^2} \right). \tag{7}$$

After the likelihood has been found for each cluster in the area, one may predict the future motion of the partial trajectory. One option is to estimate the future behavior with the mean value of the cluster with maximum likelihood. Another option is to give several possibilities represented by a cluster with the associated probability [15].

2.3 Optimal paths

By an optimal path it is meant the best path between two points with respect to some restrictions. It might be the least time consuming path or the shortest path between two positions, given some obstacles. In general, one may consider any landscape as a disordered energy landscape. Where the energy represent some measure. It could be related to the time it takes to pass the given point, the difficulty of the terrain, a financial cost, etc. The term "energy" relates to the many problems in physics and technology[14]. Optimal paths is important in a wide range of areas such as non-Newtonian flow trough porous media, Internet routing, plasticity, spin glasses, protein folding and the traveling salesman problem [13, 14]. Especially, when it comes to the functioning of the Internet, optimal paths are essential.

The Internet is a large network, consisting of computers and routers linked together. A signal's traveling time is dependent of both the physical distance between the source and destination, as well as the bandwidth at each link. Since the Internet's start in the 1970s the traffic load has increased with

an unprecedented pace [2]. In order to efficiently send information through the network, optimal paths need to be found. Internet routing uses link state routing protocols as a tool to navigate the data traffic. Within these protocols, routers interchange information about their link state with their neighbors. Then, by using an algorithm such as the Dijkstra algorithm, a shortest path tree for each node is found [2].

The optimal path between two points is defined as the path where the sum of energies along it, is the smallest. Consider a two-dimensional landscape where each point $\vec{r} = (x, y)$ is assigned an energy $t(\vec{r})$. Let P be a path of length L through this landscape. The path is parametrized by the distance l from the start point, $\vec{r}(l) \in P$. The start and endpoint of the path is $\vec{r}_i = \vec{r}(0)$ and $\vec{r}_f = \vec{r}(L)$, respectively. An optimal path between two points is defined as the most energy efficient path between the points. The optimal path is then found as the path where the total energy, T, is

$$T = \min_{\vec{r} \in P} \int_0^L t(\vec{r}(l). \tag{8}$$

There are numerous approaches on how to find such an optimal path. Models have been built on different algorithms.
One approach comes from the direct polymer problem [8]. In the directed polymer problem one studies the different configurations of a polymer. The polymer binds locally to a disordered substrate and is not allowed any back bends, hence it is called directed. In the general problem one take into account all possible configurations when calculating the Boltzmann weight of a polymer between two points [8]. However, at zero temperature one only needs to consider the configuration of lowest energy. That is, at zero temperature the directed polymer problem is directly equivalent to the problem of finding an optimal path. Another approach is to use algorithms for maximum flow on a network [13]

2.3.1 The Dijkstra algorithm

A third approach is presented by Schwartz et. al. [13] in *Optimal path in two and three dimensions*. In the paper, the Dijkstra algorithm from graph theory is implemented [13]. With the Dijkstra algorithm one may find the optimal path from a given point to all other points on a grid. Every node on the grid is assigned an energy value, initially set to infinity, and every edge between nodes are assigned energy values from a given distribution. Further the algorithm treats the data in three sets. The first set consists of nodes where the optimal path to the start node has been found. The second set consists of points that have been relaxed one or several times, but the optimal path to the start node has not been found. The nodes that have not yet been considered are in the third set.

After the grid (nodes and edges) has been initialized, a start point is chosen. The energy of the start node is set to zero and inserted into the second set. When all nodes, including a chosen start point are initialized, the algorithm enters the main loop. The main loop consists of two parts. In the first part, the minimal energy in the second set is detected. After the node with minimal energy is found, it is added to the first set. The second part of the main loop is the relaxation process. This process considers the nodes adjacent to the node added to the first set. The energy of the adjacent node is then compared to the sum of the added node from the first set and the edge between the two nodes. That is, if E_i and E_j is the energy of node i and j, i is the node added to the first set and the edge between them have an energy T_{ij}. The algorithm compares E_i and $E_j + T_{ij}$. If the sum is the smaller, four steps are carried out [13]:

(I) the sum is assigned to the adjacent node,

(II) a path is constructed between the nodes,

(III) if node j belongs to the second set, its previous paths to other nodes are removed,

(IV) if it does not belong to the second set, it is added to it.

2.3.2 The Bellman-Ford algorithm

In general, algorithms such as the Dijkstra algorithm, are so called *relaxation algorithms* or *label correction algorithms*. That is, as described in 2.3.1, every node is assigned a label which iteratively is corrected until the optimal value is found. In this class of algorithms one finds the similar Bellman-Ford algorithm [3]. The Bellman-Ford algorithm finds the shortest path from a given node, to all other nodes on a directed weighted network. If the distances in the network are viewed as energies, the shortest path is equivalent to the optimal path. For each node i, two values are stored: an unspecified $P[i]$ and a temporary distance $D[i]$. The distance $D[i]$ is the temporary distance between the start node s and i. More precisely $D[i]$ represents the upper boundary of the real distance d_{is} between i and s. Initially the distances are set to infinity, that is $D[i] = \infty, i \neq s$ and $D[s] = 0$.

For each edge between any two nodes i and j in the network, the values are updated by a relaxation process. First the value of $D[i]$ is compared with the sum of $D[j]$ and the length of the edge (between i and j), E_{ij}. If the sum is smaller, then $D[i]$ is assigned the value of the sum and $P[i]$ is set to j. As soon as $D[i]$ is equal to the correct distance from s to i, node i is said to be *accurate* [3]. Hence, before the relaxation process starts, s is accurate, while the remaining $i \neq s$ are not. The relaxation step is said to be *correct* if it is performed on an edge between i and j that lies on the shortest path between s and i. From this definition it follows that if j is accurate while i is not, a correct relaxation will make i accurate. It is evident that for each iteration of all nodes, at least one correct relaxation must take place. If N is the number of nodes in the grid, it will take $N-1$ correct relaxations before all distances are correct. In other words, the worst case scenario would give a total of $N-1$ iterations. After all edges have been correctly relaxed and all distances are correct, the values $D[i]$ will point to its predecessor on the shortest path [3].

3. Method

3.1 Description of methods

The methods used in the thesis consist of several parts. However, all parts of the implementation are based on the algorithm for finding optimal paths. This section gives a description of the basic algorithms, whereas section 3.2 explains how these algorithms are being applied.

3.1.1 Finding the optimal path

In this thesis, a hierarchy of optimal paths is studied. The whole hierarchy of paths is referred to as a pathscape. The hierarchy is found by implementing the iterative algorithm introduced by A. Hansen and J. Kertèsz in *Phase Diagram of Optimal Paths* [6]. The algorithm was further generalized by L. Talon et. al. [14] to identify optimal paths on a lattice. As for the Bellman-Ford and Dijkstra- algorithm, this algorithm may also be considered as a label correcting algorithm (sec. 2.3.2).

Consider a lattice based on an energy landscape. Let $\vec{r}_i$ and $\vec{r}_j$ be two neighboring nodes connected by $\vec{r}_{i,j}$. A threshold energy, $t_{i,j}$, is assigned to each link. Further, a variable V_i is assigned to each node $\vec{r}_i$, initially $V_i = 0$ for all i. For the nodes on the boundary of the lattice, the values V_i stays fixed. The nodes in the interior of the lattice, are iteratively updated. The updating process is given by the equation

$$V_i \to V_i = \min_{j(i)}(t_{i,j(i)} + V_{j(i)}), \tag{9}$$

where $j(i)$ denotes neighboring nodes of node i.
Figure 1 shows a general lattice. Between the two points i and j, the idea of a threshold energy ($t_{i,j} = t(i,j)$) is marked by a blue line. The neighbors of i are marked with colored dots, red for nearest neighbors and green for second nearest.

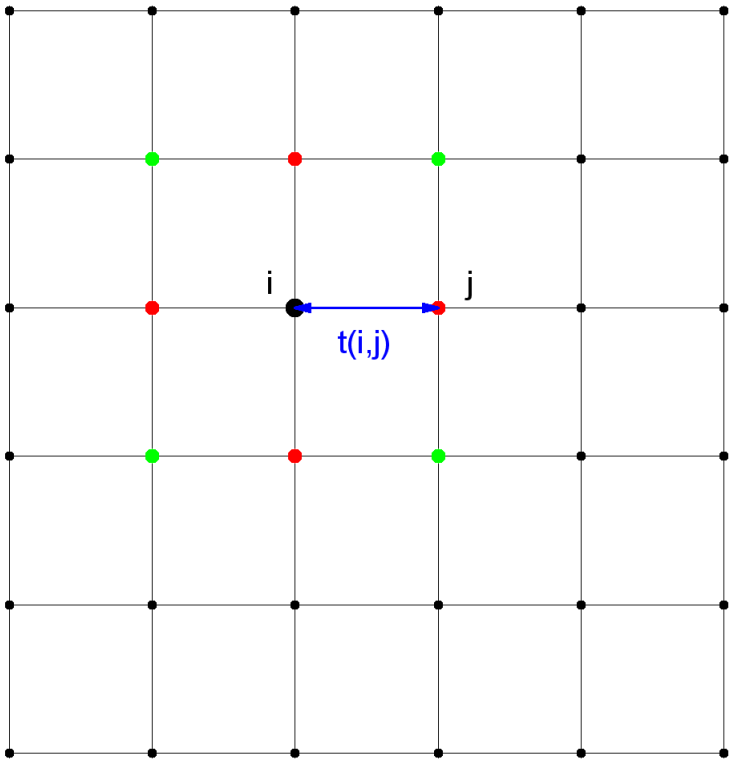

Figure 1: A general lattice. Two neighboring points (i and j) are shown with their associated threshold energy $t_{i,j} = t(i,j)$. The nearest neighbors of i are marked with red dots, while the next nearest are marked green.

However, the updating scheme used in this thesis is further specialized compared to Eq. 9, as the optimal path is allowed to move along the cell-diagonal. That is, the updating process needs to consider both the nearest and second nearest neighbors, in total 8 neighbors. Since the diagonal is a factor $\sqrt{2}$ longer then the horizontal and vertical edges, the threshold energies must be scaled accordingly. Eq. (9) is redefined as

$$\begin{aligned} V_i \rightarrow V_i &= \min_{j(i)}(k_{j(i)} t_{i,j(i)} + V_{j(i)}), \\ k_{j(i)} &= \begin{cases} 1 \text{ if } j(i) \text{ is a nearest neighbor to } i, \\ \sqrt{2} \text{ if } j(i) \text{ is a second nearest neighbor.} \end{cases} \end{aligned} \tag{10}$$

After N updates, the variable V_i contains the sum of threshold energies along the optimal path of length N originating from $\vec{r}_i$. Consider now a node $\vec{r}_0$ on the boundary of the lattice. In order to find the optimal path from an internal node $\vec{r}_i$ to $\vec{r}_0$, one sets the value $V_{\vec{r}_0}$ to zero, whilst for the remaining boundary nodes the value $V_{\vec{r}_b}$ is set to a large value M. Next, the updating process for the internal nodes is carried out according to Eq. (10), until all values V_i does no longer change. When all of the values have reached a constant, all paths have reached the only node with V_i fixed at zero, namely $\vec{r}_0$. In this case the variable V_i will contain the sum $T_{i,0}$ as defined in Eq. (8) for the optimal path between $\vec{r}_i$ and $\vec{r}_0$. If the same process is repeated for some other boundary node $\vec{r}_1$ and interior node $\vec{r}_j$, one gets a new sum, $T_{j,1}$. By the same process one also calculates the values $T_{i,1}$ and $T_{j,0}$. Now the total energy of the optimal path between nodes $\vec{r}_0$ and $\vec{r}_1$ passing through $\vec{r}_i$ may be expressed as

$$T_{0,i,1} = \min_{j(i)}(T_{0,i} + t_{i,j(i)} + T_{j(i),1}, T_{0,j(i)} + t_{j(i),i} + T_{i,1}). \tag{11}$$

3.1.2 Creating an energy landscape

An energy landscape where the objects move around, is created from previous position data. The datasets that are used contain longitude and latitude coordinates on the earths surface. The problem is simplified by turning the area of interest into a square grid in two dimensions.

The earth is being approximated by a sphere with radius r_E. Further the area of interest is approximated by a flat square. If $long_{min}$ and lat_{min} denotes the lower longitude and lower latitude value respectively, the point $(long_{min}, lat_{min})$ is the origin of the introduced coordinate system. Let the distance (in meters) between $long_{max}$ and $long_{min}$ be denoted x_{max}. The dataset is then transformed through the following expression for the longitudinal coordinates

$$x_i = \frac{\pi r_E}{180} \frac{N-1}{x_{max}} (long_i - long_{min}), \tag{12}$$

and similar for the latitude,

$$y_i = \frac{\pi r_E}{180} \frac{N-1}{y_{max}} (lat_i - lat_{min}). \tag{13}$$

This means that $x_i, y_i \in \{0, N-1\}$. Hence, the dataset is transformed from an area spanned by $[long_{min}, long_{max}] \times [lat_{min}, lat_{max}]$ with dimension $[deg]$ to a dimensionless area of size $N \times N$. The algorithms that are used in this thesis, demands that the physical length between nodes is of constant length in both the longitudinal and latitudinal direction. That is, the area that are studied are constructed as a square.

The area of interest is further discretized by introducing a grid with cell size 1×1. Each grid point (i, j) is assigned a density $\rho_{i,j}$. The density is given by the number of data points within a cell length distance of the point. If a data point, $(x, y)_{i,j}$, is located in the cell with lower left corner (i, j), it gives a contribution not only to the density in (i, j), but also to the other cell corner points $((i, j+1)$, $(i+1, j)$ and $(i+1, j+1))$. A general grid cell is shown in figure 2 with random placed object coordinates within it.

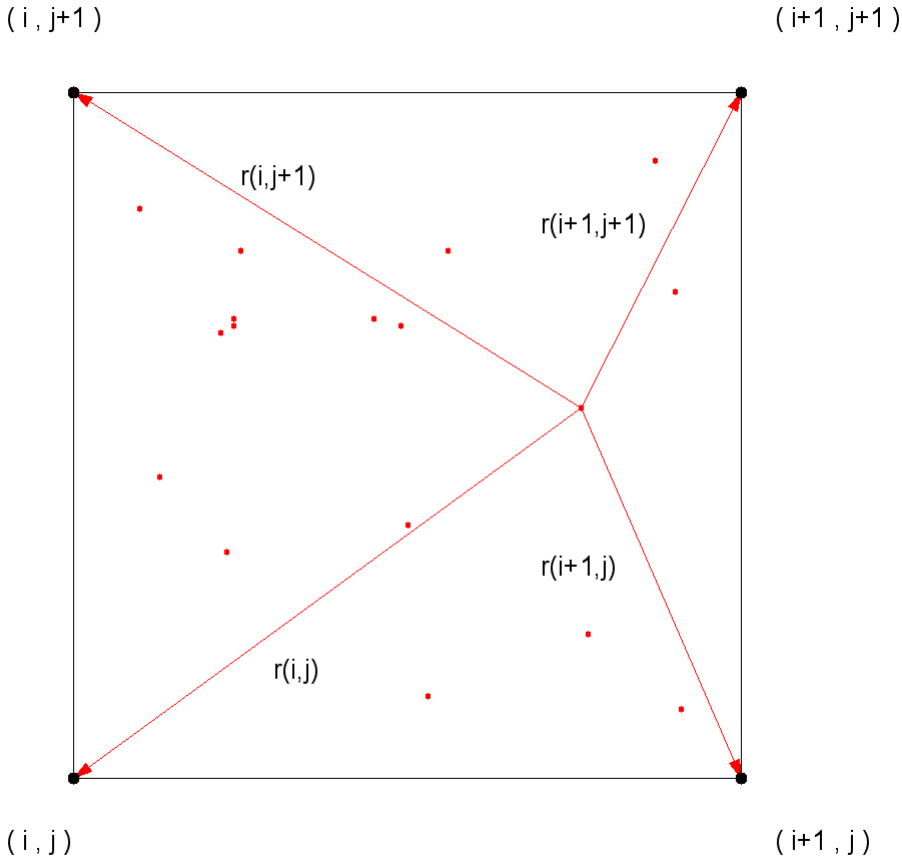

Figure 2: A general grid cell, the coordinates (red dots) within the cell are randomly placed. For a given point, the distance to each corner is illustrated by vectors.

For each corner, the contribution to its density is determined by the distance between the data point and the corner. In Figure 2 the distances between a coordinate within a cell and the cell's corners are shown. Let the four corners of a cell be denoted by $1, 2, 3$ and 4, and the corresponding distances between a data point within the cell and a corner, be denoted $r_k, k = 1, 2, 3, 4$. If $R = \sum_{k=1}^{4} r_k$, the weighted contribution to the density of the k'th corner is

$$W_k = \frac{1}{3}\frac{R - r_k}{R}. \tag{14}$$

The factor 1/3 is introduced to normalize the weights, that is $\sum_k W_{k=1}^4 = 1$. Further the density $\rho_{i,j}$ in the point (i, j) is the sum over all weights $W_{i,j}$ associated with the point. As Eq. (14) shows, the data point contributes the most to the density in the closest corner, second most to the second nearest corner and so on.

The result of this process is to locate the most trafficked areas as those with highest density. In the theory of optimal paths one aims to find the path with lowest cost, ie the most energy efficient path. Therefore, each grid point (i, j) is associated with an energy $e_{i,j}$, given by

$$e_{i,j} = (\frac{1}{\rho_{i,j}})^{\alpha}. \tag{15}$$

According to this definition, the most trafficked areas are identified as regions with low energy. The exponent α is used to adjust the magnitude of the differences in the landscape. The use of α will be further discussed in section 3.1.3.
For further use, the energies must be represented as threshold energies linking the adjacent nodes. For simplicity, let e_i denote the energy in node i, and $e_{j(i)}$ denote the energy in $j(i)$. As before, $j(i)$ denotes a node adjacent to node i. In order to have symmetry between the threshold energies $t_{i,j(i)}$ and $t_{j(i),i}$, this energy is chosen to be

$$t_{i,j(i)} = \frac{e_i + e_{j(i)}}{2}, \tag{16}$$

hence, $t_{i,j(i)} = t_{j(i),i}$.

3.1.3 Adjusting the landscape

By introducing the exponent α in Eq. (15), one adjusts the differences throughout the landscape. From the definition of the energy in Eq. (15) one finds that $\alpha = 0$ and $\alpha = 1$ are critical values. $\alpha = 0$ results in equal threshold energies throughout the grid, so that the optimal path between any two points will be equal to the shortest path between them. As the value of α is increased, the optimal path found by the proposed algorithm in section 3.1.1, moves further away from the shortest path. That is, as α increases, more weight is given to the historical traffic picture. When $\alpha = 1$, Eq. (15) shows that the energy is the inverse of the density in a given point. If α is between 0 and 1 the landscape is leveled out, while values above 1 increases differences.

3.1.4 Calculating the length of a path

In order to calculate the length of an optimal path, an additional algorithm is used. This may seem like a trivial task, however a few conditions need to be taken into consideration. If a path is allowed to traverse a cell in the grid along the diagonal as well as along the edges, the links in the path are of different length. That is, there are links of length 1 and of length $\sqrt{2}$. Hence, the length of the path is not directly known from the number of nodes in the path. Another obstacle is the fact that the endpoints of the path is not necessarily known. This is the fact when using intervals of possible endpoints for the pathscape to detect patterns, as described in section 3.2.2. Since the only restriction is that the endpoints must be inside the given intervals, the exact nodes are not known. The algorithm must therefore be able to find the ends of a given path.

When the files containing the coordinates of a path are created, the coordinates are listed by a double loop (first in the x-direction, then in the y-direction). Hence, the list is sorted with respect to the grid, not to the position in the path. The algorithm searches this list, finds an end and creates a new list. The new list is sorted from end to end along the path. To find the ends, the algorithm makes use of the fact that the distance between neighboring coordinates is either 1 or $\sqrt{2}$. For each point, the number of other points within a radius of $\sqrt{2}$ (*neighbors*) is detected. Accordingly, the endpoints will have only 1 neighbor and the remaining points will have 2.

Further, the algorithm selects one of the ends as a start point. From this point, the next step on the path is found by searching for points within a distance of $\sqrt{2}$. The process is then repeated finding the next step and so on. To prevent the algorithm from oscillating between two points, all coordinates are assigned new values lying outside the $N \times N$ grid as soon as they have been included in the path. When a point is found to be the next in the path, it is stored along with the distance to its predecessor. The total distance of the path is then found as the sum over all intermediate distances.

This algorithm has proven to fail in given situations, and are not applicable to a general case. However, in simple cases where the endpoints are known and the direction is stable, it is a useful tool. Therefore, it is applied for generating the results of section 4.1. For more complex problems such as pattern detection in section 4.3, the construction of an optimal path is more restricted, which gives a simpler calculation of the pathlength.

3.2 Application

The algorithms have been applied in two different ways to gain knowledge about a system of moving objects. Finding an optimal path yields possibilities to obtain regular patterns of motion and prediction of the route of a single object. That is, the method may be a learning stage (section 2.2.1) in the process of predicting motion, or it may be used as both a learning stage and as a prediction stage.

3.2.1 Predicting a single route

The process of predicting a single route is the most straight forward use of the algorithms. In order to predict a route, both a start point and destination is needed, as the pathscape algorithm works between two defined points or areas. Since the algorithm is constructed to work on a square area, such an area is constructed from these points. As described in section 3.1.1, the endpoints must be located on the boundary of the area.

Let $(long_i, lat_f)$ be the start point and $(long_f, lat_f)$ be the destination point. Further, consider the change in longitude and latitude, $\Delta long = |long_f - long_i|$ and $\Delta lat = |lat_f - lat_i|$. The dimension of the grid is then set to $max(\Delta long, \Delta lat)$. For the time being, assume $\max(\Delta long, \Delta lat) = \Delta long$. Also, let $\overline{lat}$ be the average of lat_i and lat_f. A square area with the two points on the boundary is then given by the four values

- $long_{min} = \min(long_i, long_f)$,
- $long_{max} = \max(long_i, long_f)$,
- $lat_{min} = \overline{lat} - \frac{1}{2}\Delta long$,
- $lat_{max} = \overline{lat} + \frac{1}{2}\Delta long$.

The four values are then used as input for the algorithm, creating the energy landscape as described in section 3.1.2. In the case where $\max(\Delta long, \Delta lat) = \Delta lat$, $long$ and lat are switched in the above expressions. For the special case where $\Delta long = \Delta lat$, the area is simply the square with diagonal between the start point and destination point.
As soon as the area has been set, the energy grid is initialized using the historical data within the given area. Next, a pathscape is created over the

area with the two given endpoints. From the pathscape, the optimal path is located by isolating the nodes where the resulting energy is smallest.

3.2.2 Finding patterns of motion

When applying these algorithms as an approach to find the regular patterns of motion, the idea is to combine the results of multiple pathscapes. However, a full survey of the area would be a lengthy process. Instead of finding the best paths by searching through combinations of all points on the boundaries, the boundaries are split into intervals. By iteratively creating pathscapes between intervals on the boundary of the area, a set of optimal paths are made and stored. Every optimal path created has an energy and length associated with it. The paths are sorted by their energy, weighed against their length raised to some power β. That is, the energy of the paths are rewritten

$$E'_{path} = C_p \frac{E_{path}}{(L_{path})^\beta}, \tag{17}$$

where L_{path} is the length of the path, and E_{path} and E'_{path} are the original and weighed energies, respectively. The constant C_p is introduced to further help separate paths.
By doing this, one finds the paths that are most optimal on a *global* basis. The idea is that the global most optimal paths represent the regular patterns present in the area. As described in section 3.1.1, the energy is the sum over all threshold energies along the path. This means that a short path through a high energy region may have the same total energy as a longer path through a low energy region. For this reason, the scaling against the path's length is necessary to isolate the regular motion patterns. By increasing β, one can make longer paths more favorable.

As before, the grid that is being used is of dimension $N \times N$. Let L_I be the length of the intervals used for the analysis. Assume this gives a total of n intervals, with $n-1$ of length L_I. The last interval is then of length $N \mod L_I$. Since the grid is a square, the intervals will be equal in both directions. Now, pathscapes are created and optimal paths are extracted by iteratively running the algorithms for any pair of intervals on

different boundaries. That is, if one starts at a given interval at the lower boundary, one finds the optimal path to all intervals on the left, right and upper boundary. Since the optimal path is not a directed path, going from interval i to interval j is equivalent to going from j to i. Consider starting the iterations by creating pathscapes from the lower boundary. There are then 3 possible edges for the destination interval, each of the edges having n intervals. Next, one starts at the left boundary, from here there are only 2 edges for the destination, as the combination between the lower boundary and left boundary has already been used. Further, consider starting at the upper boundary. The only unused combination is now upper to the right boundary. This gives a total of $(3+2+1)n^2 = 6n^2$ pathscapes.

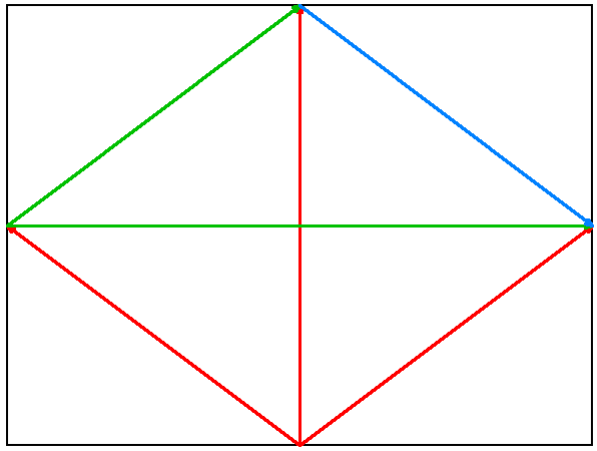

Figure 3: The schematic setup of the iterative process, showing the 6 different combinations of edges.

The process is carried out by creating sets of n^2 pathscapes for each combination of edges. Figure 3 shows the schematic buildup for the combinations of edges. For each patscape the optimal path is located and its energy and length is stored. The energy is scaled according to Eq. (17) and stored. Further, the scaled energies for all $6n^2$ optimal paths are sorted from lowest to highest. This means that the number of intervals gives a maximum of possibles paths to make out a pattern. That is, the number of intervals will influence and restrict the results. In additon to the number of intervals, their position relative to the cluster's endpoints, will have an impact on the results. As each pair of intervals only generates one optimal path, one has

the risk of leaving out clusters from the pattern. For instance, if two distinct clusters of trajectories are going between the same two intervals, only the most optimal of them will be forwarded as one of the $6n^2$ possible global optimal paths. Thus, one of the clusters are lost, and will not be present in the resulting pattern. Therefore, one will typically need more intervals for more complex patterns.

For this application, the updating process are performed according to Eq. (9). That is, the path is only allowed to move between nodes along the edges. By restricting the updating process, one has that the length of a path is directly given by the number of nodes within it. If an optimal path between two intervals traverse N_{path} nodes, the length of the path is given by

$$L_{path} = N_{path} - 1. \tag{18}$$

This is the length that is used for the scaling of energy introduced in Eq. (17).

4. Results

The data sets used for simulations in this thesis are AIS-coordinates. Throughout the result section, two real datasets are studied. These sets are denoted A and B. Figure 4 and 5 shows the shape of the datasets. In section 4.1, a random initialized grid is introduced to compare the result.

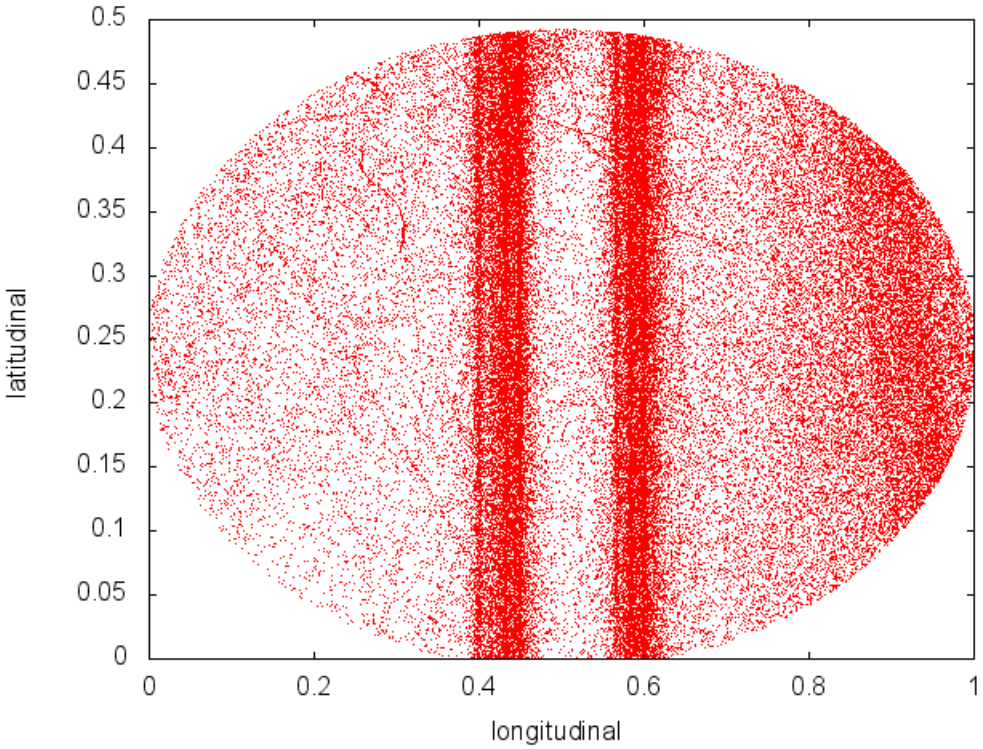

Figure 4: *Set A*: Plot of AIS-coordinates in an area for a given time frame. The coordinates are represented relative to the minimum longitudinal and latitudinal values.

As Figure 4 and 5 show, the first dataset (A) consists of a simpler traffic pattern than B. While A is dominated by two clusters of routes moving vertically through the center of the area, B includes a more complex motion pattern consisting of several clusters.

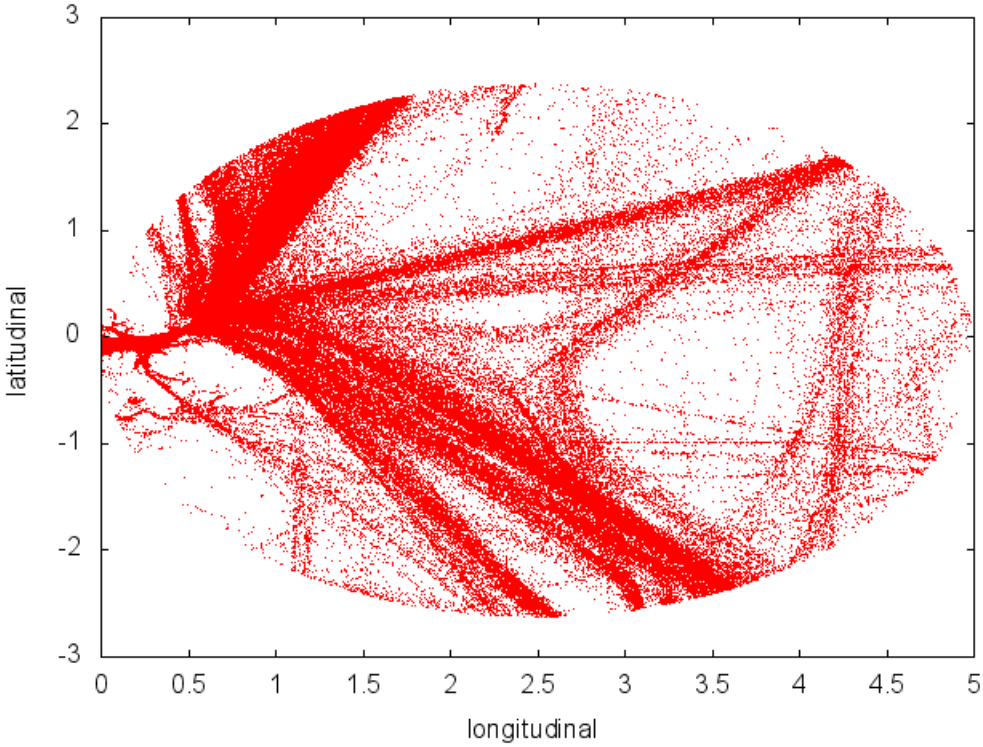

Figure 5: *Set B*: Plot of AIS-coordinates from another area for a given time frame. The coordinates are represented relative to the minimum longitudinal and latitudinal values.

4.1 The effect of the exponent α

In order to visualize the effects of α, a series of simulations have been performed. The simulations consist of finding an optimal path along the main diagonal for different values of α. As described in section 3.1.3, one would expect the optimal path to be the main diagonal itself for $\alpha = 0$, and that it moves further away from it as α is increased. The simulations have been performed for sample spaces that are taken from A and B. The simulations have also been performed on a grid initialized with random energy values. In addition to visualizing the optimal paths, their length as a function of α are also studied.

The algorithm that performs this process consists of two main steps. First, it creates an area as described in section 3.1.2, before a pathscape is created (sec. 3.1.1) for different values of α.

4.1.1 Random grid

Before the two datasets (A and B) are studied, consider an area initialized with random coordinates. The exponent α is used in the same way as before. That is, the magnitude of the energy fluctuations increases with increasing α.

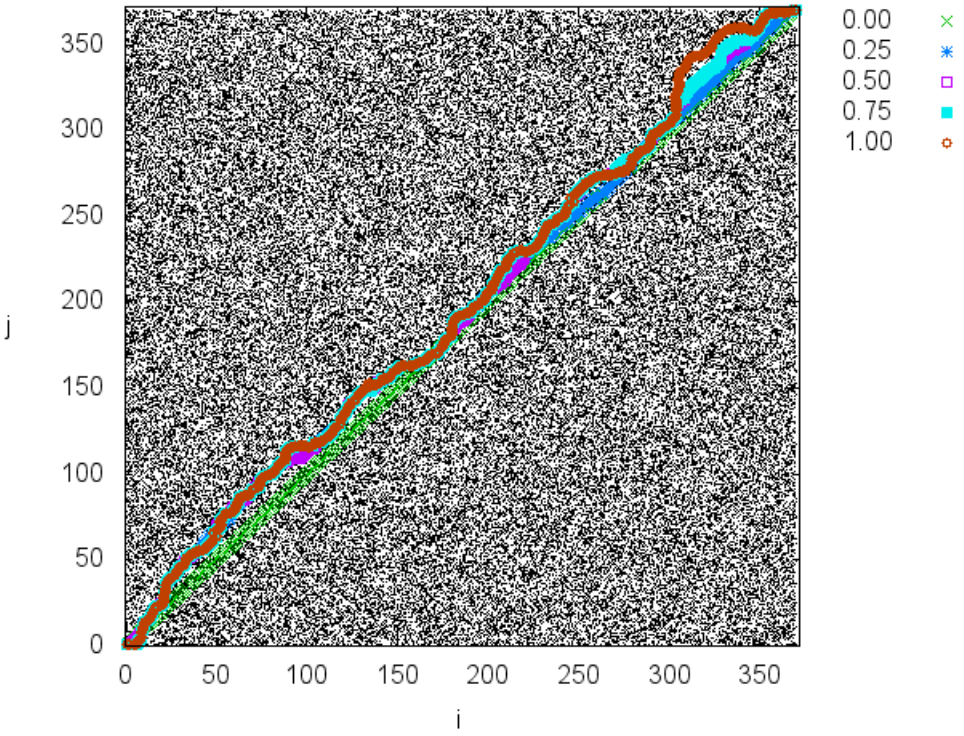

Figure 6: Optimal paths along the main diagonal for varying values of α. The grid is of dimension 371×371 and randomly initialized. The value of α ranges from 0 to 1 in intervals of 0.25.

Figure 6 shows the results of five paths along the main diagonal for different values of α. As the figure shows, the green line that represents the optimal path for $\alpha = 0$, is the diagonal line itself. As the value of α is increased, it is seen that the paths differ from the diagonal line. In Figure 7, the pathlength is plotted as a function of α. The pathlength is scaled by the length of the diagonal, so that the value for $\alpha = 0$ is 1.

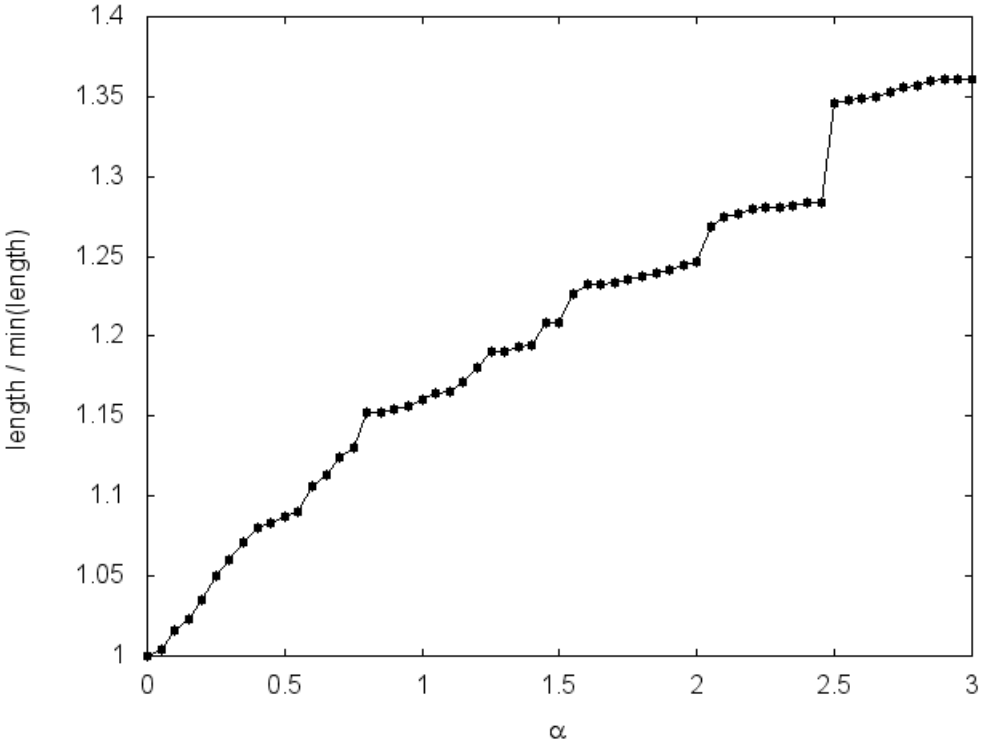

Figure 7: The pathlength of the optimal paths for varying values of α. The values are scaled by the length of the diagonal. Dots represent actual calculated values.

As expected, the length is strictly increasing for increasing values of α. Moreover, there seems to be a change in behavior at $\alpha = 0.7$. For $0 < \alpha \leq 0.7$, the curve is linear, while for $\alpha > 0.7$ the values seperates into groups of nearly constant length. This indicates that for some intervals of α, the optimal path remains approximately the same, i.e some paths are more stable than others.

4.1.2 Set A

From A, a square area which spans 0.3° in both the longitudinal and latitudinal direction is selected. The data is transformed into metric values as described in section 3.1.2. By setting the cell size to 100×100 m^2, it forms a grid of dimension 334×334. This grid is shown in Figure 8. The area is selected, so that the two vertical clusters are included. For varying values of α, the optimal path between $(0, 0)$ and (N, N) is found. The results are represented in Figure 9 and 10.

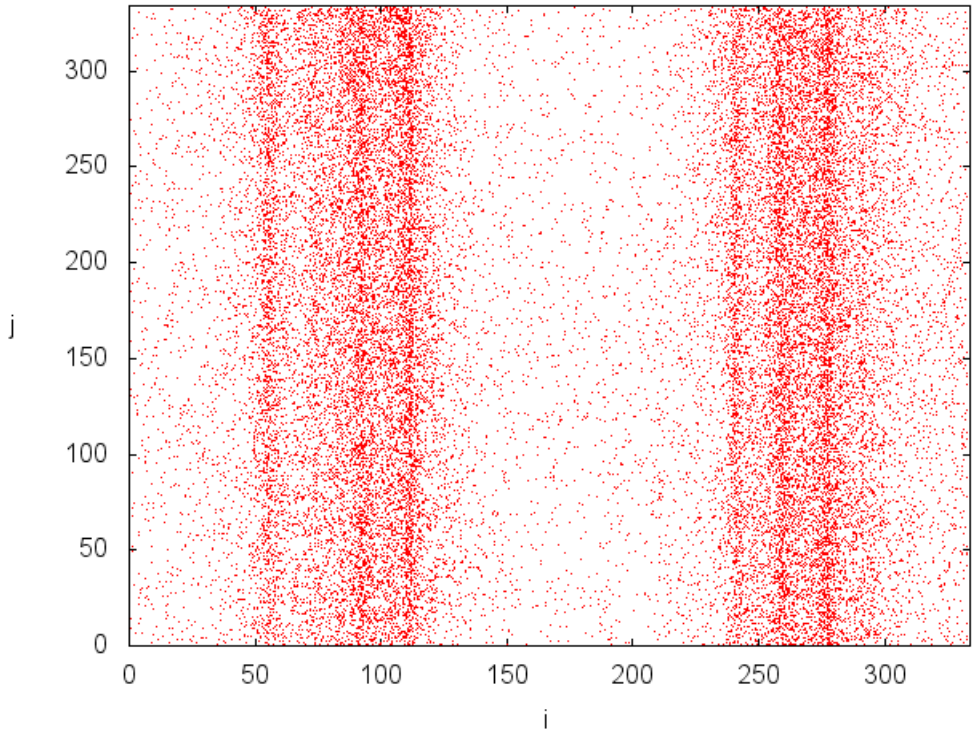

Figure 8: The selected area from data set A, spanning 0.3° in both the longitudinal and latitudinal direction. The data is transformed to a grid of dimension 334×334 with a cell size of 100×100 m^2. The area includes two large bands of trajectories moving vertically.

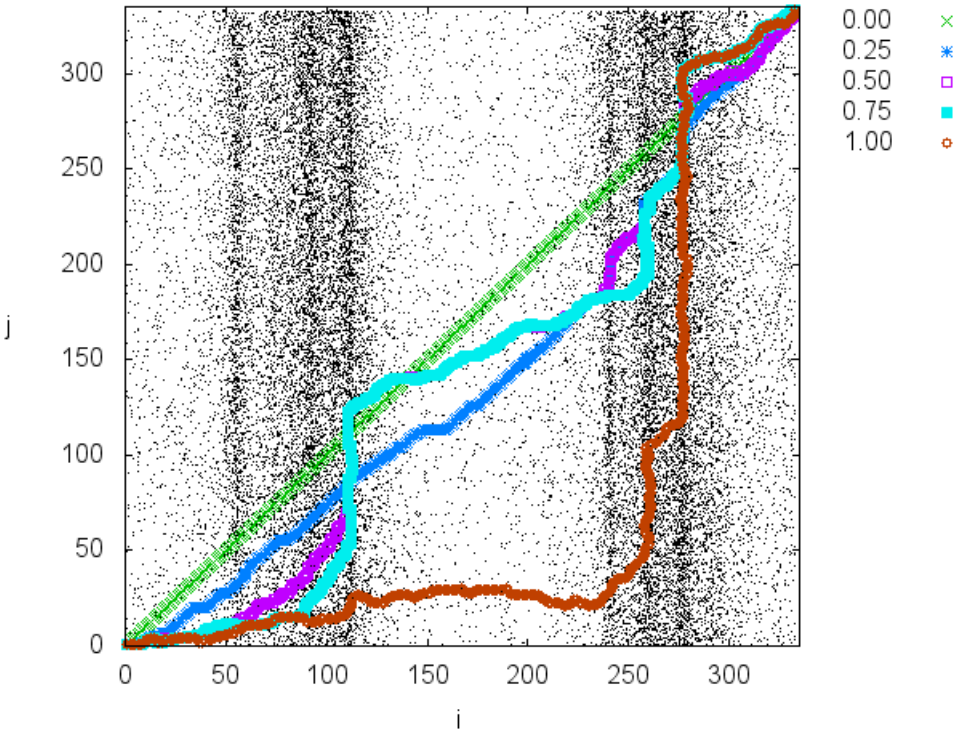

Figure 9: Optimal paths over the diagonal for varying values of α. The grid is initialized from the coordinates shown in Figure 8. The cell size is 100×100 m^2 and there are a total of 334 grid points in both directions. The value of α ranges from 0 to 1 in intervals of 0.25.

Figure 9 shows the optimal paths, for five selected values of α between 0 and 1. As for the random case, one sees that the paths differ increasingly from the diagonal. However, as α increases, one sees how the paths align with the clusters. Similarly to the random case, there is a change in behavior at approximately $\alpha = 0.75$. Figure 9 shows, that up to $\alpha = 0.75$ the optimal paths lie fairly close to the diagonal line. For the optimal path, calculated for $\alpha = 1$ (red line), the pattern is dominating. The same tendencies becomes evident in Figure 10. The curve is approximately linear for $0 < \alpha \leq 0.75$, while there is a group of paths with similar length for $0.75 < \alpha \leq 1$. As one can see from Figure 9, these paths move stepwise vertically and horizontally. The paths follow one of the two main clusters in the horizontal direction, and cross the high energy areas in nearly horizontal lines.

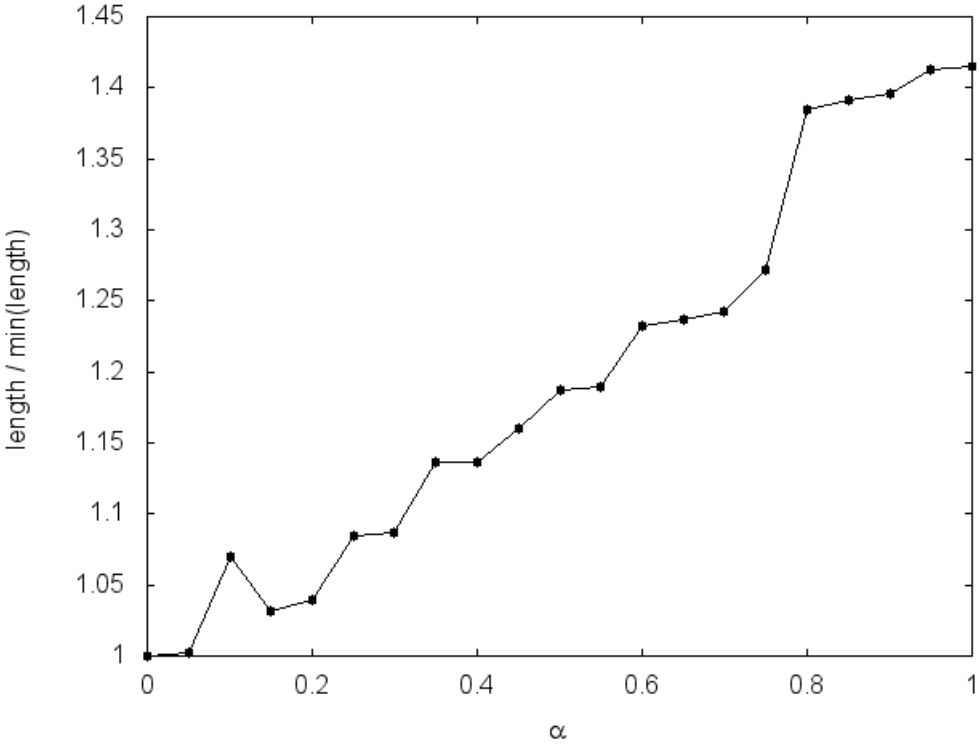

Figure 10: The pathlength of the optimal paths for varying values of α. The values are scaled by the length of the diagonal. Dots represent actual calculated values. The function is approximately linear for $\alpha \leq 0.75$ and makes a leap to an approximately constant level for $\alpha > 0.75$

4.1.3 Set B

From dataset B, an area of original size $3.0° \times 3.0°$ is selected. That is, an area which is 100 times greater than the one selected from set A. The cell size is set to 1000×1000 m^2 which gives the same grid size as for A, with $N = 334$ grid points. The selected area includes several distinct clusters, which spread out from the left edge of the area. Figure 12 shows the realization of optimal paths between $(0, 0)$ and (N, N), for the same five values of α used in the previous cases.

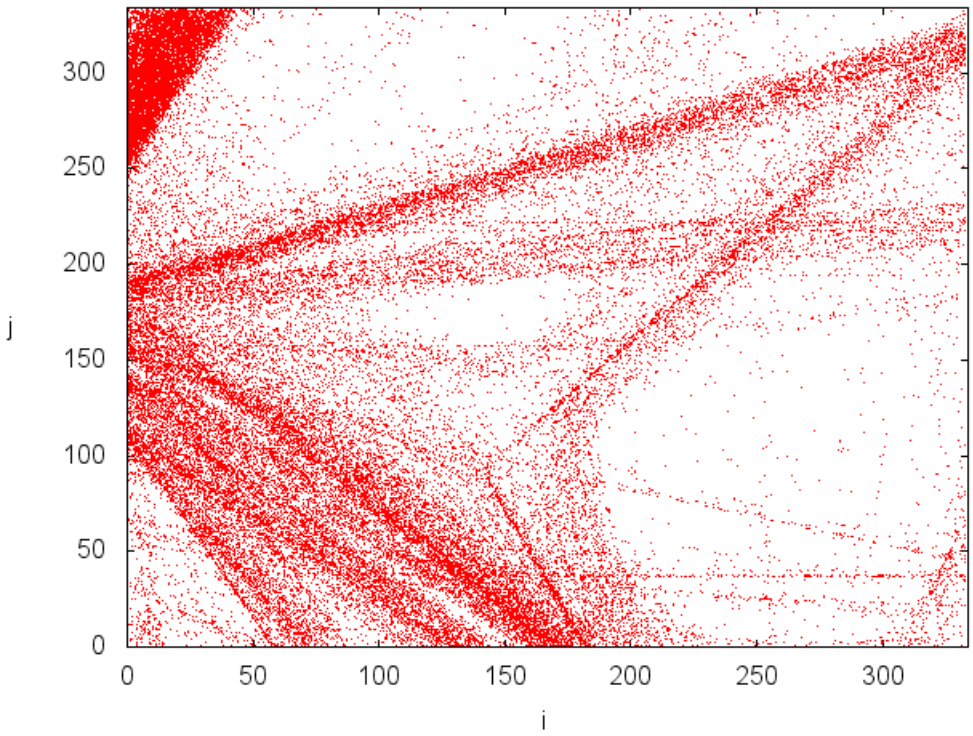

Figure 11: The selected partial area from dataset B. The original section given in geodetic coordinates, spanned over $3.0°$ in both the longitudinal and latitudinal direction. The distance between grid points are 1000×1000 m^2, which gives a grid of dimension 334×334.

In Figure 11, one sees clusters that partially move along the main diaogonal, one below the diagonal and the other above. The higher one is thicker than the lower, indicating higher density (lower energy). However, the lower one

is closer to the actual diagonal line. As α is increased, these trajectories become clear candidates for an optimal route. The result of this is shown in Figure 12. The green line shows the realization of $\alpha = 0$. As expected it is equal to the diagonal. Further, the blue line for $\alpha = 0.25$ follows one of the two clusters moving along the diagonal. As Figure 12 shows, the three remaining lines ($\alpha = 0.50, 0.75$ and 1.0) almost perfectly overlap each other. They are all following the cluster starting at the upper right corner until they reach the left edge.

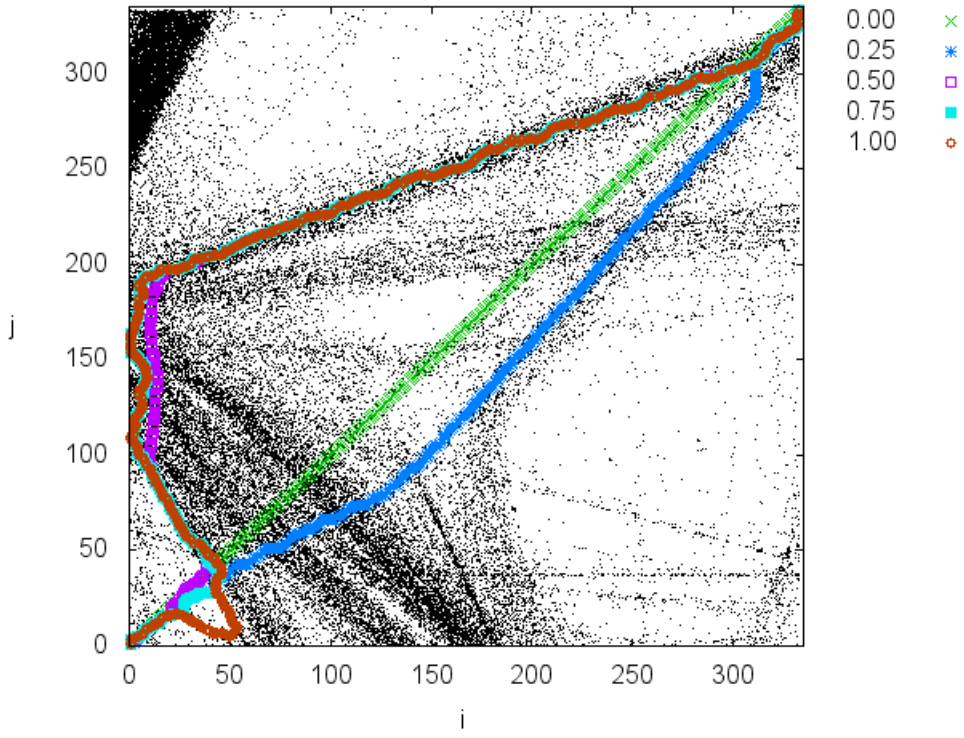

Figure 12: Optimal paths along the main diagonal for varying values of α. The grid is initialized from the data shown in Figure 11. The value of α ranges from 0 to 1 in intervals of 0.25.

In Figure 13, the pathlength is plotted as a function of α for the given section of dataset B. As the figure shows, the piecewise grouping of paths occurs already from $\alpha = 0.05$.

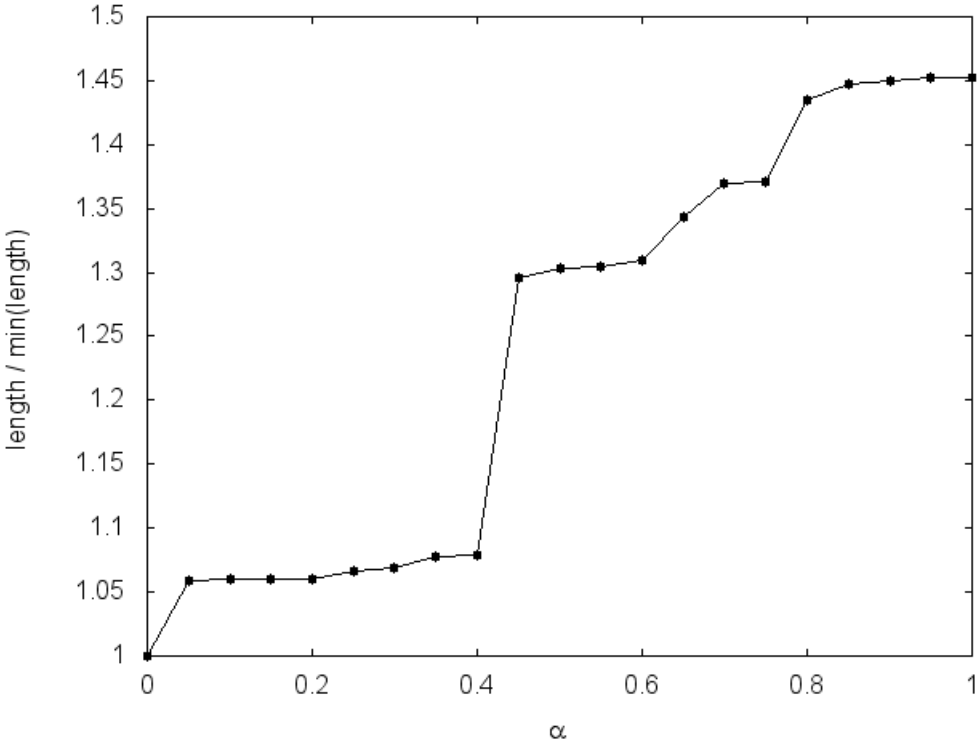

Figure 13: The pathlength as a function of α for $0 \leq \alpha \leq 1$. The calculated length is scaled by the length of the diagonal. The curve forms groups/levels of paths having the same length for different values of α. There are three distinct levels in addition to the $\alpha = 0$ level.

4.2 Estimating a single route

As described in section 3.2.1, a routine for estimating a path between two points have been made. In this section, the results of some selected trajectories through dataset A are represented. The dataset is sorted into indiviudal time series, which consists of coordinates belonging to the same sailing. In Figure 14, the result of the algorithm is shown along with the recorded coordinates for a selected sailing. The selected sailing is moving through the left cluster in set A. Figure 15 shows the predicted path, along with a section of area A. The area that is shown in Figure 15 is generated automatically by the algorithm, in order to create a square with the endpoints lying on the boundary. For both figures, a cell size of $100 \times 100\ m^2$ has been used. The exponent α has been set to 0.05.

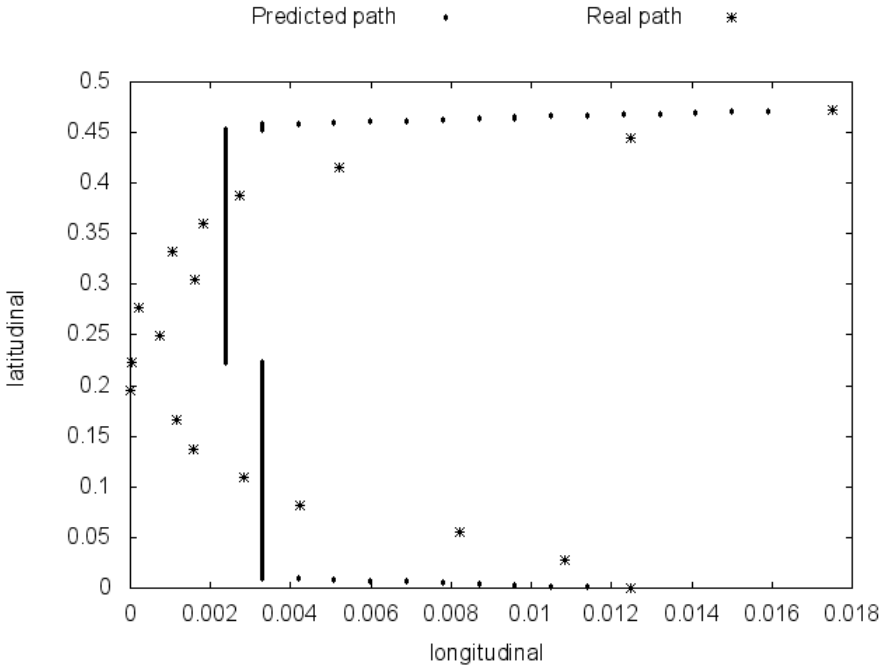

Figure 14: Predicted path for a selected sailing. The estimated coordinates are plotted with dots and the real recorded coordinates with stars. The calculations have been performed with a cell size of $100 \times 100\ m^2$ and $\alpha = 0.05$. The coordinates have been transformed by subtracting the minimum value of the set. Note the different scales on the axes.

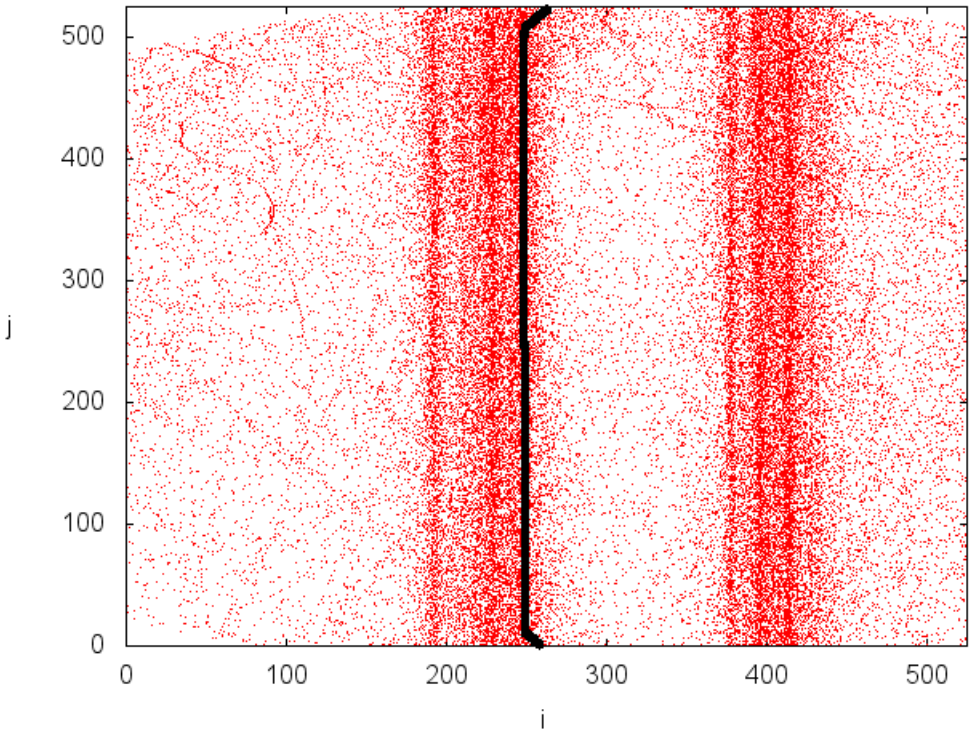

Figure 15: The predicted path from Figure 14, plotted over the dataset used for analysis. The dataset is constructed as described in section 3.2.1, and is a square grid with dimension 525×525 and cell size $100 \times 100\ m^2$.

Figure 14 and Figure 15 show an example where the estimation of the path is fairly good. As one sees in Figure 14, the predicted path and the recorded path both move left, towards the cluster shown in Figure 15. When the paths reach the cluster, they follow its vertical direction. With the same value of α, another route has been estimated. The path of this sailing lies close to the other cluster present in A. Figure 16 and 17 give the same results as before for the new sailing. The new sailing is of similar length as the first one, thus the gridsize is approximately the same.

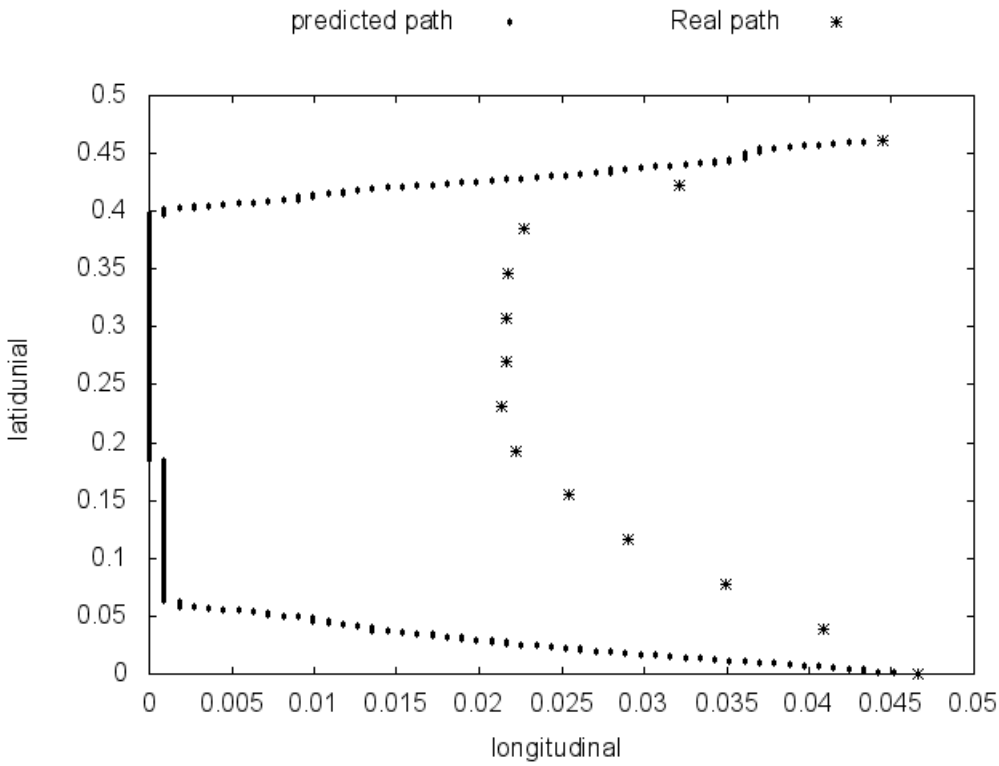

Figure 16: Predicted path for a selected sailing. The estimated coordinates are plotted with dots and the real recorded coordinates with stars. The calculations have been performed with a cell size of $100 \times 100\ m^2$ and $\alpha = 0.05$. The coordinates have been transformed by subtracting the minimum value of the set. Note the different scales on the axes.

From figure 16 one sees a similar behaviour of the paths as in Figure 14, however the estimate is not as close to the real value as before. In Figure 17 the predicted path is shown with the coordinates in the dataset. Using both Figure 16 and 17, one sees that the real path moves along the right edge of the cluster. The estimated path, that are shown in both figures, moves further into the cluster than the real value. That is, too much weight is put to the center of the cluster, so that the estimate misses its target. The four figures (Fig. 14 - 17) show that although one value of α seems to work fine in one case, it fails in another.

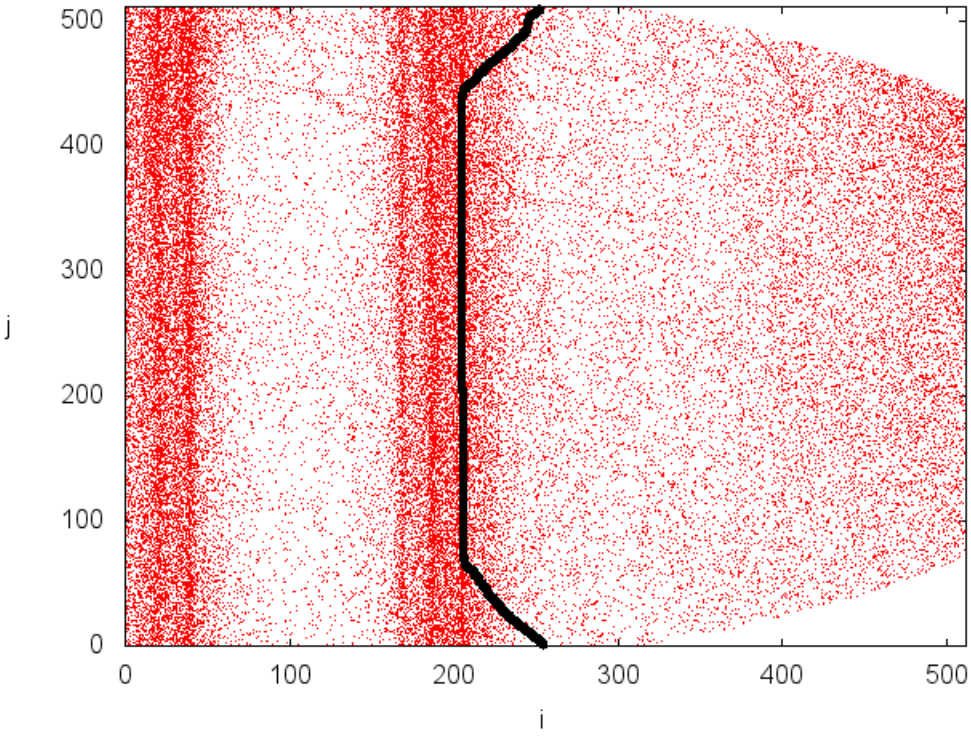

Figure 17: The predicted path from Figure 16, plotted over the dataset used for analysis. The area is a square grid with dimension 515×515 and cell size 100×100 m^2.

4.3 Finding patterns of motion

In section 3.2.2, a method for determing the regular patterns of motion in an area is described. The method consists of creating a set of pathscapes between intervals on the boundary. Each of the pathscapes generates an optimal path between the given two intervals. If n is the number of intervals on each boundary, the method generates a set of $6n^2$ optimal paths. Further, the paths are sorted according to Eq. 17, which yields a list of the global most optimal paths. This method has been applied to the two datasets A and B. To each of the datasets, different values for the involved variables are used.

In addition to the figures showing the detected patterns, there are also included graphs which show the scaled energy given in Eq. 17, for different paths. The global optimality of a path is determined using this energy. The graph will give an indication of how many of the paths that are relevant for an existing pattern.

4.3.1 Set A

The area selected for the analysis of dataset A, is slightly bigger than the one in figure 8. It is expanded by 0.1° in both directions. That is, it is now an area of size $0.4^\circ \times 0.4^\circ$. The area overlaps with the one chosen for the calculations in section 4.1, which gives the same structure. In figure 18, the result is shown. The figure is generated with a grid of size 148×148 and a cell dimension of 300×300 m^2. The boundaries have been sectioned into two intervals of length 75 and 73. From the total 24 optimal paths, the two paths shown, are found to be the two most optimal. The variable values used for the result in Figure 18, are summarized in table 1. In the table, N° denotes the original size of the area ($N^\circ \times N^\circ$), while N is the dimension of the grid and s_c is the cell size. Further, n is the number of intervals and the interval length is given by L_I as described in section 3.2.2. The three constants α, β and C_p are the same as introduced through Eq. 15 and Eq. 17.

Table 1:

$N°$	$0.4°$
N	148
s_c	300 m
n	2
L_I	75/73
α	0.8
β	0.8
C_p	10

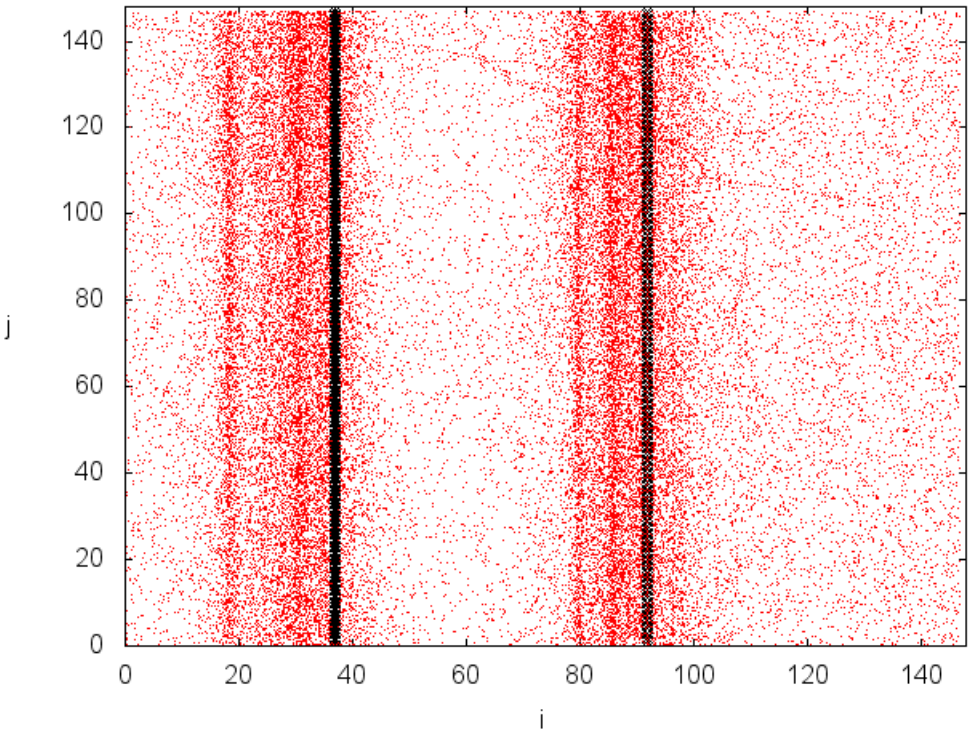

Figure 18: The result output, showing the pattern found in dataset A. The figure includes the two globally most optimal paths, which represents the regular patterns of motion in the area.

The traffic pattern in dataset A consists of two main clusters, moving vertically through the center of the area. As Figure 18 shows, the pattern is detected by the method and the clusters are represented by two black lines. These lines represents the two globally most optimal paths through the area. In Figure 19, the scaled energies of the paths are represented, sorted from minimal to optimal. As Figure 19 shows, there is a group of five low energy paths that stand out. The idea of the scaled energy, is that it may be used to determine which paths that belong in the actual pattern (sec. 3.2.2). That is, Figure 19 indicates that the five globally most optimal paths should be included to describe the pattern.

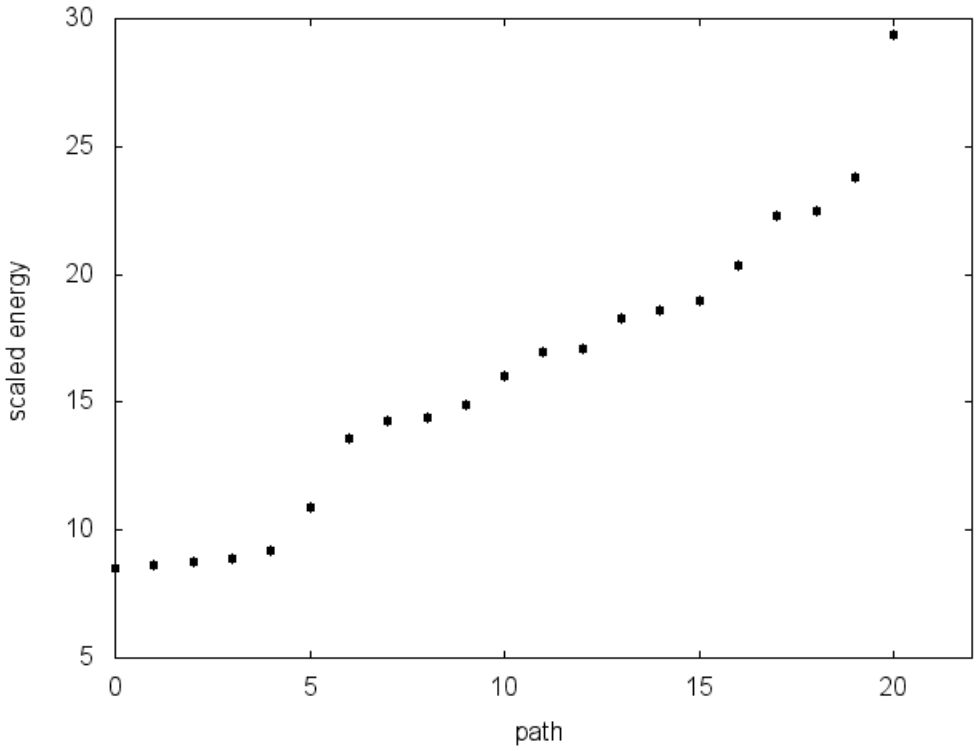

Figure 19: The scaled energies of optimal paths between selected intervals, ranging from most optimal 0 to least optimal 20. Three paths have been left out from the figure due to very high energy values.

In figure 20, the same result as in Figure 18 is shown with the whole group of five paths included. As the figure shows, the additional paths overlap with the two most optimal for most of their length. However, the remaining (non overlapping) length does not contribute to the correct pattern. Based on the look of the figures, the result of Figure 18 is regarded as being more correct than Figure 20.

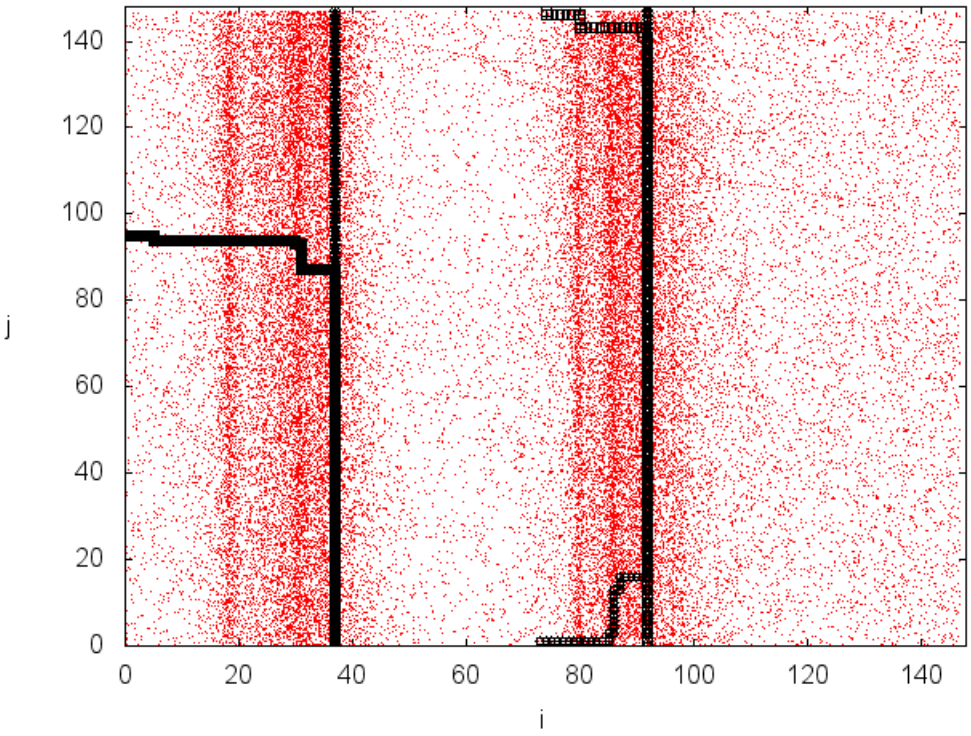

Figure 20: The located patterns in dataset A. The figure shows the same as Figure 18 with five paths included, instead of two. The values of the variables used to create the figure is the same as well, and are listed in table 1.

4.3.2 Set B

For dataset B, the section of the area that is considered is the one shown in Figure 11. That is, an area that spans 3.0° in both the longitudinal and latitudinal direction. The structure of the selected area is more complex than the one in A. Therefore, the analysis of area B is performed more thoroughly than A. The calculations have been performed, using both two and three intervals on each boundary. The first figure (Fig. 21) is the result for dataset B, using the same variables as for Figure 18. That is, the area of size $3.0° \times 3.0°$ are transformed to a grid with dimension 166×166. Further, each boundary is split into two intervals of length 90 and 76. All values are listed in table 2.

Table 2:

$N°$	3.0°
N	166
s_c	2000 m
n	2
L_I	90/76
α	0.8
β	0.8
C_p	10

As Figure 21 shows, there are several distinct clusters that are found by the method. Figure 21 actually includes the twenty most optimal paths. This means that many of the paths overlap for most of their length. Paths that are starting at closely related points at the boundary, are likely to seek the same cluster. The boundary between the intervals are at $(0, 90)$ and $(90, 0)$. Especially in the vertical direction where many of the clusters originate from, a special behavior is observed. From both the lower and higher interval, paths move along the edge in a low density region and traverse the area along two routes, one moving against the upper right corner, the other against the lower left.

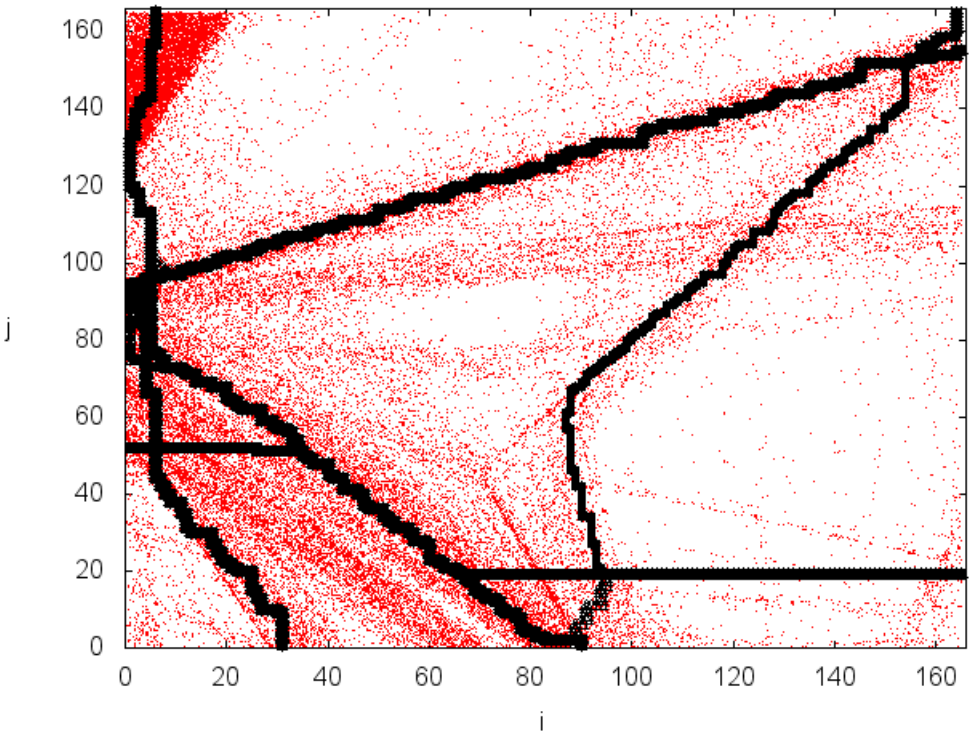

Figure 21: The pattern in the given section of B, calculated with two intervals per edge. In accordance with Figure 22, twenty paths are included in the figure.

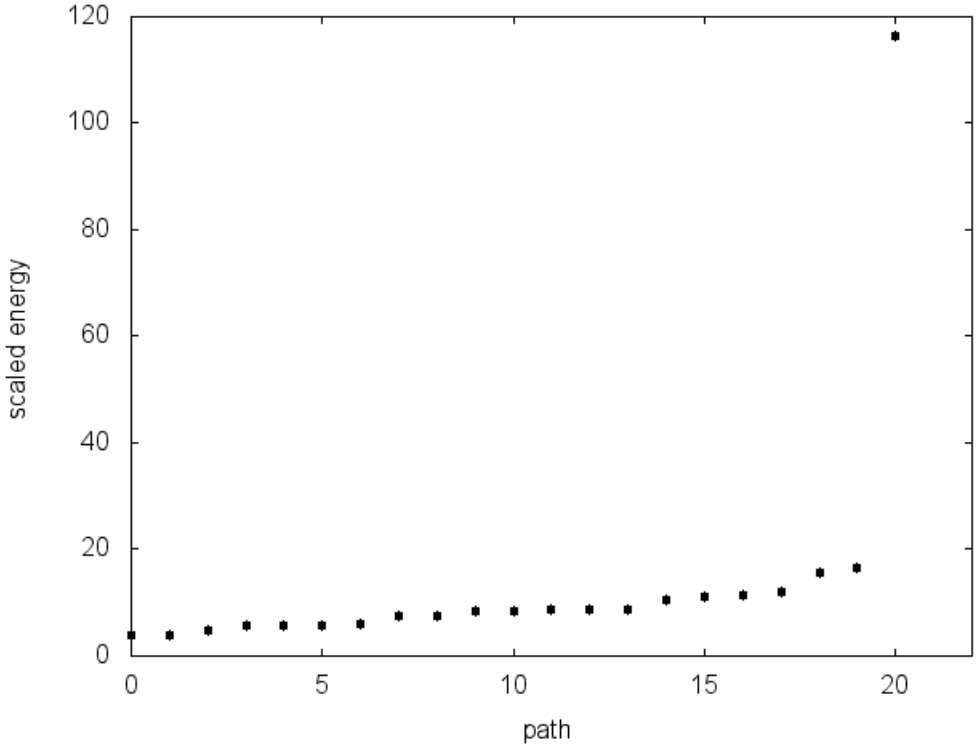

Figure 22: The scaled energy for the paths showed in Figure 21. The paths are sorted, ranging from the lowest energy to the highest. Three of the total twenty four paths are excluded, due to very high energies.

Figure 21 shows that most of the clusters of the pattern are detected. However, some obvious parts are ignored: at least one (partly two) clusters moving horizontal through the center, as well as an additional cluster between the two detected ones moving from approximately $(0, 60)$ to $(60, 0)$. The problem of detecting these clusters, are related to the number and size of intervals as discussed in section 3.2.2. Both of the clusters are being dominated by more optimal ones that share their end intervals. To address this issue, another calculation have been performed on the same grid, only with three intervals on each edge. The interval size is now set to $L_I = 60$ for the first two and $L_I = 46$ for the third. The remaining variables are held constant. Table 3 lists all variables and the results are shown in figure 23.

Table 3:

$N°$	3.0°
N	166
s_c	2000 m
n	3
L_I	60/46
α	0.8
β	0.8
C_p	10

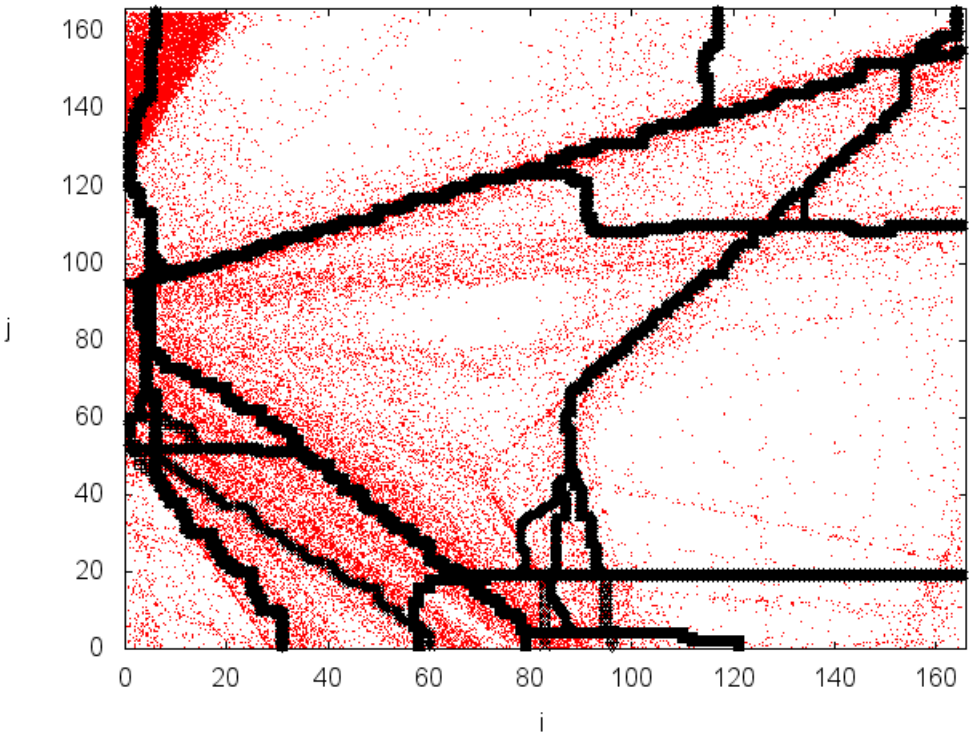

Figure 23: The pattern found in the section of B with three intervals per edge. As figure 24 suggests, a total of 48 paths are included in the figure.

In Figure 23, one sees that at least one of the two "missing" clusters from Figure 21 are covered by a black line. The central cluster of three, moving from the left edge to the lower edge, are now considered as part of the pattern. The horizontal line at $j \simeq 100$ is only partially covered by the new result. As the figure shows, the right side of a path is aligned with the cluster, while the left side are being dominated by the cluster moving against the top right corner. Still, Figure 23 shows an improved coverage of the pattern from Figure 21. However, in addition to improved coverage, the increase in included paths also leads to impurities. At the major intersections, clusters are represented by multiple lines. As for the simpler case from dataset A, one has that many of the included paths, overlap for most of their length. This creates impurities by splitting of paths near their ends.

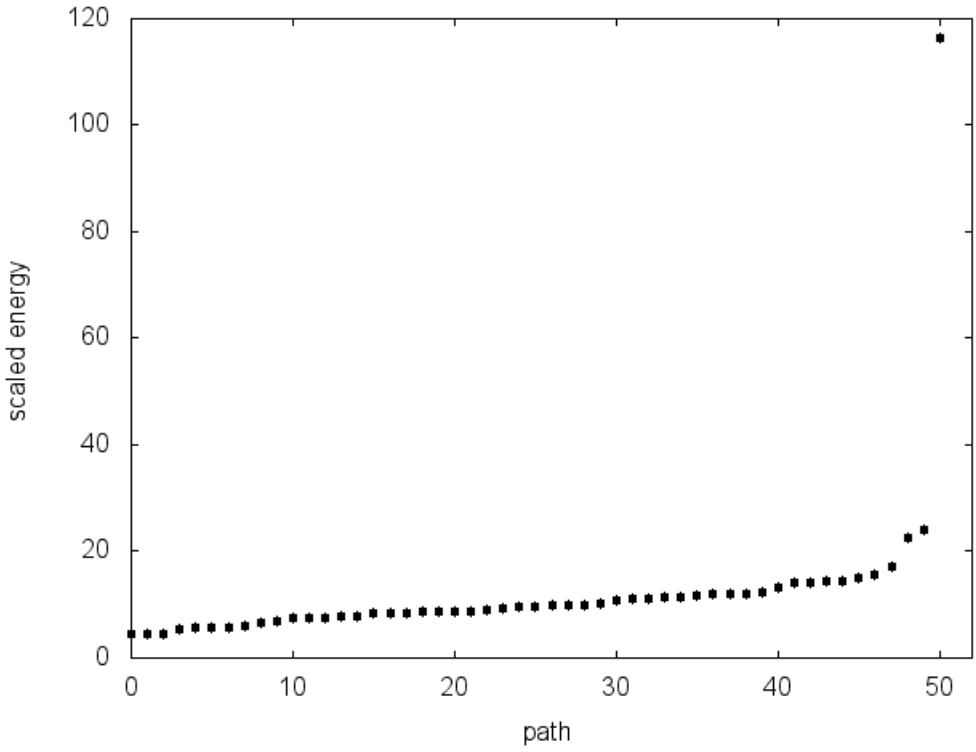

Figure 24: The scaled energies for the paths showed in Figure 23. The paths are sorted, ranging from the lowest energy to the highest. Three of the total fifty four paths are left out, due to very high energies.

5. Discussion

5.1 The exponent α

Through section 4.1, the effect of α is shown for both the patterns in dataset A and B, as well as on a random grid. For the random case two domains appear, as the value of α is increased. From Figure 7, one can see how the pathlength scales linearly for values of α up to a certain value. Above this value, groups of paths having approximately the same pathlength appear. This indicates that some paths actually stay nearly unchanged for several values of α. This property becomes more clear in the case where actual patterns exist.

In Figure 10 and 13, the same result is shown for given sections of A and B, respectively. For A, which consists of a simpler pattern than B, the scaling of the pathlength is similar to the random case. The scaling in A starts of with an approximately linear dependence. As the value of α is increased, and more weight is given the pattern, a group of paths with nearly constant length appears. These are paths that move stepwise, vertically along the clusters of the pattern and horizontal through the areas outside. If α was to be further increased, the paths would most likely converge into straight lines. The lines would move vertically along the most optimal of the two clusters and horizontal along the edges.

For the more detailed pattern in B, the behavior is somewhat different. As Figure 13 shows, the staircase form appears earlier than for the two other cases. This effect may be caused by the fact that there are more clusters present in this pattern. Another part of the explanation may be the orientation (of especially two) of the clusters. As two of the clusters are partly oriented in the same direction as the diagonal line, they obviously represent good possibilities for the optimal path. Since these clusters are close to the shortest path between $(0, 0)$ and (N, N), only a small weight given to the pattern (small values of α) is enough to direct the path through them.

The results of the three cases, show how an energy landscape created from real data, is a landscape with large differences. The landscape has deep energy valleys where the regular traffic is located, and large energy mountains where there is no traffic. It is comparable with the random case, as the

same behavior is found for different values of α. In both cases, semi stable paths appear at some point. Depending of the complexity of the present patterns and also how well defined each cluster is, the structure dominates the optimal paths already at low values of α.

5.2 Routes

In section 4.2, a few examples of the route predicting algorithm were shown. The two cases that were represented, revealed a challenge for further use of the algorithm. In order for the algorithm to be of any practical use, more knowledge about the exponent α must be gained. As illustrated by Figure 14 and 16, the same value of α did not generate good results in both cases. This thesis does not include any testing, other than visual, of the performance of the algorithm for different values of α. That is, there has not been implemented any calculations of the correctness of the predicted paths.

The exponent α is an important variable, for all applications of the pathscape algorithm. It is perhaps best illustrated when comparing an estimate for a single vessel route with its real trajectory. As described in the theory section (sec. 3.1.3) and later by the results of varying α values, the exponent weighs the impact of the historical patterns. It is clear, that a real vessel can only follow a pattern to a certain limit. In order to sail efficiently, it can only differ so much from a straight line. Therefore, when predicting single routes, relatively small values of α need to be used.

Since the energy landscape that are created for the analysis, only take into account the position data, there is no time reference in the results. That is, the method gives an estimate for the positions along the path, but does not give any information about the time of the passings. However, this simple approach does not only work as a disadvantage. By only using coordinates, the datasets are kept simple. Further, the algorithms are simpler than the traditional statistical clustering methods, as they cluster trajectories through multiple layers that demand more information [15]. A possibility to gain such information, is to combine the results with the measured speed of the object that is studied. The speed may be estimated by the average speed between the first two recorded coordinates.

Another disadvantage with this method is the fact that it takes both a start

point and a destination to perform its analysis. If the method shall be able to predict the motion through an area in advance, a destination needs to be specified. However, in some real life cases this is not such an unreasonable demand. For instance AIS - data, that are used for the work in this thesis, includes a vessels destination port [12]. However, this information is not always accurate, and in order to include a distant port of destination one would typically end up with large areas. A possible approach is to locate possible intermediate points (either by random, or estimated based on prior knowledge), and predict a path between the start point and the intermediate point. The process could be repeated, creating several intermediate points until one reaches the given destination.

5.3 Patterns

The results in section 4.3, show the potential of a pattern recognizing method based on combinations of pathscapes.The figures that are made for both dataset A and B, show that optimal paths are useful for finding regular patterns of motion. Further, the results showed that sectioning the edges into only a few intervals, were enough to generate a good estimate of the pattern. As all the points along the edges, may be represented by only a few intervals, the running time of the algorithms are kept short. There are however some issues with the method.

The idea of the method is that, with proper scaling and variable values, the patterns should be automatically detectable. For this to apply, there needs to be established a clear boundary between paths that should be included in the pattern, and those that should not. This problem is best highlighted by the figures created for dataset A; Figures 18-20. The sorted energy values (Figure 19) suggested that the motion pattern in the area consists of five paths (five clusters), whereas a visual evaluation of the dataset clearly indicated two distinct clusters. Further, it is seen from Figure 18 and Figure 20 that the difference between two and five included paths are minimal. The reason being, that the three added paths in Figure 20 overlap with the two most optimal ones, for most of their length. In other words, one may say that there are two main paths with branches near the ends, instead of five.

For the more complex structure in B, the same effect appears, especially when increasing the number of intervals. An increase of intervals from 2 to

3, yields an increase in possible paths from 24 to 54. For both the 2×2 and 3×3 case, the figures showing the scaled energies of the paths (Figure 22 and 24), were not as clear as the one for dataset A. Figure 19 showed a distinct group of 5 globally most optimal paths. Figure 22 on the other hand, shows a group of 20 paths of a total of 24, while Figure 24 gives a group of 48 out of 54. Further, one sees from Figure 23 that the overall changes to the pattern made by an addition of 28 paths is small. As more details in the pattern are localized, there are also more impurities or clutter present, for the same reason as above. Most of the added paths, overlap with the already present ones for most of their length. The paths branch out in different directions near their ends and intersections.

An idea to improve this part of the method, is to filter paths included in the pattern by their degree of overlapping. That is, for each path that energy-wise should be included in the pattern, one checks their similarity to lower energy paths. If for instance, a path is found to overlap with a globally more optimal path for some percentage of its length, it is not included in the pattern. For the simple case discussed for dataset A, the three last paths (of the five in the low energy group) would be neglected due to their similarity with the two most optimal ones. That is, this method would be applicable, at least to A.

6. Conclusion

The purpose of this thesis was to develop a method for learning motion patterns and predict future motion of moving objects, in an area with regular traffic patterns. Two main methods have been developed, and are tested on real AIS data. The methods are based on theory of optimal paths through disordered energy landscapes. By translating a trafficked area into an energy landscape, these methods could be applied to any situation involving moving objects.

The first method gives an estimate for a single object's route, while the other focuses on detecting regular patterns of motion. The methods ability to analyze traffic patterns are clearly shown in section 4.2 and 4.3. Especially the pattern recognizing method has shown good results, as it has efficiently detected the patterns at hand. Moreover, the methods demonstrate great potential for real life applications!

6.1 Further Work

Although the algorithms that have been created for this thesis show promising results, there are still questions and issues left.

Does there exist a global value of α, which yields good results in several cases?
Is it possible for the algorithms to work fully automatically? That is, what can be done to minimize the need for external inputs?

As described in section 5.2, the main issue with the route predicting algorithm is the sensitivity of α. However, it is possible that there exists a specific value of α which optimizes the performance of the algorithm for a given pattern. For the algorithm to be applied automatically, a best value of α would be necessary. If a measure of error of the estimated path was introduced, one could imagine the algorithm to be able to iteratively search for the value of α that minimizes the error.

For the pattern recognizing algorithm, the issue of overlapping paths is discussed in 5.3. The scaled energy of a path is used as a measure for whether or not the path should be included in the pattern. This method included too many paths in the pattern, making it cluttered. A suggestion to improve this is to neglect paths that overlap with more optimal ones, for a certain percentage of their length.

Further work with the methods should include a proper measure of error in the results, so that its performance may be compared to existing methods. A measure of error would also be required for a thorough study of the α-sensitivity in the single route predicting algorithm.

Appendices

A Draft for Article

A draft for an article to be published is included in the appendix. The article focuses on the pattern recognizing method from the thesis.

Finding Patterns of Motion using Optimal Paths

Mads Fromreide and Alex Hansen

Department of Physics, Norwegian University of Science and Technology
madsfromreide@gmail.com
alex.hansen@ntnu.no

Abstract

The ability to navigate safely and efficiently through a given landscape is relevant for any intelligent moving object. Examples range from robotic science and traffic analysis to the behavior within an ecosystem. In this paper, we propose a method for detecting regular patterns of motion, by modeling the environment as an energy landscape and locate optimal paths through it. A working algorithm is implemented and applied to maritime traffic systems. The results that are found, show that the method have great potential for analyzing and determine regular patterns of motion.

I. Introduction

The understanding of the regular behavior of a dynamic system, gives opportunities to predict future events within it. Objects tend to move in patterns, and are often influenced by other surrounding objects. That is, the motion of a moving object is typically dependent of the system as a whole. Some objects may attract each other, or try to avoid each other depending of their nature. Therefore, predicting the future motion of an object involves predicting the motion of other nearby objects as well. There are several different ways to approach motion prediction. One may predict the motion of every single object in a system by treating them individually [4]. However, for large systems this is an unproductive method. A better approach is to exploit the fact that objects tend to move in patterns. By establishing a model for the regular patterns of motion in the area, one gets the opportunity to predict the motion of a single object. Applications range from behavior of ecosystems, to robotic navigation and large traffic systems [3].

Motion prediction typically operates in two stages. A learning step, which learns the regular patterns of motion, and a prediction step. Such two step processes, may be grouped in two main groups of techniques; Grid-based techniques and cluster-based techniques [3]. The grid based techniques models the environment as a grid, and determines the transition probability between nodes. The grid is then used directly for motion prediction. Clustering based techniques uses statistics, to group similar trajectories in representative clusters. The representative clusters, are then used for motion prediction.

In this paper, we propose a grid-based technique for learning motion patterns by locating optimal paths in a disordered energy landscape. The method is applied to vessel traffic, using AIS coordinates. The coordinates are transformed to a dimensionless area. A grid is introduced over the area and a density associated with the coordinates, is introduced on each grid point. The optimal paths through the area is found by implementing the iterative algorithm introduced by A. Hansen and J. Kertèsz in *Phase Diagram of Optimal Paths* [1].

The algorithm was further generalized by L. Talon et. al. [2] to identify optimal paths on a lattice.

This paper is organized as follows. Section II gives an introduction to the approach of the model, and a description of the algorithms that are used. In section III, the results of our analysis are represented. In section **??** we give a brief discussion of our findings.

II. Methods

Our energy landscape is created from recorded coordinates of moving vessels. The datasets that are used, contain longitude and latitude coordinates on the earths surface. The problem is simplified by turning the area of interest into a square grid in two dimensions.
The earth is being approximated by a sphere with radius r_E. Further the area of interest is approximated by a flat square. If $long_{min}$ and lat_{min} denotes the lower longitude and lower latitude values respectively, the point $(long_{min}, lat_{min})$ is the origin of the introduced coordinate system. Let the distance (in meters) between $long_{max}$ and $long_{min}$ be denoted x_{max}. The dataset is then transformed through the following expression for the longitudinal coordinates

$$x_i = \frac{\pi r_E}{180}\frac{N-1}{x_{max}}(long_i - long_{min}), \qquad (1)$$

and similar for the latitude,

$$y_i = \frac{\pi r_E}{180}\frac{N-1}{y_{max}}(lat_i - lat_{min}). \qquad (2)$$

The area of dimension $N \times N$ is then discretized by introducing a grid with cell size 1×1. A density ρ_{ij} is introduced for each node (i, j) in the grid. A recorded coordinate (x, y) within a cell, contribute to the density of all four corners of the cell. Let r_k $(k = 1, 2, 3, 4)$ be the distance from a given coordinate to the k'th corner of the cell. The addition to the density given by the coordinate to each corner is given by

$$W_k = \frac{1}{3}\frac{R - r_k}{R}, \qquad (3)$$

where $R = \sum_{k=1}^{4} r_k$. The factor $1/3$ is introduced to normalize the weight, $\sum_{k=1}^{4} W_k = 1$. The density, ρ_{ij} is then given by the sum of weights associated with node (i, j).

Further, an energy associated with each grid point is introduced as follows

$$e_{i,j} = (\frac{1}{\rho_{i,j}})^{\alpha}. \qquad (4)$$

The exponent α, that are introduced in Eq. 4, is an important variable as it controls the magnitude of the energy fluctuations in the landscape. If $\alpha < 1$ the fluctuations are smoothed out. When $\alpha = 0$ all points on the grid is assigned the same density. Thus, every optimal path between two points will be equal to the shortest path between them.

The algorithm that are implemented to locate optimal paths requires the energy landscape to consists of nodes with threshold energies between them. A threshold energy is introduced by

$$t_{i,j(i)} = \frac{e_i + e_{j(i)}}{2}, \qquad (5)$$

where the notation $j(i)$ indicates that $j(i)$ is adjacent to i.

An optimal path between two points is defined as the path where the sum over all threshold energies along it, is smallest. The algorithm introduced by A. Hansen and J. Kertèsz locates an optimal path between two points on the boundary of an area [1]. The algorithm is an iterative label-correcting algorithm and it generates a hierarchy of optimal paths between two points or interval of points, which is referred to as a *pathscape*[1, 2].

A variable V_i is assigned to each node $\vec{r}_i$, initially $V_i = 0$ for all i. For the nodes on the boundary of the lattice the values V_i stays fixed, while the ones lying in the interior of the lattice is iteratively updated. The updating process is as follows

$$V_i \to V_i = \min_{j(i)}(t_{i,j(i)} + V_{j(i)}). \quad (6)$$

After N updates, the variable V_i contains the sum of threshold energies along the optimal path of length N originating from $\vec{r}_i$. Consider now a node $\vec{r}_0$ on the boundary of the lattice. In order to find the optimal path from an internal node $\vec{r}_i$ to $\vec{r}_0$, one sets the value $V_{\vec{r}_0}$ to zero, whilst for the remaining boundary nodes the value $V_{\vec{r}_b}$ is set to a large value M. Next, the updating process for the internal nodes are carried out according to Eq. (6), until all values V_i does no longer change. When all of the values have reached a constant, all paths have reached the only node with V_i fixed at zero, namely $\vec{r}_0$. In this case the variable V_i will contain the sum of threshold energies ($T_{i,0}$) along the optimal path between $\vec{r}_i$ and $\vec{r}_0$. If the same process is repeated for some other boundary node $\vec{r}_1$ and interior node $\vec{r}_j$, one gets a new sum, $T_{j,1}$. By the same process one also calculates the values $T_{i,1}$ and $T_{j,0}$. Now the total energy of the optimal path between nodes $\vec{r}_0$ and $\vec{r}_1$ passing through $\vec{r}_i$ may be expressed as

$$T_{0,i,1} = \min_{j(i)}(T_{0,i} + t_{i,j(i)} + T_{j(i),1}, T_{0,j(i)} + t_{j(i),i} + T_{i,1}). \quad (7)$$

To find patterns of motion, optimal paths are found between intervals on the boundaries of a given area. Each edge of the area that are studied is sectioned into n intervals. This gives a total of $6n^2$ optimal paths through the area. By comparing the optimal paths, found between all sets of intervals, one locates the *globally* most optimal paths. Each optimal path is characterized by its total energy (E_p) and its length (L_p). Further, a scaled energy is introduced as follows

$$E'_p = C_p \frac{E_p}{(L_p)^\beta}, \quad (8)$$

where C_p and β are constants which are applied to adjust the scaling. It is this scaled energy, that are compared to find the globally most optimal paths. Since the energy of an optimal path is a sum over the threshold energies along it, shorter paths will in general have lower energy than longer ones. That is, short paths through high energy regions may possibly be preferred over long paths through low energy regions. To avoid this, the scaled energy in Eq. 8 is introduced.

III. Results

The model have been tested on datasets containing AIS coordinates. The results from two datasets are included in the article. The two sets are denoted A and B. Fig. 1 shows dataset A, while Fig. 2 shows dataset B. As the figures indicates, A has a simpler structure than B. A consists of two vertical clusters, while B includes multiple clusters with different orientation.

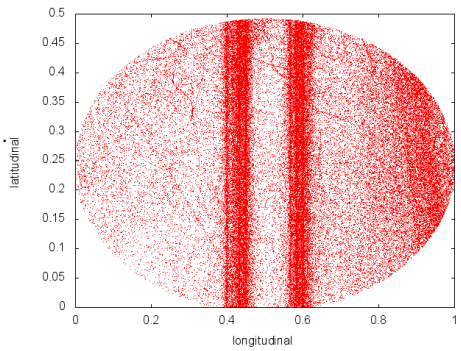

Figure 1: Set A*: Plot of AIS-coordinates in an area for a given time frame. The coordinates are represented relative to the minimum longitudinal and latitudinal values.*

From the two areas, a square section is selected for the analysis. From A an area which spans 0.4° in both directions through the center is selected. The area is selected, so that the two vertical clusters are included. A larger area is selected from B, as it spans 3.0° in both the longitudinal and latitudinal direction.

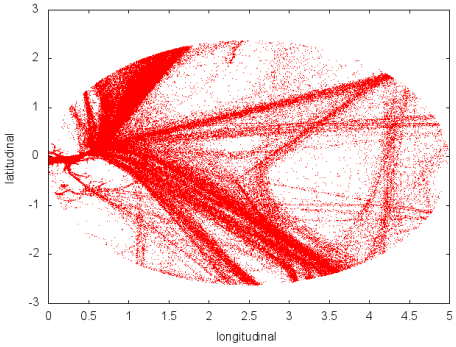

Figure 2: Set B*: Plot of AIS-coordinates from another area for a given time frame. The coordinates are represented as described in Figure 1*

In figure 3, the result is shown for the selected area of A. Here, the two globally most optimal paths are included. The calculation is performed using a grid of dimension 148×148 with cell size 300×300 m^2. Further, the boundaries have been sectioned into two intervals. The variable vales are $\alpha = 0.8$, $\beta = 0.8$ and $C_p = 10$. As the figure shows, these two paths overlaps with the clusters, hence they are representative for the pattern.

The idea of the scaled energy of the paths, is that the model should automatically determine the number of paths that makes out a pattern. That is, with proper scaling, paths that should be included in the pattern should have similar scaled energy. In figure 4 shows the scaled energy (as introduced in Eq. 8) sorted from the lowest to the highest value.

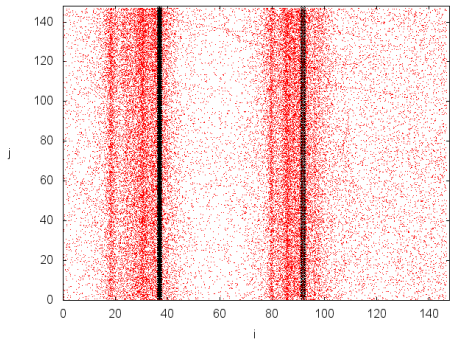

Figure 3: *The result output, showing the pattern found in dataset A. The figure includes the two globally most optimal paths, which represents the regular patterns of motion in the area. The grid has dimension* 148×148 *with cell size of* 300×300 m^2.

The result shown in Fig. 4, shows that a group of five paths stand out from the rest. This indicates, that a total of five paths should be included in the pattern. Figure 5 shows the same result as in Figure 3 with five included paths, instead of two. As the figure shows, the new added paths does not make any contributions to the actual pattern. The new paths overlaps with the two original ones for most of their lengths. However, near the ends, the paths branch out and leaves the direction of the clusters.

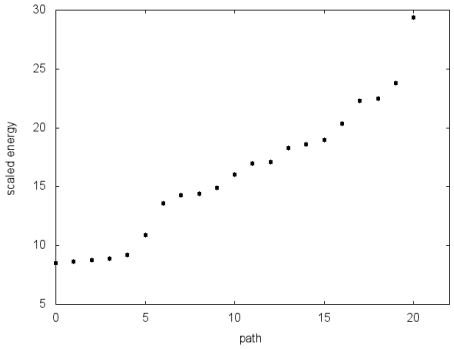

Figure 4: *The scaled energies of optimal paths between selected intervals, ranging from most optimal (0) to least optimal (20). Three paths have been left out from the figure due to very high energy values.*

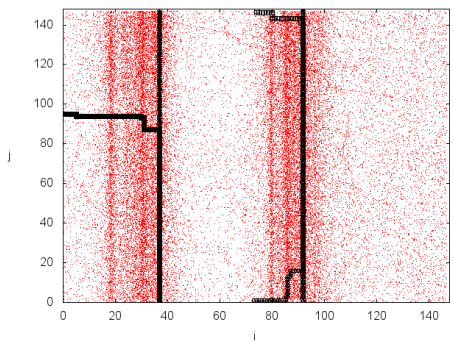

Figure 5: *The located patterns in dataset A. The figure shows the same as Figure 3 with five paths included, instead of two.*

For the more detailed structure in B, the pattern detection is performed with both 2 and 3 intervals per edge. Figure 6 shows the result for the analysis performed with 2 intervals per edge. The associated scaled energy distribution is showed in Fig. 7. The analysis is performed with $\alpha = 0.08$, $\beta = 0.8$ and $C_p = 10$. The grid is of dimension 166×166 and has a cell size of $2000 \times 2000\ m^2$.

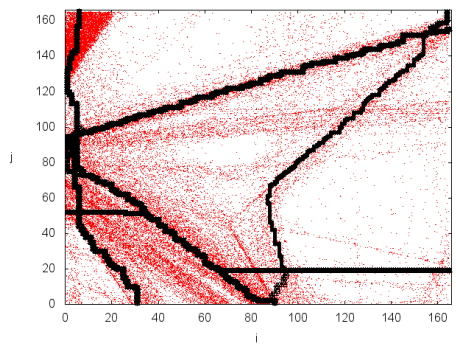

Figure 6: *The pattern in the given section of B, calculated with two intervals per edge. In accordance with Figure 7,* 20 *paths are included in the figure.*

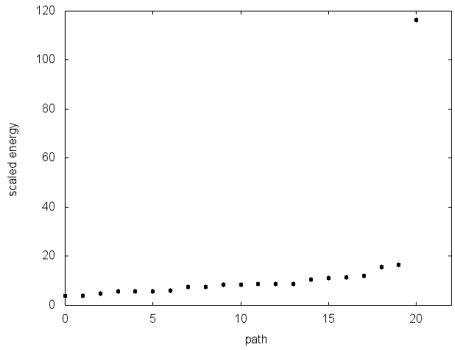

Figure 7: *The scaled energy for the paths showed in Figure 6. The paths are sorted ranging from the lowest energy to the highest.* 3 *of the total* 24 *paths are excluded, due to very high energies.*

Figure 7 indicates that 20 of the 24 created paths should be included in the pattern. Figure 6 includes all these paths. However, several of them overlap for most of their length, giving the impression that there are fewer than 20 paths in the figure. Thus, one see that as for the simpler case in A, the method overestimates the number of paths needed to make out the pattern. Figure 6 shows that the calculations performed with 2 intervals per edge does not capture all the details of the pattern.

For instance a large cluster moving horizontal through the center of the area is not included, neither is the middle of three clusters moving from the left towards the lower boundary. Both of these missing clusters are being dominated by other clusters moving between the same two intervals. Since the method of using optimal paths between intervals, only gives one optimal path per pair of intervals, only the most optimal one is forwarded as a canditate for the pattern. Therefore, both the number of intevals as well as the length and position of boundaries between intervals, has an impact on the results. An increase in intervals per edge, from 2 to 3 results in an increase in possible paths for the pattern from 24 to 54. This, obviously gives a higher chance to detect finer details of the pattern.

The results of the analysis with 3 intervals per edge is shown in Figure 8 with the associated scaled energy distribution in Figure 9. In accordance with the scaled energies (Fig. 9), 48 paths are included to make out the pattern in Figure 8. As some more details are being covered by the extra set of paths, added from Fig. 6, there are also more impurities in the representation of the pattern. As before the model have overestimated the number of paths, which gives overlapping lines which branch out near the ends and intersections.

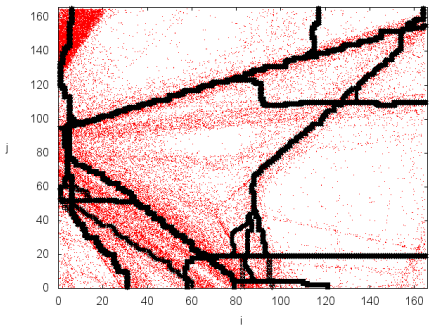

Figure 8: *The pattern found in the section of B with three intervals per edge. As figure 9 suggests, a total of* 48 *paths are included in the figure.*

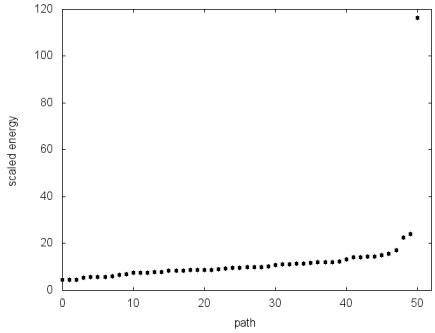

Figure 9: *The scaled energies of the paths showed in Figure 8. The values are sorted from lowest to highest.* 3 *of the total* 54 *values are left out due to very high values.*

IV. Discussion

Through the results in the previous section, the potential of a pattern recognizing method based on combinations of pathscapes is shown. From a visual point of view, the method locates the motion patterns efficently in both the simple case (A) and in the more complex traffic picture (B). However, a proper performance test has not been perfomed. Further, the results showed that sectioning the edges into only a few intervals, were enough to generate a good estimate of the pattern. As all the points along the edges, may be represented by only a few intervals, the running time of the algorithms are kept short. The grid dimension and size of the cells does not seem to influence the results that are found.

The main issue that emerged from the results, is the use of the scaled energy to determine how many paths to include in the pattern, as the model consistently overestimated the number. For both the simple case (A) and the two versions of the more complex system (B), paths that did not contribute to the pattern were included. A possible technique for fixing this problem is filter the paths by comparing them to more optimal ones. If a path is found to overlap with a more optimal one, for

a certain percentage of its length, it could be neglected.

V. Conclusion

In this paper, a method for finding patterns of motion by combining optimal paths through an area is proposed. A method for transforming a set of coordinates to an energy landscape is suggested. An algorithm for finding optimal paths in a disordered energy landscape, is applied to find optimal paths between intervals on the boundaries of a given area. Optimal paths are created between all pairs of intervals, producing candidates for representative trajectories of the pattern. By comparing all paths by their energy scaled by their length, a selection of the globally most optimal paths are found. This group of paths, are the representative trajectories for the clusters of the pattern.

This method is tested on real AIS data. Two different datasets, of different size and complexity are studied. In both cases, the results give evidence for the potential of the model.

VI. References

[1] A. Hansen and J. Kertèsz. Phase diagram of optimal paths. *Physical Review Letters*, 93(4):040601–1, 2004.

[2] L. Talon, H. Auradou, M. Pessel, and A. Hansen. Geometry of optimal path hierarchies. *EPL*, 103(3), 2013.

[3] Vasquez, D. and Fraichard, T. Motion prediction for moving objects: A statistical approach. *Proceedings - IEEE International Conference on Robotics and Automation*, 2004(4):3931–3936, 2004.

[4] Qiuming Zhu. A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion. In *Systems Engineering, 1990., IEEE International Conference on*, pages 216–219, Aug 1990.

B Source code

The source code for finding patterns of motion, within a dataset of longitudinal and latitudinal coordintes are included in the appendix. The full program consists of three files. Section B.1 represents the specialized version of the general pathscape algorithm found in *Geometry of Optimal Path Hierarchies* by L. Talon et. al. [14]. The algorithms used to predict single routes and study the effect of α, are left out of the printed version of the thesis, in order to minimize its length. However, they follow the same basic principle as the functions that are shown. All source code, that are used to create the results in this thesis are included in the digital enclosures.

B.1 pathscape algorithm

```
/*
**  This version of pathscapeorg.c
**  is ment to be used for detecing patterns, it is called by grid.c
**
**    The function patchsape() takes a number of inputs
**        - Result directory
**        - Grid size N ( NxN)
**        - The parameter alpha
**
**    The functions fillGrid() and update()
**    are called by pathscape().
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
void fillGrid();
void update();
FILE *grid;
int N;
int i, j, k, sum, sumgoal;
double infty = 100.0;          // "Infinity"
double inftybound = 10000.0;   // "Larger Infinity" for the boundary
double eps = 0.00000001;       // "no change" < eps
double pot;
bool condition;
char dir[150];
double **dens;
// V
double ***v;
double ***tv;
```

```
void pathscape(char directory[],int sx0[],int sy0[],int sx1[],
               int sy1[],int alength,int blength,int n,double po) {
    sprintf(dir, directory);
    N = n;
    pot = po;
    double dnew, dold = 0.0;
    /*
    ** Opening files
    */
    //Names
    char gridn[200], in1n[200], in2n[200];
    char out1n[200], out2n[200], gpl1n[200], gpl2n[200];
    sprintf(gridn, "%s%s", dir, "\\gridvalues.dat");
    sprintf(out1n, "%s%s", dir, "\\vij.dat");
    sprintf(out2n, "%s%s", dir, "\\vij_modified.dat");
    sprintf(gpl1n, "%s%s", dir, "\\plot.txt");
    sprintf(gpl2n, "%s%s", dir, "\\plot_no.txt");
    // Open
    grid = fopen(gridn,"r");
    FILE *out1 = fopen(out1n, "w");
    FILE *out2 = fopen(out2n, "w");
    FILE *gpl1 = fopen(gpl1n,"w");
    FILE *gpl2 = fopen(gpl2n,"w");
    // Allocate arrays
    dens = (double**) malloc(sizeof(double*)*(N+1));
    for (i = 0; i < N+1; i++) {
        dens[i] = (double*) malloc(sizeof(double)*(N+1));
    }
    v = (double***) malloc(sizeof(double**)*2);
    tv = (double***) malloc(sizeof(double**)*2);
    for (i = 0; i < 2; i++) {
        v[i] = (double**) malloc(sizeof(double*)*(N+1));
        tv[i] = (double**) malloc(sizeof(double*)*(N+1));
    }
    for (i = 0; i < 2; i++) {
        for (j = 0; j < N+1; j++) {
            v[i][j] = (double*) malloc(sizeof(double)*(N+1));
            tv[i][j] = (double*) malloc(sizeof(double)*(N+1));
        }
    }
    // Fill the grid with values from file
    fillGrid();
    fclose(grid);
    /*
    *   Set the endpoint values, given from grid().
    */
    for (i = 0; i < alength; i++) {
            v[0][sx0[i]][sy0[i]] = 0.0;
```

```
        tv[0][sx0[i]][sy0[i]] = 0.0;
}
for (i = 0; i < blength; i++) {
        v[1][sx1[i]][sy1[i]]= 0.0;
        tv[1][sx1[i]][sy1[i]] = 0.0;
}
condition = false;
sumgoal = 2*(N-1)*(N-1);
int itcount = 0;
// Main loop - updates the pathscape by calling update().
while   (condition != true) {
    dnew = 0.0;
    itcount++;
    update();
    for (i = 1; i < N; i++) {
        for (j = 1; j < N; j++) {
            dnew = dnew + v[0][i][j] + v[1][i][j];
        }
    }
    if  (fabs(dnew - dold) <= eps)
        condition = true;
    dold = dnew;
if(itcount % 100 == 0)
printf("Iterations: %i  Differance: %lf\n", itcount, dnew - dold);
}
for (i = 0; i < N+1; i++) {
    for (j = 0; j < N+1; j++) {
        fprintf(out1,"%i %i %lf\n", i, j, v[0][i][j] + v[1][i][j]);
        fprintf(out2,"%lf ", v[0][i][j] + v[1][i][j]);
    }
    fprintf(out2, "\n");
}
fclose(out1);
fclose(out2);
/*
**  Create plotscript and plot
*/
fprintf(gpl1,"cd '%s'\n",dir);
fprintf(gpl1,"set term pdf\n");
fprintf(gpl1,"set out 'pathscape.pdf'\n");
fprintf(gpl1,"set view map\n");
fprintf(gpl1,"set palette gray\n");
fprintf(gpl1,"splot 'vij.dat' using 1:2:3 with image\n");
fprintf(gpl1,"exit\n");
fclose(gpl1);
char plot[200];
sprintf(plot,"%s %s","gnuplot",gpl1n);
system(plot);
```

```
    fprintf(gpl2,"cd '%s'\n",dir);
    fprintf(gpl2,"set term pdf\n");
    fprintf(gpl2,"set out 'pathscape_modified.pdf'\n");
    fprintf(gpl2,"set view map\n");
    fprintf(gpl2,"set palette gray\n");
    fprintf(gpl2,"splot 'vij_modified.dat' using 1:2:3 with image\n");
    fprintf(gpl2,"exit\n");
    fclose(gpl2);
    sprintf(plot,"%s %s", "gnuplot", gpl2n);
    system(plot);
}
void fillGrid(){
    double read;
    int toss;
    // Read in the number of coordinates
    // corresponding to each grid point
    // and inverse the value.
    for (i = 0; i < N+1; i++) {
        for (j = 0; j < N+1; j++) {
            fscanf(grid ,"%i %i %lf\n", &toss, &toss, &read);
            if(read >= 0.00000001)
                dens[i][j] = pow(1.0/read, pot);
            else
                dens[i][j] = pow(infty,pot);
        }
    }
    for (i = 0; i < 2; i++) {
        for (j = 0; j < N+1; j++) {
            for (k = 0; k < N+1; k++) {
                v[i][j][k] = 0.0;
                if  (j == 0 || j == N || k == 0  || k == N )
                    v[i][j][k] = inftybound;
                tv[i][j][k] = v[i][j][k];
            }
        }
    }
}
void update() {
    double minv, up1, up2, up3, up4, up5, up6, up7, up8;
    for (i = 0; i < 2; i++) {
        for (j = 1; j < N; j++) {
            for (k = 1; k < N; k++) {
                tv[i][j][k] = v[i][j][k];
            }
        }
    }
    for (i = 0; i < 2; i++) {
    // Only update nodes in interior, hence j {1,N}, not {0, N+1}
```

```
            for (j = 1; j < N; j++) {
                for (k = 1; k < N; k++) {
                up1 = (dens[j][k+1] + dens[j][k])/2.0 + tv[i][j][k+1];
                up2 = (dens[j][k-1] + dens[j][k])/2.0 + tv[i][j][k-1];
                up3 = (dens[j+1][k] + dens[j][k])/2.0 + tv[i][j+1][k];
                up4 = (dens[j-1][k] + dens[j][k])/2.0 + tv[i][j-1][k];
                minv = up1;
                if(up2 <= minv)
                    minv = up2;
                if(up3 <= minv)
                    minv = up3;
                if(up4 <= minv)
                    minv = up4;
                v[i][j][k] = minv;
              }
            }
        }
    }
```

B.2 main file

```
/*
**  grid.c
**      - Works as the main file for the project
**      - Folder for results must be specifed in line 28
**      - The dataset contating coordiantes must be
**        translated to a suitable form
**      - The prefered size of area to be analyzed
**         must be specified in lines 71 - 74
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pathscapeorg.c"
#include "replot.c"
#include "transback.c"
#include "calclength.c"
void bubblesorting(double arr[], int nElements);
double *x, *y, *xt, *yt, *sorted;
double **grid2d, **pscape;
int main () {
    // Directory of results
    char dir[150] = "specify\the\result\folder\here";
    /*
    ** Parameters
    **  N           :  Dimension of grid N*N
    **  length      :  number of datapoints
    **  R           :  Earths radius [m]
    **  pi          :  pi
    **  infty       :  Value of "infinity"
    **  alpha       :  Parameter for adjusting the energylandscape
    **  scalepow    :  Scaling parameter for energies
    **  scaleconst  :  Scale constant, to enhance differences in energy
    **  alenght     :  Interval length on boundary,
    **  times       :  The number of paths included in the pattern
    **
    */
    int N; // User specified
    int length = 0;
    int R = 6371000;
    double pi = 3.14159265359;
    double infty = 10000.0;
    double alpha = 0.8;
    double scalepow = 0.8;
    double scaleconst = 1.0/10.0;
    int alength = 10;
    int times = 1;
    // Other Variables
    int i, j, k, l, m, xint, yint, change;
    int newlength = length, s, drop, nside;
    int ni, nj, nP, startx, starty, xs, ys;
```

```
double xi, yi, xmax, ymax, xmin, ymin;
double degrad, avspeed;
double longmin = 10000, longmax = 0;
double latmin = 10000, latmax = 0;
double startlong, startlat, endlong, endlat;
double longdif, latdif, avlong, avlat;
double dimension, gridsize, pathlength;
double w1, w2, w3, w4, r1, r2, r3, r4, rtot;
double dropd;
char name[200], gnu[200];
char choise = 'n';
// SET THE BOUNDARY VALUES
// Specify the area of interest by its
// bounday values given in geodetic coordinates
startlong = 0.0;
endlong = 0.0;
startlat = 0.0;
endlat = 0.0;
/*
**  Creating Data Files for results and information
*/
char datan[200], out1n[200], out2n[200], out3n[200];
char minin[200], nPpath[200];
char gpl1n[200], gpl2n[200], gpl3n[200];
// INPUT FILE
sprintf(datan, "%s%s",dir,"\\coords.dat");
// OUTPUT ADN PLOTSCRIPTS
sprintf(out1n, "%s%s",dir,"\\gridvalues.dat");
sprintf(out2n, "%s%s",dir,"\\coords_reduced.dat");
sprintf(out3n, "%s%s",dir,"\\coord_trans.dat");
sprintf(minin, "%s%s",dir,"\\minimumvalues100.dat");
sprintf(nPpath,"%s%s",dir,"\\number_val.dat");
sprintf(gpl1n, "%s%s",dir,"\\plot_full.txt");
sprintf(gpl2n, "%s%s",dir,"\\plot_part.txt");
sprintf(gpl3n, "%s%s",dir,"\\plot_optimal_paths.txt");
FILE *data = fopen(datan, "r"); // input coordinates
FILE *out1 = fopen(out1n, "w"); // Gridvalues
FILE *out2 = fopen(out2n, "w"); // reduced coordinate-set
FILE *out3 = fopen(out3n, "w"); // Transformed coordinates
FILE *gpl1 = fopen(gpl1n,"w");  // Gnuplot script
FILE *gpl2 = fopen(gpl2n,"w");  // Gnuplot script
FILE *gpl3 = fopen(gpl3n,"w");  // Gnuplot script
/*
** Allocation
*/
x = (double*) malloc(sizeof(double)*length);
y = (double*) malloc(sizeof(double)*length);
/*
** Reading - The reading of coordinates, depends of form of file
*/
```

```
for (i = 0; i < length; i++) {
    fscanf(data, "%lf %lf\n", &xi, &yi);
    x[i] = xi;
    y[i] = yi;
    if (xi >= longmax)
        longmax = xi;
    if (xi <= longmin)
        longmin = xi;
    if (yi >= latmax)
        latmax = yi;
    if (yi <= latmin)
        latmin = yi;
}
fclose(data);
/*
**  Create plotscripts
*/
// For the original dataset
fprintf(gpl1, "cd '%s'\n",dir);
fprintf(gpl1, "set term png enhanced\n");
fprintf(gpl1, "unset key\n");
fprintf(gpl1, "set out 'original_data.png'\n");
fprintf(gpl1, "plot 'coords_routes_sorted.dat' using 1:2\n");
fprintf(gpl1, "exit");
fclose(gpl1);
//For the reduced dataset
fprintf(gpl2, "cd '%s'\n",dir);
fprintf(gpl2, "set term png enhanced\n");
fprintf(gpl2, "unset key\n");
fprintf(gpl2, "set out 'partial_data.png'\n");
fprintf(gpl2, "plot 'coord_trans.dat' using 1:2\n");
fprintf(gpl2, "exit");
fclose(gpl2);
//Plot the original data:
sprintf(gnu,"%s %s","gnuplot", gpl1n);
system(gnu);
longdif = fabs(endlong - startlong);
latdif = fabs(endlat - startlat);
if  (longdif > latdif)  {
    avlat = 0.5*(endlat + startlat);
    longmin = fmin(endlong, startlong);
    longmax = fmax(endlong, startlong);
    latmin =  avlat - 0.5*longdif;
    latmax =  avlat + 0.5*longdif;
}
else if (longdif < latdif)   {
    avlong = 0.5*(endlong + startlong);
    latmin = fmin(endlat, startlat);
    latmax = fmax(endlat, startlat);
```

```
    longmin = avlong - 0.5*latdif;
    longmax = avlong + 0.5*latdif;
}
else    {
    longmin = fmin(endlong, startlong);
    longmax = fmax(endlong, startlong);
    latmin = fmin(endlat, startlat);
    latmax = fmax(endlat, startlat);
}
// Count the number of datapoints in the desired area
// and make new arrays of the new length
for(i = 0; i < length; i++) {
    if(x[i]>longmax || x[i]<longmin || y[i]>latmax || y[i]<latmin) {
        newlength--;
        x[i] = 1000.0;
    }
}
xt = (double*) malloc(sizeof(double)*newlength);
yt = (double*) malloc(sizeof(double)*newlength);
// Set the length of x and y to the new length and put back points
s = 0;
for(i = 0; i < length; i++) {
    if(x[i] < 1000.0){
        xt[s] = x[i];
        yt[s] = y[i];
        s++;
    }
}
free(x);
free(y);
length = newlength;
x = (double*) malloc(sizeof(double)*length);
y = (double*) malloc(sizeof(double)*length);
for (i = 0; i < length; i++) {
    x[i] = xt[i];
    y[i] = yt[i];
}
free(xt);
free(yt);
/*
**  Print out the (long,lat) - coordinates in the limited area
*/
for (i = 0; i < length; i++) {
    fprintf(out2, "%lf %lf\n", x[i], y[i]);
}
fclose(out2);
/*
**  - Transform the maximum points from [deg] to [m]
```

```
**    to get the span of the area
**  - Ask for grid size in meters
**  - Calculate the number of gridpoints
*/
degrad = 1.0/180.0;
xmax = pi*R*degrad*(longmax - longmin);
ymax = pi*R*degrad*(latmax - latmin);
while (choise != 'y')   {
printf("\n");
printf("Enter the prefered distance (in [m]) between grid points:");
scanf("%lf", &gridsize);
printf("\n");
N = ((int) (ymax/gridsize));
printf("The given distance gives a %i x %i grid\n", N, N);
printf("Press 'y' to continue, or press 'n' to change the distance:");
scanf(" %c", &choise);
printf("\n");
}
// Create plotscripts that need the value of N
// For plotting the optimal paths
fprintf(gpl3, "cd '%s'\n", dir );
fprintf(gpl3, "set term png enhanced\n");
fprintf(gpl3, "unset key\n");
fprintf(gpl3, "set xrange [0:%i]\n", N);
fprintf(gpl3, "set yrange [0:%i]\n", N);
fprintf(gpl3, "set xlabel \"i\"\n");
fprintf(gpl3, "set ylabel \"j\" rotate by 0\n");
fprintf(gpl3, "set out 'multiple_paths.png'\n");
fprintf(gpl3, "plot 'coord_trans.dat' using 1:2 with dots, ");
for(i = 0; i < times; i++) {
  fprintf(gpl3,"'minimumvalues%i.dat' using 1:2 lc rgb 'black'", i);
  if(i < times - 1)
     fprintf(gpl3, ", ");
}
fprintf(gpl3, "\n");
fprintf(gpl3, "exit");
fclose(gpl3);
/*
**  Allocate memory for arrays, which size is given by N
*/
grid2d = (double**) malloc(sizeof(double*)*(N+1));
pscape = (double**) malloc(sizeof(double*)*(N+1));
for (i = 0; i < N+1; i++) {
    grid2d[i] = (double*) malloc(sizeof(double)*(N+1));
    pscape[i] = (double*) malloc(sizeof(double)*(N+1));
}
// Initializing grid
for (i = 0; i < N+1; i++) {
```

```
    for (j = 0; j < N+1; j++) {
        grid2d[i][j] = 0.0;
        pscape[i][j] = 0.0;
    }
}
/*
** Transformation
*/
for (i = 0; i < length; i++) {
    x[i] = (N-1)*(( pi*R*degrad*(x[i] - longmin) )/xmax);
    y[i] = (N-1)*(( pi*R*degrad*(y[i] - latmin) )/ymax);
}
// Count the number of objects within each grid cell
// and write the result to file
for(i = 0; i < length; i++) {
    // Find the lower left corner
    xint = (int)(x[i]);
    yint = (int)(y[i]);
    // Find the distances to each corner of the surrounding square
    r1 = sqrtf((x[i]-xint)*(x[i]-xint) +
               (y[i]-yint)*(y[i]-yint));
    r2 = sqrtf((x[i]-xint)*(x[i]-xint) +
               (y[i]-(yint+1))*(y[i]-(yint+1)));
    r3 = sqrtf((x[i]-(xint+1))*(x[i]-(xint+1)) +
              (y[i]-(yint+1))*(y[i]-(yint+1)));
    r4 = sqrtf((x[i]-(xint+1))*(x[i]-(xint+1)) +
              (y[i]-yint)*(y[i]-yint));
    rtot = r1 + r2 + r3 + r4;
    w1 = (rtot - r1)/(3*rtot);
    w2 = (rtot - r2)/(3*rtot);
    w3 = (rtot - r3)/(3*rtot);
    w4 = (rtot - r4)/(3*rtot);
    grid2d[xint][yint] += w1;
    grid2d[xint][yint+1] += w2;
    grid2d[xint+1][yint+1] += w3;
    grid2d[xint+1][yint] += w4;
     fprintf(out3,"%lf %lf \n", x[i], y[i]);
}
for (i = 0; i < N+1; i++) {
    for (j = 0; j < N+1; j++) {
        fprintf(out1,"%i %i %.15f\n", i, j, grid2d[i][j]);
    }
}
// Plot the coordinates within the area of interest
sprintf(gnu, "%s %s", "gnuplot", gpl2n);
system(gnu);
```

```
free(x);
free(y);
fclose(out1);
fclose(out3);
// Parameters
int tempalength = alength, blength = alength, an, nnside;
an = N/alength + 1;
int lastlength = N - (an-1)*alength;
int ans6 = 6*an*an;
int sx0[alength], sy0[alength], sx1[alength], sy1[alength];
char totaln[200], totaloutn[200], totalalln[200];
char newoutn[200], numvals[200], tralan[200];
sprintf(totaln, "%s%s", dir, "\\vij.dat");
sprintf(numvals, "%s%s", dir, "\\number_val.dat");
sprintf(newoutn, "%s%s", dir, "\\sorted_minpaths.dat");
sprintf(totalalln, "%s%s", dir, "\\total_all.dat");
FILE *ut2 = fopen(totalalln, "w");
// loop over all pairs of axes
for(nside = 0; nside < 6; nside++) {
    sprintf(totaloutn, "%s%s%i%s", dir, "\\total", nside, ".dat");
    FILE *ut = fopen(totaloutn, "w");
    // loop over all intervals startpoints  < an
    for(j = 0; j < an; j++) {
        // loop over all intervals endpoints  < an
        for(s = 0; s < an; s++){
            if(j == an-1)
                alength = lastlength;
            else
                alength = tempalength;
            if(s == an-1)
                blength = lastlength;
            else
                blength = tempalength;
            for(i = 0; i < alength; i++) {
                    if(nside == 0) {
                        sx0[i] = i + j*tempalength;
                        sy0[i] = 0;
                    }
                    if(nside == 1) {
                       sx0[i] = 0;
                       sy0[i] = i + j*tempalength;
                    }
                    if(nside == 2) {
                        sx0[i] = i + j*tempalength;
                        sy0[i] = 0;
                    }
                    if(nside == 3) {
                       sx0[i] = 0;
                       sy0[i] = i + j*tempalength;
```

```
                }
                if(nside == 4) {
                    sx0[i] = i + j*tempalength;
                    sy0[i] = N;
                }
                if(nside == 5) {
                   sx0[i] = N;
                   sy0[i] = i + j*tempalength;
                }
        }
        for(i = 0; i < blength; i++) {
                if(nside == 0) {
                    sx1[i] = i + s*tempalength;
                    sy1[i] = N;
                }
                if(nside == 1)  {
                    sx1[i] = N;
                    sy1[i] = i + s*tempalength;
                }
                if(nside == 2) {
                    sx1[i] = 0;
                    sy1[i] = i + s*tempalength;
                }
                if(nside == 3)  {
                    sx1[i] = i + s*tempalength;
                    sy1[i] = N;
                }
                if(nside == 4) {
                    sx1[i] = N;
                    sy1[i] = i + s*tempalength;
                }
                if(nside == 5)  {
                    sx1[i] = i + s*tempalength;
                    sy1[i] = 0;
                }
        }
        printf("nside: %i j: %i s: %i\n", nside, j, s);
// Generate a pathscape
pathscape(dir, sx0, sy0, sx1, sy1, alength, blength, N, alpha);
// Run repl() to get the minimumpath in the pathscape
repl(dir, N, 100);
// Open the file "minimumvalues.dat"
// to get the energy of the minimumpath
FILE *minpathFile = fopen(minin, "r");
fscanf(minpathFile, "%i %i %lf\n", &drop, &drop, &xmin);
fclose(minpathFile);
// Run calc_pathlength() to get the minimumspath length
// For now, the function only retuns the number of points
```

```
    // on the minimumpath - 1
    pathlength = calc_pathlength(dir, N, 100);
    printf("pathlength returned : %lf \n", pathlength);
    if(pathlength == 0.0001)
        xmin = 100;
    fprintf(ut, "%i %i %i %lf %lf\n", nside, j, s, xmin, pathlength);
    fprintf(ut2, "%i %i %i %lf %lf\n", nside, j, s, xmin, pathlength);
    }
  }
    fclose(ut);
}
fclose(ut2);
FILE *ina = fopen(totalalln, "r");
double unsorted[ans6], tunsorted[ans6];
int first[ans6], second[ans6], tside[ans6];
sorted = (double*) malloc(sizeof(double)*(ans6));
for(i = 0; i < ans6; i++) {
    fscanf(ina,"%i %i %i %lf %lf\n",&nnside,&ni,&nj,&xi,&pathlength);
    unsorted[i] = xi/(scaleconst*pow(pathlength, scalepow));
    tunsorted[i] = xi/(scaleconst*pow(pathlength, scalepow));
    first[i] = ni;
    second[i] = nj;
    tside[i] = nnside;
    }
    fclose(ina);
    bubblesorting(unsorted, ans6);
    printf("sorted: \n");
    for(i = 0; i < ans6; i++) {
        printf("%lf\n", sorted[i]);
    }
    FILE *newout = fopen(newoutn,"w");
    for(i = 0; i < ans6; i++) {
        for(j = 0; j < ans6; j++) {
            if(tunsorted[i] == sorted[j])
                s = j;
        }
    fprintf(newout, "%i %i %i %.15f %i\n",tside[i],first[i],
            second[i],tunsorted[i],s);
    }
    fclose(newout);
    int wpath, npaths;
    /*
    **  The actual pathscapes are not stored,
    **  only their optimal path information.
    **  The pathscapes chosen for the pattern
    **  must therefore be run again.
    */
```

```
    for(npaths = 0; npaths < times; npaths++) {
        FILE *inagain = fopen(newoutn, "r");
        for(i = 0; i < ans6; i++) {
fscanf(inagain,"%i %i %i %lf %i\n",&nnside,&ni,&nj,&xi,&wpath);
            if(wpath == npaths){
                j = ni;
                s = nj;
                nside = nnside;
            }
        }
            if(j == an - 1)
                alength = lastlength;
            else
                alength = tempalength;
            if(s == an - 1)
                blength = lastlength;
            else
                blength = tempalength;
            for(i = 0; i < alength; i++) {
                if(nside == 0) {
                    sx0[i] = i + j*tempalength;
                    sy0[i] = 0;
                }
                if(nside == 1)  {
                    sx0[i] = 0;
                    sy0[i] = i + j*tempalength;
                }
                if(nside == 2) {
                    sx0[i] = i + j*tempalength;
                    sy0[i] = 0;
                }
                if(nside == 3) {
                    sx0[i] = 0;
                    sy0[i] = i + j*tempalength;
                }
                if(nside == 4) {
                    sx0[i] = i + j*tempalength;
                    sy0[i] = N;
                }
                if(nside == 5) {
                    sx0[i] = N;
                    sy0[i] = i + j*tempalength;
                }
            }
            for(i = 0; i < blength; i++) {
                if(nside == 0) {
                    sx1[i] = i + s*tempalength;
                    sy1[i] = N;
                }
                if(nside == 1)  {
                    sx1[i] = N;
```

```
                        sy1[i] = i +s*tempalength;
                    }
                    if(nside == 2) {
                        sx1[i] = 0;
                        sy1[i] = i + s*tempalength;
                    }
                    if(nside == 3)  {
                        sx1[i] = i + s*tempalength;
                        sy1[i] = N;
                    }
                    if(nside == 4) {
                        sx1[i] = N;
                        sy1[i] = i + s*tempalength;
                    }
                    if(nside == 5)  {
                        sx1[i] = i + s*tempalength;
                        sy1[i] = 0;
                    }
                }
        pathscape(dir, sx0, sy0, sx1, sy1, alength, blength, N, alpha);
        repl(dir, N, npaths);
        fclose(inagain);
      }
        sprintf(gnu,"%s %s", "gnuplot", gpl3n);
        system(gnu);
    return 0;
}
/*
**  Simple implementation of the bubble sorting algorithm,
**  used to sort optimal paths by energy
*/
void bubblesorting(double arr[], int nElements) {
    double temp;
    int i, j;
    for (i = 0; i < nElements - 1; i++) {
        for (j = 0; j < nElements - 1 - i; j++) {
            if(arr[j] >= arr[j+1]) {
                temp = arr[j+1];
                arr[j+1] = arr[j];
                arr[j] = temp;
            }
        }
    }
    for (i = 0; i < nElements; i++) {
        sorted[i] = arr[i];
    }
}
```

B.3 Other functions

```
/*
**  replot.c
**      - called from grid.c
**      - used to localize minimupath in pathscapae
**      - originally used to replot data,
**      - plots the minimumpath
*/
#include <stdio.h>
#include <stdlib.h>
double **dens;
void repl(char directory[], int N, int npaths) {
    //Parameters
    double infty = 10000.0;      // "infinity"
    double perc = 0.000000001;  // "zero"
    //Directory
    char dir[150];
    sprintf(dir, directory);
    // Help parameters
    int i, j, toss;
    double read, minval  = 5*infty;
    /*
    ** Creating and opening datafiles and gnuplot script
    */
    char inn[200], inNn[200];
    char out1n[200], out2n[200], out3n[200];
    char gpln[200], plot[200];
    sprintf(inn, "%s%s", dir, "\\vij.dat");
    sprintf(out1n, "%s%s", dir, "\\part_vij.dat");
    sprintf(out2n, "%s%s%i%s", dir, "\\minimumvalues",npaths,".dat");
    sprintf(out3n, "%s%s", dir, "\\number_val.dat");
    sprintf(gpln, "%s%s", dir, "\\plotpaths.txt");
    sprintf(plot, "%s %s", "gnuplot", gpln);
    FILE *in = fopen(inn,"r");
    FILE *out1 = fopen(out1n,"w");
    FILE *out2 = fopen(out2n,"w");
    FILE *out3 = fopen(out3n,"w");
    FILE *gpl = fopen(gpln,"w");
    // Plot script
    fprintf(gpl, "cd '%s'\n", dir);
    fprintf(gpl, "set term pdf\n");
    fprintf(gpl, "set out 'optimalpath.pdf'\n");
    fprintf(gpl, "set view map\n");
    fprintf(gpl, "set palette gray\n");
    fprintf(gpl, "set xrange [0:%i]\n", N+1);
    fprintf(gpl, "set yrange [0:%i]\n", N+1);
    fprintf(gpl, "splot 'part_vij.dat' using 1:2:3 with image\n");
```

```
fprintf(gpl, "exit\n");
fclose(gpl);
dens = (double**) malloc(sizeof(double*)*N+1);
for (i = 0; i < N+1; i++) {
    dens[i] = (double*) malloc(sizeof(double)*N+1);
}
// Read in data and find minimum
for (i = 0; i < N+1; i++) {
    for (j = 0; j < N+1; j++) {
        fscanf(in, "%i %i %lf\n", &toss, &toss, &read);
        dens[i][j] = read;
        if  (dens[i][j] < minval)
            minval = dens[i][j];
    }
}
double nextminval = 5*infty;
for (i = 0; i < N+1; i++) {
    for (j = 0; j < N+1; j++) {
        if(dens[i][j] < nextminval && dens[i][j] > minval)
            nextminval = dens[i][j];
    }
}
// Print new data
for (i = 0; i < N+1; i++) {
    for (j = 0; j < N+1; j++) {
        if  (dens[i][j] > (1.0 + perc)*minval)
            dens[i][j] = infty;
        fprintf(out1, "%i %i %lf\n", i, j, dens[i][j]);
    }
}
int counter = 0;
for(i = 0; i < N+1; i++){
    for(j = 0; j < N+1; j++) {
        if(dens[i][j] <= minval*(1.0 + perc)){
        fprintf(out2,"%i %i %.15f\n", i, j, dens[i][j]);
        counter++;
        }
    }
}
fprintf(out3,"%i\n", counter);
printf("%i", counter);
free(dens);
fclose(in);
fclose(out1);
fclose(out2);
fclose(out3);
system(plot);
```

```
}
```

7. References

[1] Karl Gunnar Aarsæther. *Modeling and analysis of ship traffic by observation and numerical simulation*. PhD thesis, Norwegian University of Science and Technology, Department of Marine Technology, 2011.

[2] F.A. Aloul, B.A. Rawi, and M. Aboelaze. Routing in optical and non-optical networks using boolean satisfiability. *Journal of Communications*, 2(SPL.ISS. 4):49–56, 2007.

[3] Michael J. Bannister and David Eppstein. Randomized speedup of the bellman-ford algorithm. *CoRR*, abs/1111.5414, 2011.

[4] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. *Proceedings - IEEE International Conference on Robotics and Automation*, 4:3601–3606, 2002.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977.

[6] A. Hansen and J. Kertèsz. Phase diagram of optimal paths. *Physical Review Letters*, 93(4):040601–1, 2004.

[7] A.K. Jain, R.P.W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[8] M. Kardar and Y.-C. Zhang. Scaling of directed polymers in random media. *Physical Review Letters*, 58(20):2087–2090, 1987.

[9] International Maritime Organization. Ais transponders, 2014. URL: `http://www.imo.org/OurWork/Safety/Navigation/Pages/AIS.aspx`.

[10] International Maritime Organization. History of solas, 2014. URL: `http://www.imo.org/KnowledgeCentre/ReferencesAndArchives/HistoryofSOLAS/Pages/default.aspx`.

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

[12] B. Ristic, B. La Scala, M. Morelande, and N. Gordon. Statistical analysis of motion patterns in ais data: Anomaly detection and motion prediction. *Proceedings of the 11th International Conference on Information Fusion, FUSION 2008*, 2008.

[13] Nehemia Schwartz, Alexander L. Nazaryev, and Shlomo Havlin. Optimal path in two and three dimensions. *Phys. Rev. E*, 58:7642–7644, Dec 1998.

[14] L. Talon, H. Auradou, M. Pessel, and A. Hansen. Geometry of optimal path hierarchies. *EPL*, 103(3), 2013.

[15] Vasquez, D. and Fraichard, T. Motion prediction for moving objects: A statistical approach. *Proceedings - IEEE International Conference on Robotics and Automation*, 2004(4):3931–3936, 2004.

[16] S. Watanabe. *Pattern recognition: human and mechanical*. Wiley, 1985.

[17] Qiuming Zhu. A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion. In *Systems Engineering, 1990., IEEE International Conference on*, pages 216–219, Aug 1990.