Bus Inf Syst Eng 62(5):435–450 (2020)
https://doi.org/10.1007/s12599-020-00661-0

RESEARCH PAPER

Extracting Maritime Traffic Networks from AIS Data Using Evolutionary Algorithm

Dominik Filipiak · Krzysztof Węcel · Milena Stróżyna · Michał Michalak · Witold Abramowicz

Received: 5 February 2020 / Accepted: 19 June 2020 / Published online: 28 July 2020

Abstract The presented method reconstructs a network (a graph) from AIS data, which reflects vessel traffic and can be used for route planning. The approach consists of three main steps: maneuvering points detection, waypoints discovery, and edge construction. The maneuvering points detection uses the CUSUM method and reduces the amount of data for further processing. The genetic algorithm with spatial partitioning is used for waypoints discovery. Finally, edges connecting these waypoints form the final maritime traffic network. The approach aims at advancing the practice of maritime voyage planning, which is typically done manually by a ship's navigation officer. The authors demonstrate the results of the implementation using Apache Spark, a popular distributed and parallel computing framework. The method is evaluated by comparing the results with an on-line voyage planning application. The evaluation shows that the approach has the capacity to generate a graph which resembles the real-world maritime traffic network.

Accepted after one revision by Christof Weinhardt.

D. Filipiak (✉) · K. Węcel · M. Stróżyna · W. Abramowicz
Department of Information Systems, Poznań University of Economics and Business, Aleja Niepodległości 10, 61-875 Poznań, Poland
e-mail: dominik.filipiak@ue.poznan.pl

K. Węcel
e-mail: krzysztof.wecel@ue.poznan.pl

M. Stróżyna
e-mail: milena.strozyna@ue.poznan.pl

W. Abramowicz
e-mail: witold.abramowicz@ue.poznan.pl

Keywords Maritime traffic network · Vessel routing · Route planning · AIS · Maritime traffic graph · Waypoint discovery · Graph discovery · Artificial intelligence · Genetic algorithm

1 Introduction

In the maritime domain, a safe and efficient vessel operation requires a prescient berth to berth voyage planning, resulting in a route that consists of waypoints and legs. A waypoint is a single coordinate within a route, at which a vessel stops or changes its course. Despite the existence of a number of supporting bridge systems, such a voyage is normally planned manually by the ship's crew. This task might be supported by additional checking facilities, e.g., warning about unsafe water depths. Such support is especially important in areas with a high traffic density. In addition, navigators who are unfamiliar with a sea area do not necessarily have information about past experience and best practices in the considered area.

This problem can be addressed by an assistance system that supports the navigator in planning a safe and efficient route before the voyage starts, by providing a network of typical traffic routes based on past behavior of other, similar ships that were traveling in a given area. In this paper we show that such a network can be extracted automatically based on past trajectories of ships using various data science methods. Past ships trajectories can be extracted from the Automatic Identification System (AIS) – an automatic tracking system for ships equipped with a transponder that in specified time intervals sends information (messages) about ships' identification, location, course, speed, etc. AIS messages are then collected by terrestrial stations or satellites and are often used for analyzing the maritime traffic.

Our research was directed by the following research questions: How is it possible to automatically and efficiently discover patterns in the maritime routing based on historical AIS data? What kind of data science method might be applied for such a pattern discovery and for creating a network representing the maritime traffic? How can we design and implement a method capable of processing huge amounts of maritime data efficiently? To answer these questions, we propose an analytical process for the discovery of a shipping network that consists of three consecutive steps. The approach is developed based on the Design Science methodology, using evolutionary and graph algorithms as a theoretical foundation. Given historical AIS data, the presented method aims at constructing a network reflecting vessel traffic. This approach is also a response to the lack of methods that discover the critical maritime waypoints in an efficient manner, based on the analysis of big amounts of historical data, thus aiming to advance practice of maritime voyage planning that is typically done manually by a ship's navigation officers (Zhang et al. 2018).

We demonstrate the results of the process implementation using Apache Spark, a popular distributed and parallel computing framework. As a proof-of-concept, the results for data from the Baltic Sea area are presented. These results show that our approach has the capacity to generate the maritime traffic network based on real-world maritime traffic.

2 Methodology

The approach presented in this paper follows the Design Science (DS) methodology by Hevner et al. (2008) because it supports the development of new, innovative artifacts. Such artifacts should provide a contribution to the existing body of knowledge and take the form of constructs, instantiations, models, or methods. In case of our research, we developed a method artifact (consisting of three algorithms), which aims at helping navigators in planning a maritime route by automatic discovery of waypoints and defining optimal routes. We use evolutionary computing and graph theory methods as a theoretical background. We have followed DS guidelines and the iterative research methodology (Hevner and Chatterjee 2010) consisting of six activities: *problem identification and motivation*, *objectives of a solution*, *design and development*, *demonstration*, *evaluation*, and *communication*.

The identification of the research problem and out motivation has been presented in Sect. 1. The definition of the solution objectives and its requirements from theory and practice has been conducted based on a literature review (Sect. 3). The third step (*design and development*) focuses on how to combine the identified practical and theoretical requirements with a systematic design of the artifact. Therefore, we explain the concept and assumptions of the proposed method as well as its main components in Sect. 4. The method consists of three components: the CUSUM algorithm used as a pre-processing step, a parallel genetic algorithm for waypoints discovery, and a graph algorithm for detecting edges between waypoints. Thus, the final results generated by the method can be used for an effective planning of a maritime route.

As soon as the method was developed, we applied it within laboratory experiments, which were conducted based on the real AIS data, to demonstrate its applicability (Sect. 5). Based on the results of the experiments, the method is evaluated. The evaluation focuses on two important aspects, namely, the quality of waypoints and routes discovered and the overall efficiency of the method to process large amount of AIS data. The quality of the results is measured based on a comparison with maritime routes defined using a mixture of quantitative and qualitative analysis of low-level elements of the solution. The efficiency is shown by the processing time of a sample AIS data and the scalability of the solution. Both criteria of evaluation ensure necessary rigor of analysis to prove that the artifact addresses the practical applicability. The steps 3–5 were repeated iteratively, meaning that the results of the evaluation were transferred back to the *design and development* step. Using this multi-step evaluation, we intend to ensure the validity of the results and iteratively improve the developed solution. Finally, we outline our contribution, discuss limitations and indicate future work (Sect. 6).

3 Related Work

Our work focuses on a generation of a maritime traffic network (which essentially is a graph) that can be used later in different scenarios. A number of scholars have carried out empirical studies on naval routing and voyage planning. An optimal route can be defined as the blend of shortest time, minimal fuel consumption, and general safety of navigation (Wang et al. 2018). *Routing* and *path planning* seem to be used interchangeably in the literature. Following Tu et al. (2018), there are some formal differences between them. Path planning in its simplest form can be defined as finding the shortest path between two points, using the great-circle distance or rhumb line and considering the obstacles. Routing can be defined as a prediction of a vessel's next position based on its current position and a number of features, such as speed Tu et al. (2018). Other scholars refer to it as *route design* (Cai et al. 2014) or *navigation planning* (Tan et al. 2018). The term can be narrowed down to some specific meaning, for instance weather routing adds an additional layer of complexity by considering conditions such as wind or sea currents. Other researchers focus on the planning of fuel efficiency (Schøyen and Bråthen 2015).

Tu et al. (2018) distinguished three main classes of approaches to the ship routing problem: methods based on physical models, methods based on learning models, and hybrid methods. Physical models are useful for simulation purposes and one can indicate curvlinear, lateral, and ship models. Learning-based models consist of neural networks, the Gaussian process, the Kalman filtering method, and the Minor Principal Component among others. Hybrid methods are a blend of any two of these.

Among hybrid methods a number of approaches can be enumerated. The isochrone method was proposed by James (1957). Originally, it was not suitable for computers, but the method was extended by Hagiwara and Spaans (1987), as well as by Fang and Lin (2015). The calculus of variations, approaching the issue as a continuous minimum optimization problem, was used by Haltiner et al. (1962). It was later extended by Bijlsma (2001). Wang et al. (2018) point out that this method is not very useful for practical applications. One can also use dynamic programming, which treats the issue as a discrete multi-stage decision problem (Bellman 1952). It was later used by a number of scholars for this problem (De Wit 1990; Calvert et al. 1991; Shao et al. 2012). Wang et al. (2018) argue that this method has a high complexity combined with high accuracy. The Dijkstra algorithm finds the shortest path in a directed graph and it was applied in a number of weather routing research papers (Mannarini et al. 2016; Montes 2005; Panigrahi et al. 2012; Sen and Padhy 2010). However, Sen and Padhy (2010) claim that this approach does not yield a smooth path.

Introduced by Ester et al. (1996), DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) is a popular method for spatial clustering. In DBSCAN, contrary to many unsupervised learning algorithms, the number of desired clusters is not its hyper-parameter (i.e., the number of clusters does not have to be known upfront). DBSCAN also detects and deals with outliers in an automatic way, which is desirable for (usually) noisy data such as AIS. Ester and Wittmann (1998) later extended this approach and prepared an incremental learning version of the DBSCAN algorithm. DBSCAN has been used in many contexts and variations with AIS data, such as for detecting fishing spots Mazzarella et al. (2014), or finding abnormal trajectories in a parallel manner Chen et al. (2017). In another example, Pallotta et al. (2013) presented *Traffic Route Extraction and Anomaly Detection* (TREAD), which is a methodology for an incremental and unsupervised machine learning approach for building the maritime traffic network through waypoints discovery, low-likelihood anomaly detection, and route prediction. The waypoints discovery component relies on the incremental version of DBSCAN. TREAD was later used and extended by Arguedas et al. (2017) as *Maritime Traffic Knowledge Discovery and Representation System*, which aims at traffic network creation. The construction of the network – preceded by waypoints detection, route detection, and route decomposition – relies on the Douglas–Pecker algorithm for breakpoints detection (these will serve as nodes) along with a custom algorithm for creating traffic lanes (edges).

Wang et al. (2018) proposed to use a genetic algorithm in the weather routing research. Indeed, swarm and evolutionary algorithms can solve maritime routing and planning related problems. Different swarm intelligence methods have been used in various scenarios. For example, Kosmas and Vlachos (2012) used simulated annealing, whereas Tsou and Cheng (2013) proposed ant colony optimization for this task. A number of studies have demonstrated the usage of evolutionary algorithms in different configurations, such as multi-objective evolutionary algorithm (Marie and Courteille 2009; Szłapczynska and Smierzchalski 2009; Vettor and Soares 2016), or real-coded genetic algorithm (Maki et al. 2011; Wang et al. 2018). Dobrkovic et al. (2015, (2018) used genetic algorithm paired with spatial partitioning to enhance the process of clustering vessel positions and allow fast computation of increasing amounts of data. Their research is one of the first that focuses not only on proposing a robust and accurate algorithm but also on the speed of the algorithm, enabling it to be used in real-life applications where large data volumes have to be processed quickly. In our research we are guided by similar assumptions, therefore the paper of Dobrkovic et al. (2018) constituted a starting point for our study.

4 Method

In this section we present our approach to the maritime voyage planning problem. The research objective is to obtain a network representing the maritime traffic. More formally, the process can be perceived as a directed graph building, in which its vertices represent waypoints ("maritime crossroads") connected by edges ("maritime roads"). The resulting graph should reflect the real maritime routes, such as local traffic separation schemes. This goal is contrary to some approaches that focus solely on visualization – albeit they can reflect and represent the real traffic accurately, they cannot be used in route planning. A graph-based representation does not have this limitation and can be queried with standard search algorithms – such as Dijsktra's or A* – for maritime route planning purposes. Our approach consists of three main steps. The first one can be considered as a pre-processing step – the CUSUM method substantially reduces the volume of AIS data to process (Sect. 4.1). Then, the genetic algorithm with spatial partitioning is responsible for the identification of nodes in the graph, which represent sea waypoints (Sect. 4.2). The final step is edge detection for the graph (Sect. 4.3). All designed algorithms were implemented in a distributed and parallel processing manner (see notes on the implementation in Sect. 5).

4.1 CUSUM

The CUSUM (cumulative sum) algorithm is a well-known technique, typically used for quality control in production processes Page (1954). The method enables to detect abrupt changes in given observations Faithfull (2017). In our work, CUSUM is used for change detection. It aims at processing the collected AIS data in order to find preliminary waypoints for further analysis. CUSUM analyzes trajectories of ships (sequence of AIS messages sent by a ship in a given voyage) and detects messages that describe a significant change in speed or course. These messages are the preliminary waypoints, from which the final waypoints will be selected (see Sect. 4.2).

Following Basseville and Nikiforov (1993), by the abrupt change we understand a point on the timeline at which properties of a current observation change. Before and after this moment, the properties are constant in some sense. Based on this definition, it is possible to map AIS messages to a data stream. The main objective is to detect significant maneuvers (e.g., sudden change of a course) by sequential analysis of its trajectory. CUSUM has a few implementations, such as one-sided algorithm for observations with the expected direction of the changes Lamm and Hahn (2017), as well as two-sided, which handles increases and decreases of the observed variable. As the maneuvers in the AIS data can be identified primarily by the increase or decrease of the course (or speed), the two-sided algorithm has been taken into consideration. We can assume that AIS messages represent a certain stream of data (Faithfull 2017):

$$\mathbf{y} = [y_1 \quad y_2 \quad \dots \quad y_n]. \tag{1}$$

We first define decision function g_k Basseville and Nikiforov (1993) in the positive form and negative form:

$$g_k^+ = \left(g_{k-1}^+ + y_k - \mu_0 - \frac{\upsilon}{2}\right)^+, \qquad g_k^- = \left(g_{k-1}^- - y_k + \mu_0 - \frac{\upsilon}{2}\right)^+. \tag{2}$$

Basseville and Nikiforov (1993) proposed the following equation for determining of an alarm time:

$$t_a = \min\{k : (g_k^+ \geq h) \cup (g_k^- \geq h)\}, \tag{3}$$

where t_a is a point where the decision function g_k^+ or g_k^- reached the previously defined threshold h.

Three parameters should be provided as an input: μ_0, υ and threshold h. The first one (μ_0) is calculated dynamically and stabilizes the decision function with a moving average value from the last z observations. Lamm and Hahn (2017) provided the result with a range of AIS messages between 3 and 6 observations (value of z). The second parameter, υ, requires the knowledge of the whole trajectory. Lamm and Hahn (2017) suggested using an upper quantile of all $|\Delta y|$, because this measure indicates the structure of a given voyage. The threshold h controls the sensitivity of the algorithm. Depending on the context, we set this parameter between 1 (higher sensitivity) and 4 (lower sensitivity, with the risk of skipping significant maneuvers). The more sensitive the algorithm is, the more change points will be detected.

The above steps and conditions are formalized in Algorithm 1. If the decision function reaches the threshold, the current observation is saved as a waypoint candidate. In order to achieve an optimal efficiency and parallel AIS data processing, we used Apache Spark for the implementation. Our experiments indicated that CUSUM "filters out" around 80–95% AIS messages, depending on algorithms' hyper-parameters and a given set of trajectories. Examples of manoeuvring points detected by CUSUM based on course changes are presented in Fig. 1.

Algorithm 1 CUSUM change detection

```
1: function EXEC-CUSUM($h, n_{sma}, o$)
2:     $g_0^- \leftarrow 0, \quad g_0^+ \leftarrow 0$
3:     for $k \leftarrow 1$ to $observations$.SIZE() do
4:         $\mu_0 \leftarrow$ UPDATEMOVINGAVERAGE($o_k, n_{sma}$)
5:         $g_k^+, g_k^- \leftarrow$ CALCULATE($g_{k-1}^+, g_{k-1}^-, \mu_0, o_k$)
6:         if $g_k^- \leq 0$ then $g_k^- \leftarrow 0$
7:         if $g_k^+ \leq 0$ then $g_k^+ \leftarrow 0$
8:         if THRESHOLDREACHED($g_k^-, g_k^+$) then
9:             SAVEOBSERVATION($o_k$)
10:            $g_{k-1}^- \leftarrow 0, \quad g_{k-1}^+ \leftarrow 0$
11:            $g_k^- \leftarrow 0, \quad g_k^+ \leftarrow 0$
12:        $g_{k-1}^- \leftarrow g_k^-, \quad g_{k-1}^+ \leftarrow g_k^+$
```

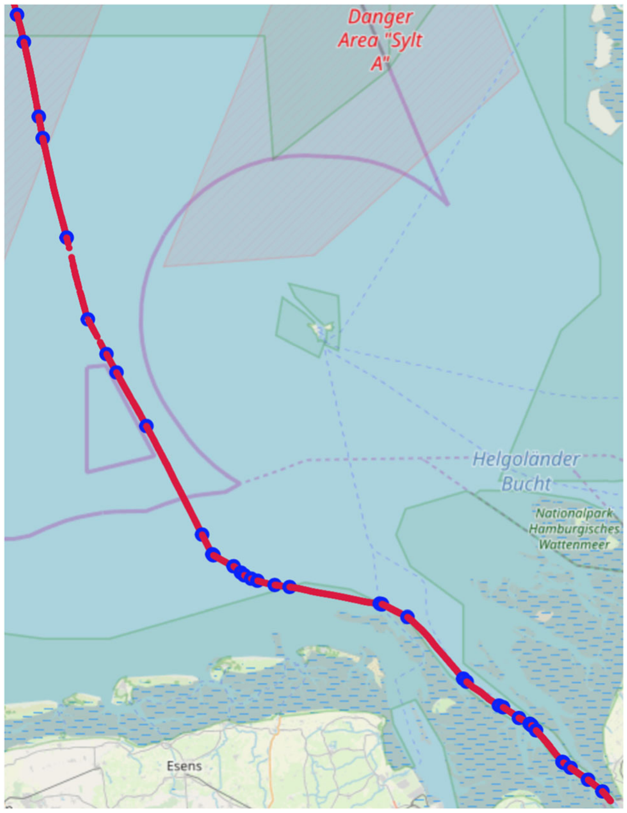


Fig. 1 Visualization of maneuvers detected by CUSUM

4.2 Parallel Genetic Algorithm

The method presented in this section takes as an input results provided by the CUSUM algorithm and consists of two main steps. First, AIS points are partitioned using the spatial partitioning algorithm. Second, the genetic algorithm processes the waypoint candidates, separately for each partition, and detects the final waypoints in a distributed manner. We use geospatial partitioning – each partition is treated separately by the algorithm. Partitions are processed in parallel and the merged sub-results are the final ones. The presented variation of the genetic algorithm is strongly influenced by the one used by Dobrkovic et al. (2015, 2018), though some major differences exist, such as different partitioning, parallelization, and the method of drawing random genes.

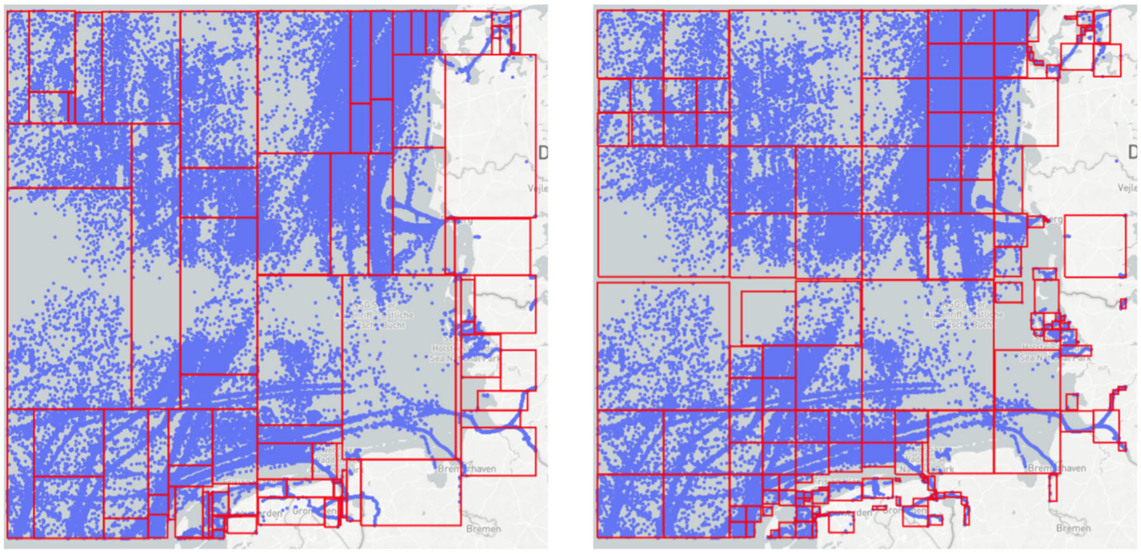
4.2.1 Spatial Partitioning

One of the main challenges in AIS data processing is their uneven spatial distribution, caused both by traffic density and data collection (terrestrial vs. satellite). To mitigate this problem, Dobrkovic et al. (2018) proposed to use QuadTrees. This issue is especially important when one is attempting to use a genetic algorithm for building network, since the densely populated areas would represent the fittest genes and less dense areas would not be inspected at all. Following their suggestion, we have tested two tree-based data structures for spatial partitioning of AIS data: *k*-d B-trees and QuadTrees. The *k*-d B-trees method is a specific juxtaposition of *k*-d trees and B-trees Robinson (1981). Similarly to *k*-d trees, a binary tree with nodes storing *k*-dimensional points is built – the *longest* axis is recursively divided using a hyperplane on a median point. However, the partitions are stored in leaf nodes, which is a feature borrowed from B-trees. In QuadTrees each node has *exactly* four children Samet (1984). This method recursively subdivides the most dense areas to four smaller ones. To test the two approaches for spatial partitioning, we used an implementation available in GeoSpark Yu et al. (2019). Sample results in the area of the German Bight are presented in Fig. 2a and b. Spatial partitioning is used by the genetic algorithm, in which each partition is treated separately. Since the population and other parameters of the genetic algorithm are set per single partition, a denser partitioning results in more waypoints in the end.

4.2.2 Genetic Algorithm

Genetic algorithms are the biologically-inspired family of algorithms, in which the process of evolution is simulated Sivanandam and Deepa (2008). These algorithms constitute an important branch of the field of artificial intelligence, namely the evolutionary computing. The solution to the problem is represented as a *population*. The overall population consists of entities called *chromosomes*. Each chromosome is built from *genes*. As in a real population, having "good" genes results in a higher chance of having offspring. The extent of being "good" is measured by a *fitness function*. Two chromosomes with good genes produce a new one by combining their genes in a *crossover* process. This results in a better population, where chromosomes with low fitness scores are replaced by new ones. Moreover, random changes are introduced to some of the genes – this process is called *mutation* and is used to maintain population diversity. The full cycle in which a new generation is created is called an *epoch*. There is an assumption that after a sufficient number of epochs, the resulting population will be much improved (in terms of fitness) compared to the initial one The process is stopped after a predefined number of epochs or when a convergence criterion is met.

We use the genetic algorithm to discover waypoints from AIS data. In our case, each gene will represent a waypoint candidate. The genetic algorithm is run on each partition separately, and the results are concatenated at the end, thus the process is parallelized. Making the use of Apache Spark, it is also distributed. The overall idea is summarized in Algorithm 2. After the partitioning step, the



(a) k-d B-tree (without waypoints) **(b)** QuadTree (without waypoints)

Fig. 2 Spatial partitioning methods in the area of the German Bight. Blue dots mark AIS points after filtering with the CUSUM algorithm and red rectangles denotes separate partitions (color figure online)

algorithm is run for each partition by passing the function DiscoverWaypoints(). When the initial population is generated, the process of generating new offspring is repeated $n-1$ times, where n is the number of epochs.

Algorithm 2 Parallel genetic AIS waypoints discovery

```
 1: function GENETIC-AIS-WAYPOINTS-DISCOVERY($rdd$, $n_{\text{part}}$)
 2:     $rdd \leftarrow$ PARTITION($rdd$, $n_{\text{part}}$)
 3:     $w \leftarrow rdd$
 4:         .MAPPARTITIONS(DISCOVERWAYPOINTS)          ▷ Distributed and parallel
 5:         .DISTINCT()
 6:     return $w$
 7: function DISCOVERWAYPOINTS($AIS$, hyperparams)
 8:     $p \leftarrow$ INITIALISEPOPULATION($AIS$, hyperparams)
 9:     for $i \leftarrow 2$ to $n_{\text{epochs}}$ do
10:         $p \leftarrow$ GENERATEOFFSPRING($AIS$, $p$)
11:     return ITERATOR($p$)
```

Following Dobrkovic et al. (2018), a good waypoint candidate is a point that has many AIS points in its proximity. We formalize it with a simple circle-like equation. Firstly, we need to define a gene – in our case, it's a triple (x, y, r), where x represents longitude, y latitude, and r radius (constant for all genes). A single chromosome contains a fixed number of genes (referred to as a chromosome length). A set of chromosomes constitutes a population. Contrary to the method of Dobrkovic et al. (2018), we initialize our population drawing random AIS points from the actual population.

4.2.3 Fitness Function

We calculate the fitness value of a chromosome f using the following formula:

$$f = \frac{1}{N}\sum_{i=1}^{N} \#\{(x,y) \in \mathcal{P} : \text{hav}(x,y,x_{c_i},y_{c_i}) \leq r\}, \qquad (4)$$

where N is the number of points (x, y) in a given partition $\mathcal{P}$. Every single gene in the chromosome carries a waypoint candidate (x_{c_i}, y_{c_i}), which actually denotes the center of the circle with a radius r (in degrees). The $\#$ operator marks cardinality of a set. For calculating the great circle distance between (x, y) and (x_{c_i}, y_{c_i}), we use the standard haversine formula:

$$\mathrm{hav}(\lambda_1, \phi_1, \lambda_2, \phi_2) = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \quad (5)$$

where λ_i and ϕ_i represent longitude and latitude (both in radians) of two points between which the distance is to be measured. For the radius of Earth R we use the standard value of 6372.8 km. If a chromosome is eligible for penalty (see the following paragraphs), its fitness is set to zero.

4.2.4 *One-Point Crossover with a Roulette Wheel Selection and Mutation*

The crossover is an operation in which a new chromosome is generated from two existing ones. Our implementation generates the new population by means of roulette wheel selection and one point crossover. Conceptually, two parents for a new chromosome are selected using the roulette wheel. The new parents are not drawn from the whole population in a uniform way – the chance of being drawn is proportional to their fitness. Therefore, the process resembles a roulette with uneven sections. Having selected two parents, we use the one-point crossover to generate a new chromosome. This procedure draws a random point at which the parents are combined. The same random number for the one-point crossover is also used for the mutation. The mutation also occurs if the resulting chromosome has its fit equal to zero.

4.2.5 *Penalties for Chromosomes*

To prevent the situation in which all waypoints are picked in a very dense area (leaving aside the less populated ones), we introduced a mechanism for penalizing such configurations. A chromosome can be perceived in terms of two values: fitness and diversity. The first term was already introduced. The second reflects how many different genes a given chromosome consists of. The diversity is the proportion of a number of unique genes to the number of all genes. This, however, would only enable to detect exactly the same waypoint candidates. Since overlapping waypoints (i.e., genes close to each other) have to be penalized as well, we check if waypoints are unique by checking if two circles are disjoint:

$$\mathrm{hav}(x_1, x_2, y_1, y_2) \geq 2r. \quad (6)$$

If the condition is not met, the chromosome receives 0 for its fitness score. Finally, a check whether a chromosome is eligible for a penalty is carried out. The chromosome is eligible for penalty either if the minimal diversity score is not reached, or if there exists at least one pair of overlapping genes.

4.3 Edges

The genetic algorithm described in the previous section generates a set of waypoints. Waypoints are equivalent to nodes of a graph. We need a method to discover the edges, i.e., which waypoints should in fact be connected. Based on historical AIS data, we look at every single trajectory of all vessels that passed an area of interest and track which waypoints they "visited". It is therefore necessary to assign the nearest waypoint for each AIS point. Having all AIS points annotated with the nearest waypoints, it may seem straightforward to reconstruct the connections between waypoints.

We refer to the process of adding information about the nearest waypoint to each AIS row as *AIS enrichment*. We add both the identifier of a waypoint and the distance to it. Algorithm 3 formalizes the approach. The core function is NeighbourKnn. Taking into account the number of rows in AIS data, the process of assigning waypoints to AIS data proved to be very time consuming. We introduced several optimization techniques to make the task feasible, including vectorized versions of distance calculation functions. Comparing the refined version to the initial approach based on row-by-row iteration, we achieved a 200.000× increase in rows/second throughput.

Algorithm 3 AIS enrichment (Spark-like notation)

```
1: function AssignNearestWaypoints($AIS$)
2:     $AIS_w \leftarrow AIS$.withColumn('waypoint', NeighbourKnn($AIS.lat, AIS.lon$))
3:     return $AIS_w$
4: function NeighbourKnn($lat, lon$)
5:     $w \leftarrow$ ReadWaypoints
6:     $nnModel \leftarrow$ NearestNeighbours($algorithm, metric$)
7:         .fit($w$)
8:     $n \leftarrow nnModel$.kneighbours($lat, lon, n_neighbours = 1$)
9:     return $n$
```

Having assigned the waypoints, the next step is the reconstruction of edges between these waypoints. The generation of edges proved to be a less challenging task from the performance point of view. The only optimization step that had to be applied was a materialization of the enriched AIS dataset. For some reason even caching in Apache Spark was not helpful – grouping of edges spawned re-calculation of the closest waypoints. Nevertheless, there were other challenges concerning the output graph. A visual introspection of maps with generated graphs proved that the method discovered "impossible" connections between some waypoints which further on had to be eliminated. It was caused partly by the low AIS data quality. A general approach to the reconstruction of edges is presented in Algorithm 4. It lists the filtering functions that are applied either to obtain graphs for different conditions or just to improve the quality:

- FilterAis – the function selects a subset of data for a given vessel type (e.g., tankers) or weather conditions (e.g., heavy wind). It is also used to build a graph on a subset of points, i.e., only important maneuvering points as identified by CUSUM (see Sect. 4.1).
- FilterTrajectory – the function is applied to trajectories of a ship. It is responsible for the selection of points out of which edges will be constructed. For example, it can only leave out AIS points that are transition points from one waypoint segment to the other (border-points). In another variant, it is used to consider as input only those edges that connect points visited within a specific time period (time-bound).
- FilterEdges – the function is applied to edges. For example, we can filter out edges that are too long (e.g., distance > 250 km) or are very rare (e.g., followed by only a single vessel). The method is mostly applied to visualizations.

consecutive AIS messages belonging to the same waypoint, especially if the distances between waypoints are long. We need to identify only these places where the "borders" between areas belonging to different waypoints are crossed, i.e., a given AIS message has a different waypoint from the previous message. In the implementation, we achieve this by applying the so-called window functions for obtaining previous values of the analyzed column. We then mark respective rows as 'changed' and classify them as edge seeds. They contain only the points where a current waypoint (`to_waypoint`) is different from the previous waypoint (`from_waypoint`). This procedure significantly reduced the number of rows, thus improving efficiency of edge generation. We can then construct a dataset with edges using a grouping by `from` and `to`, as illustrated in Algorithm 4. In this manner we preserve information about the directions of edges. We also calculate group statistics, like a number of vessels traversing specific edges or time-related statistics for further filtering.

5 A Use Case with Evaluation

The quality of results provided by methods described in the previous section depends highly on numerous hyper-parameters. It is not feasible to optimize all of them at once and there is also not a single optimization criterion. Therefore, we initially followed the greedy optimization approach – we split the evaluation into components and optimized them separately. The optimal solution for the whole system is derived from optimal solutions for the components. For evaluation purposes we used two real AIS datasets of different quality: one covering the area of the German Bight ($53° \leq \phi \leq 57°$, $2.5° \leq \lambda \leq 9.5°$) and the second one for the Baltic Sea ($53.1° \leq \phi \leq 60.94°$, $13° \leq \lambda \leq 30.73°$). We considered only messages from passenger, tanker, and cargo vessels with a navigational status 0 or 8. The German Bight data are not perfect and reflect typical problems with AIS data, such as incorrect positions reported and missing coverage in some areas. It

Algorithm 4 Edges discovery

```
1:  function DiscoverEdges($AIS$)
2:      $AIS_f \leftarrow$ FilterAIS($AIS$)
3:      $AIS_w \leftarrow AIS_f$.partitionBy('mmsi').orderBy('timestamp_ais')
4:      $AIS_w \leftarrow AIS_w$.withColumn('to_waypoint')            ▷ mark current waypoint
5:      $AIS_w \leftarrow AIS_w$.withColumn('from_waypoint')          ▷ mark previous waypoint
6:      $AIS_w \leftarrow AIS_w$.withColumn('changed')     ▷ identify rows with changed waypoints
7:      $AIS_c \leftarrow$ FilterTrajectory($AIS_w$)
8:      $edges \leftarrow AIS_c$.groupBy('from_waypoint', 'to_waypoint')
9:      $edges \leftarrow$ FilterEdges($edges$)
10:     $edges_d \leftarrow edges$.withColumn('distance_km')   ▷ calculate distance between waypoints
11:     return $edges_d$
```

When a vessel is moving along its trajectory, it passes many waypoints. We know which points are passed by, as our AIS data is already annotated with the closest waypoints (AIS enrichment). Sometimes there are several was used for the qualitative testing of the genetic algorithm. The Baltic Sea dataset does not have such problems – it was used to test the solution as a whole. We used a 48-core AMD EPYC server with 768 GB RAM and 36 TB HDD, though the presented algorithms can be run on much smaller machines. All of the cores remained busy during the most of time, which suggests a good parallelization of the algorithm. Maximum usage of hardware capabilities was one of our priorities. Since we used Apache Spark for all the calculations, the solution might be also considered as distributed and scalable.

5.1 CUSUM

There is no standard methodology for evaluating this type of algorithms. Theoretical advice can be found in some publications Gustafsson (2000). The performance of the change detection algorithm was evaluated in several steps. The main purpose was to find optimal parameters, such as the threshold h and the number of historical data n_{sma} that should be taken into account in the moving average. During this process, only individual voyages, limited by fixed coordinates were considered. The key was to find different tracks in terms of change of course and speed. Having found a set of vessels and their tracks, we manually assigned "expert" points on the map that should be alerted by the algorithm. A single expert point was a circle with a radius of 500 m. The main idea was to perform a classification of waypoints returned by the CUSUM and to calculate some measures based on it.

In the next step, we provided a range of evaluated hyper-parameters. For the threshold h, we chose between 1 and 6, whereas for the number of historical AIS messages n_{sma} we tested values between 2 and 10. The algorithm was executed for each combination of hyper-parameters' set. In a single iteration, the k-means algorithm for the list of waypoints returned by CUSUM was calculated, with a number of clusters equal to a number of expert points. The following measures were calculated in subsequent iterations:

1. Distance from the clusters centroids to the nearest expert point. For each expert point, the mean distance was calculated.
2. Confusion matrix that matches waypoints to the nearest expert point. If a waypoint is within the radius of an expert point, it is assigned to it.
3. A number of unassigned expert points. This means that there were no waypoints detected within the radius of 500 m of this expert point.

The percentage of unassigned expert points and the unassigned waypoints (outliers) turned out to be the most important measures, because they indicated whether CUSUM met the requirements set by the expert. The mean distance from centroids provided information about a distribution of the waypoints and punished outliers as well.

Each point has been classified to one of the four groups: true negative, false positive, true positive, and false negative. Based on the matrix, the respective measures were calculated, including accuracy, recall, specificity, and precision. We aggregated all single, manually collected tracks with the respective results and chose the highest-rated parameters among all samples. It turned out that the optimal combination is the threshold of 1.25 and the number of historical observations of 8. For three runs on the 8-week data from the Baltic Sea area (separately for each vessel type – tanker, cargo, and passenger), the total wall time was 15 min and 42 s, which gives roughly 5 min on average for a single type.

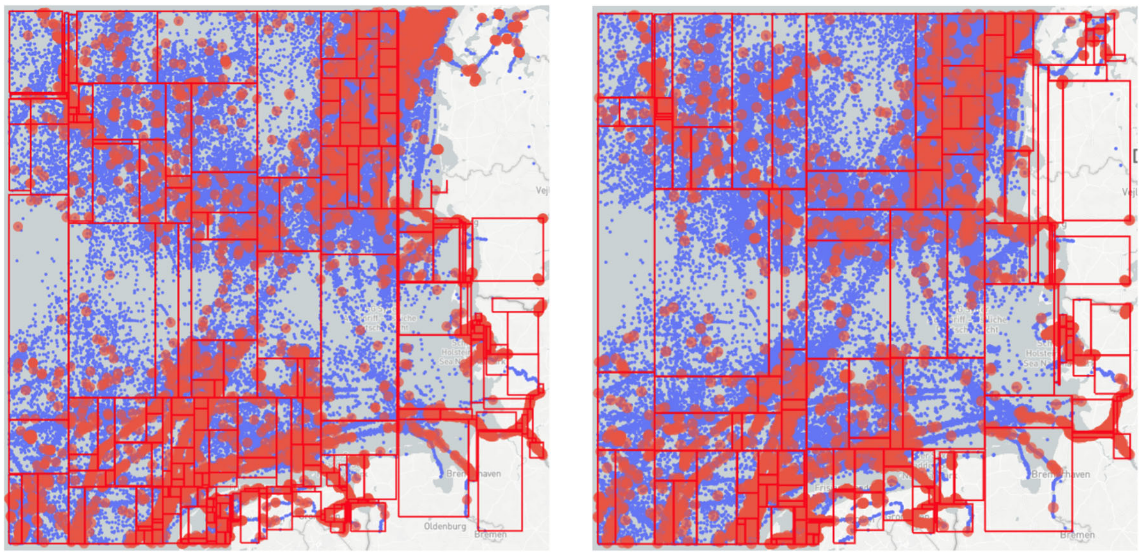
5.2 Genetic Algorithm

This section contains the results of evaluation of the genetic algorithm – the first part concerns the partitioning, and the second one the hyper-parameters of the genetic algorithm. The goal of the first was to select the method capable of finding a balanced distribution of the CUSUM-generated AIS points among partitions. Hyper-parameters that control behavior of the genetic algorithm can be tuned more easily when AIS points in partitions are evenly distributed. We performed a series of experiments, in which the two partitioning methods were compared (each for 64, 128, and 256 partitions). The results for the 4-week dataset of the German Bight are presented in Table 1. Based on the obtained results, we have chosen k-d B-trees, since the standard deviation of the amount of points in each partition tends to be smaller with numerous settings than in the other method. It is worth to mention that QuadTree has a visible tendency to produce scarcely populated partitions. Another test on the Baltic Sea data led to the same conclusion (not presented in this paper).

The evaluation of the genetic algorithm itself is less obvious, as there is no ground-truth data to compare the results with. Therefore, we rely on a qualitative evaluation of the results. There is a number of hyper-parameters to control: *chromosome length*, *radius*, *minimal diversity*, *population size*, *epochs*, *mutation factor*, *number of partitions*, and *weeks*. All the tests were conducted using the CUSUM-filtered data from the German Bight. At first, we tested the algorithm for different numbers of partitions and for 100 chromosomes in the population (1-week data). The initial observation was that the most noticeable changes can be observed in the densest sea corridors, letting it appear more continuous (resulting rather in a smooth route, as opposed to long gaps between waypoints), whereas areas scarce in waypoints did not change much. Somewhat

Table 1 Evaluation of the partitioning algorithms for the German Bight

	$p = 64$		$p = 128$		$p = 256$	
	k-d B-Tree	QuadTree	*k*-d B-Tree	QuadTree	*k*-d B-Tree	QuadTree
Count	99	232	188	583	370	1426
Mean	2164.35	923.58	1139.74	367.53	579.11	150.26
SD	674.99	1023.61	441.94	499.20	256.38	219.36
Min	1054	0	309	0	67	0
25%	1537	26	818	1	378	0
50%	2084	501	1062	112	550	31
75%	2620	1604	1436	569	731	249
Max	4338	4794	2969	2590	1484	1291



(a) partitions=256, population=20, r=0.3, epochs=200, mf=25%, weeks=8, cl=6

(b) partitions=128, population=30, r=0.3, epochs=300, mf=25%, weeks=8, cl=6

Fig. 3 Different test settings for 8-week data for the German Bight

similar results are produced for smaller population size (40 chromosomes). In comparison, increasing the number of partitions to 512 generated too many waypoints.

Next, we tested different values of epochs and radius. The empirical tests showed that the algorithm quickly converges to the solution – perhaps due to the fact that the population is drawn from the existing AIS points, contrary to the findings presented by Dobrkovic et al. (2018). The conducted experiments show that 200–300 epochs is sufficient, since further increasing of the number of epochs does not lead to a significant improvement of results. Setting the correct radius is tricky, since excesively high values are handled poorly in the dense areas (the diversity condition cannot be met). We also experimented with a mutation factor. That value must be relatively high due to the fact that the algorithm can "get stuck" in small and dense partitions, so random noise is needed. To partially mitigate the problem with the lack of AIS data coverage in some areas, we extended the analyzed dataset from 1-week AIS data to 4 weeks. The obtained results improved, as the new and desired waypoints emerged in previously empty spaces, merging dense corridors. However, along with a further extension of the dataset to 8 weeks, the results seemed to be similar. Figure 3a and b present some of the test scenarios.

The efficiency tests were conducted on 8-week data from the Baltic Sea for three filtered AIS datasets (passenger, cargo, and tankers). We ran the algorithm several times with different numbers of *k*-d B tree partitions (64, 128, 256), chromosome length and population (5, 10, 20 – the same number was used for both parameters) – each run took 300 epochs, with the radius set to 3.0 km, the minimum diversity to 25% and the mutation factor to 10%. This results in 27 combinations. The wall time for all of them was 2 min 51 s.

To sum up the results of different test settings for the genetic algorithm, it was observed that more partitions with smaller chromosomes seem to be better than fewer partitions and longer chromosomes if one wants to avoid stacking numerous waypoints in small areas. In general, the choice of hyper-parameters is area-specific and general values working in all areas cannot be determined. When CUSUM is used as input to the genetic algorithm, our recommendation is to use at least 4-week datasets. Moreover, the algorithm is very prone to missing data (in terms of the AIS data coverage) – it just does not generate waypoints in such areas. Therefore, pre-processing with trajectory reconstruction algorithms can be considered. Nevertheless, the algorithm deals quite well with single wrong AIS points (spoofed or misread).

5.3 Edges

There is no strict methodology to evaluate CUSUM and genetic algorithms directly. They were used for the purpose – generation of edges, therefore we evaluated the final output through the edges. Of course, the resulting network depends on the waypoints provided, i.e. the quality of the results of the previous steps: CUSUM and the genetic algorithm. Their impact can be partly alleviated by our approach in cases where not the whole network is evaluated but the routes that can be planned (the shortest or the fastest).

The efficiency tests were conducted on the same 8-week data from the Baltic Sea. The algorithm run was repeated for each of the 27 combinations of parameters resulting in separate sets of waypoints. According to the description in Sect. 4.3 we measured the time of the separate processing steps. First, 27 enriched AIS datasets were provided, which took 7 min 32 s. of wall time total, 16.7 s for a single dataset. Second, for each AIS dataset 16 variants of networks were provided (considering the various draught, length and width of vessels), yielding 432 networks in total. The calculation time was 23 h 26 min wall time total, 3 min 15 s on average for a single network. Third, for better understanding we also prepared visualizations and a graph analysis (also centrality), which took further 3 h 15 min in total, 28 s on average for a single network (all on 48 cores).

For clarity we show some of the generated meshes. Figure 4 depicts edges colored according to the maximum width of the ship. The darker the edge, the bigger is the maximum width of vessels sailing this route. We can easily see that very large ships go to Gdańsk, and large ships to other ports like Riga and Sankt Petersburg. Figure 5 uses colors to visualise directions both of edges and nodes. Red edges are directed towards north, blue – south, violet – both directions. Waypoint colors indicate the average direction of vessels sailing through them, calculated from course over ground (similar results are for the ships' heading). The more saturated the color is, the more consistent the direction. Traffic separation schemes can easily be identified.

5.4 Evaluation

Our methodology for evaluation is based on a golden standard – the output of our method is used to construct a recommended route which is then compared with real recommendations. We compared our vessel routes with routes generated using a contemporary software for navigation – `searoutes.com`. For the test purposes, we obtained four routes within the area of the Baltic Sea: PLGDN-RULED, RUKGD-LVRIX, SEKAA-FITKU, and SESTO-LVRIX. For all below presented results, an 8-week AIS dataset was used (2019, weeks 40–48). The AIS data was filtered, so that only AIS from the aforementioned part of the Baltic Sea for tankers, cargo, and passenger ships were included separately.

The trajectories provided by our graph and the ground-truth trajectories had different numbers of waypoints, therefore comparing them was not trivial. Out of few existing solutions we used Symmetrized Segment-Path Distance - SSPD Besse et al. (2015). Let T^i be i-th trajectory (i.e. route) of length n^i, in which p^i_k is the k-th location of T^i, and s^i_k is a line segment between p^i_k and p^i_{k+1}. The Segment-Path Distance $\mathrm{D}_{\mathrm{SPD}}$ from T^1 to T^2 is the average of all distances D_{pt} calculated from points of T^1 to the trajectory T^2. Segment-Path distance is the smallest Point-to-Segment distance D_{ps} from a given point of T^1 to a segment in T^2. More formally:

$$\mathrm{D}_{\mathrm{SSPD}}(T^1, T^2) = \frac{\mathrm{D}_{\mathrm{SPD}}(T^1, T^2) + \mathrm{D}_{\mathrm{SPD}}(T^2, T^1)}{2}, \tag{7}$$

$$\mathrm{D}_{\mathrm{SPD}}(T^1, T^2) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} \mathrm{D}_{pt}(p^1_{i_1}, T^2), \tag{8}$$

$$\mathrm{D}_{pt}(p^1_{i_1}, T^2) = \min_{i_2 \in [0, \ldots, n_2 - 1]} \mathrm{D}_{ps}(p^1_{i_1}, s^2_{i_2}), \tag{9}$$

$$\mathrm{D}_{ps} = \begin{cases} \mathrm{hav}\left(p^1_{i_1}, p^{1\mathrm{proj}}_{i_1}\right) & \text{if } p^{1\mathrm{proj}}_{i_1} \in s^2_{i_2}, \\ \min\left\{\mathrm{hav}\left(p^1_{i_1}, p^2_{i_2}\right), \mathrm{hav}\left(p^1_{i_1}, p^2_{i_2+1}\right)\right\} & \text{otherwise}, \end{cases} \tag{10}$$

where $p^{1\mathrm{proj}}_{i_1}$ is the orthogonal projection of $p^1_{i_1}$ in $s^2_{i_2}$.

From Table 2 we can conclude that more complex graphs (i.e. built with more partitions, genes, and chromosomes) result in smaller SSPD. This may be explained

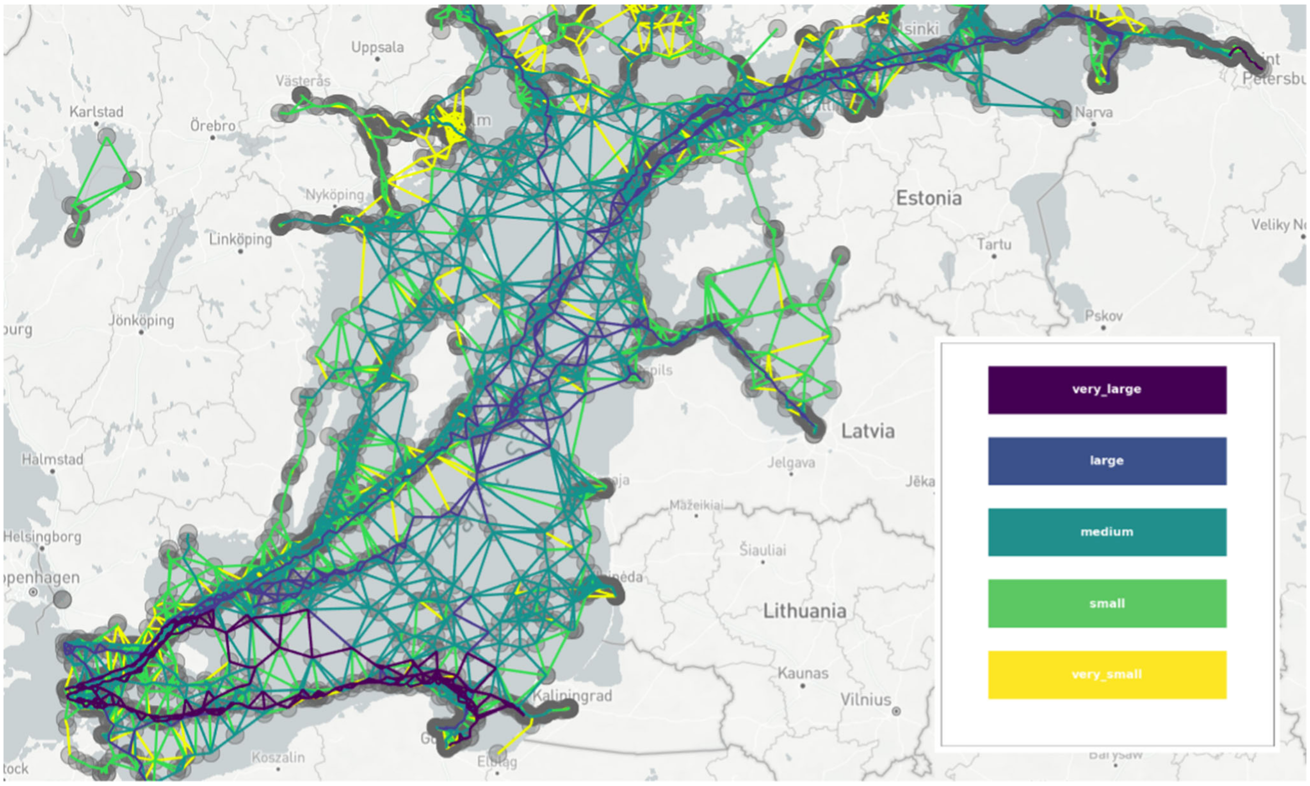


Fig. 4 Graph for the Baltic Sea generated on AIS data for an 8-week period (cargo vessels, 128 partitions, 100 genes per partition)

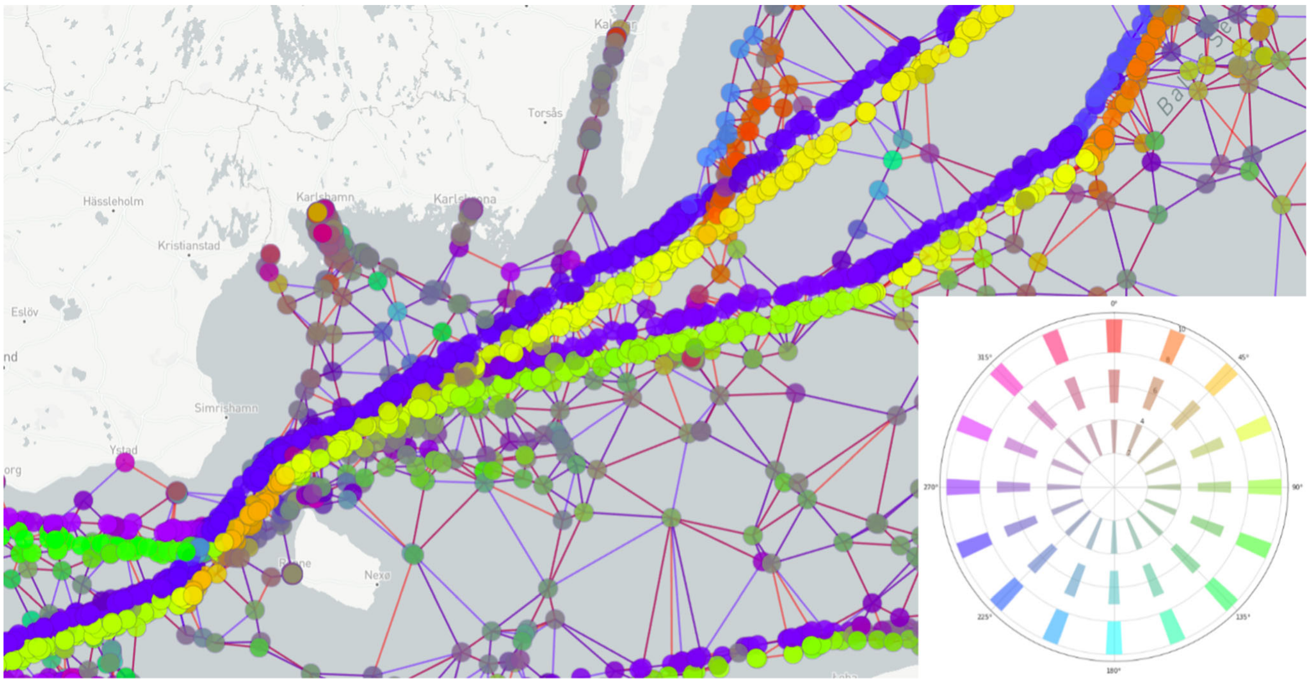


Fig. 5 Directed graph for the Baltic Sea generated on AIS data for an 8-week period (tankers, 256 partitions, 400 genes per partition)

by the fact that the genetic algorithm needs a sufficient number of genes to maintain population diversity and succeed. Therefore, the problem of parameter optimization seems to be rather about finding a good trade-off between graph simplicity and accuracy, instead of minimizing a single criterion. In our test setting, graphs generated for

Table 2 Comparison of the routes obtained with the proposed traffic network generation method with the routes from `searoutes.com`

Type	Part.	Pop.	PLGDN-RULED		RUKGD-LVRIX		SEKAA-FITKU		SESTO-LVRIX	
			D_{SSPD}	Rank	D_{SSPD}	Rank	D_{SSPD}	Rank	D_{SSPD}	Rank
C	64	S	18895.00	15	6387.26	3	32872.80	18	18964.98	25
		M	13568.92	11	8271.93	7	8107.48	5	19212.01	26
		L	12808.89	9	6929.86	4	6141.34	3	11443.68	16
	128	S	12245.38	7	11190.66	13	28842.61	15	12639.47	18
		M	8850.81	4	12109.40	17	7760.20	4	13343.73	20
		L	7712.39	3	12990.64	19	36441.50	22	6198.29	5
	256	S	18490.32	13	6352.03	2	32850.07	17	18103.47	23
		M	4528.00	1	8656.73	8	8353.06	6	7912.32	9
		L	5294.03	2	7384.74	5	5338.34	2	8267.09	11
P	64	S	38306.43	25	26365.01	27	54014.27	25	21148.30	27
		M	39041.14	26	20308.47	26	12592.17	9	11601.09	17
		L	21366.97	16	12905.82	18	33539.62	19	7771.14	8
	128	S	63183.83	27	14594.44	22	21880.23	10	6102.02	4
		M	28687.85	22	8235.09	6	45266.73	24	8276.40	12
		L	33451.82	24	11369.14	14	26842.43	13	5490.14	3
	256	S	31405.24	23	14011.30	21	12041.09	8	8158.11	10
		M	28609.40	21	13404.37	20	26458.20	12	8864.62	13
		L	26538.14	20	11102.31	12	35260.95	21	13221.02	19
T	64	S	18614.22	14	16693.67	23	40085.20	23	15529.71	21
		M	11741.68	6	17286.38	24	34459.10	20	7312.04	7
		L	12515.64	8	12020.13	16	28589.48	14	6547.74	6
	128	S	26435.09	19	5707.78	1	77791.12	26	18962.90	24
		M	9864.21	5	10091.33	9	31022.88	16	9988.21	14
		L	23980.78	17	11723.70	15	84676.07	27	16783.00	22
	256	S	13861.74	12	18598.61	25	9612.71	7	11001.06	15
		M	24284.61	18	10717.86	10	24669.91	11	5419.50	2
		L	13028.73	10	11022.79	11	4810.51	1	3346.45	1

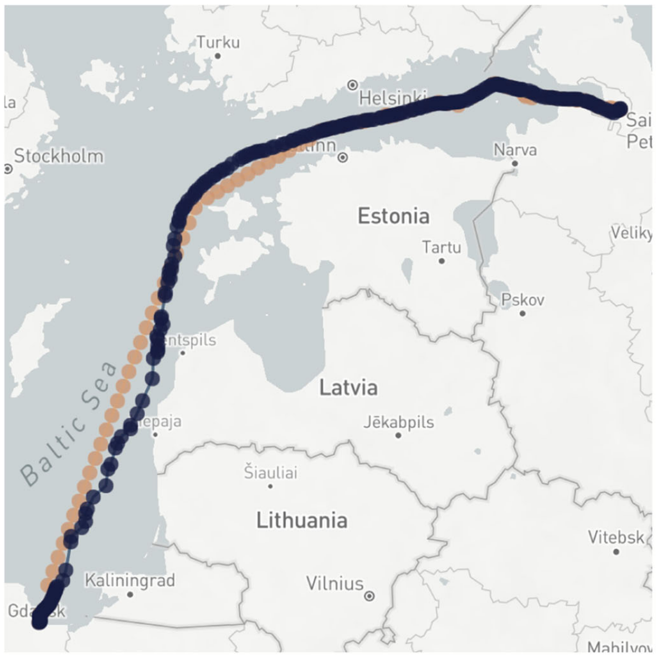
The values are presented by the SSPD metric (smaller is better). Abbreviations and parameters: C – cargo, P – passenger, T – tanker, part. – number of partitions for *k*-d B-trees, pop. – population and chromosome size in the genetic algorithm (S – 5 chromosomes × 5 genes, M – 10 × 10, L – 20 × 20). Each graph was built within 300 epochs, using *r*=3.0 km and 10% mutation factor

cargo and tanker vessels resembled the Searoutes network more closely than those for passenger vessels. Perhaps the data from passenger vessels might be *biased* with regard to the algorithm due to the fact that some vessels of this kind often operate on short routes with a certain repetitive pattern, which results in a vast number of AIS messages in a relatively small area. There are also some inconsistencies in the results (e.g., the best result of RUKGD-LVRIX), which might be attributed to the stochastic nature of the genetic algorithms (see Fig. 6 for summary).

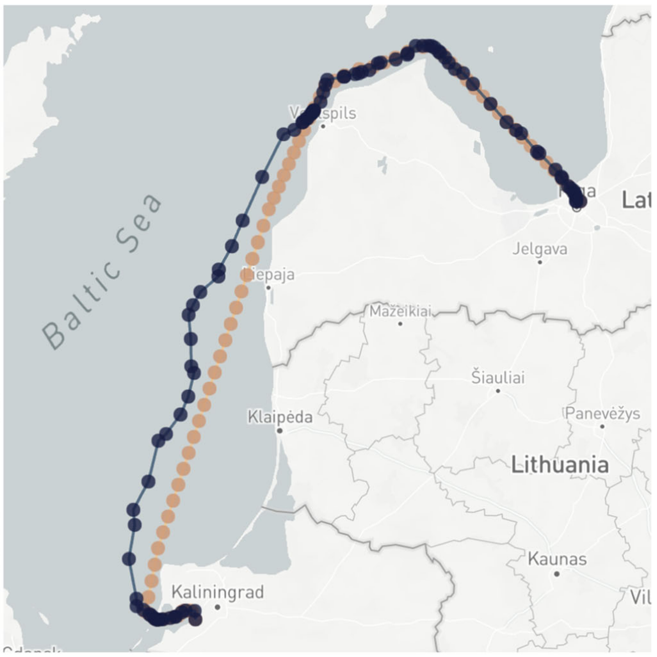
6 Summary

In this paper we introduced a method for the automatic reconstruction of a network reflecting the maritime traffic using AIS data. Such a network can be later used in vessel routing and voyage planning. In the presented method, several approaches were refined, such as the CUSUM method and the genetic algorithm, inspired by the work of Lamm and Hahn (2017) and Dobrkovic et al. (2018). Not only novel concepts were introduced, but they were also implemented and tested with a parallel and distributed computational environment on Apache Spark. Our implementation is scalable and can work with real-world AIS data streams. We also provided a real-world use-case for the Baltic Sea and compared our routes with the ones from `searoutes.com`.

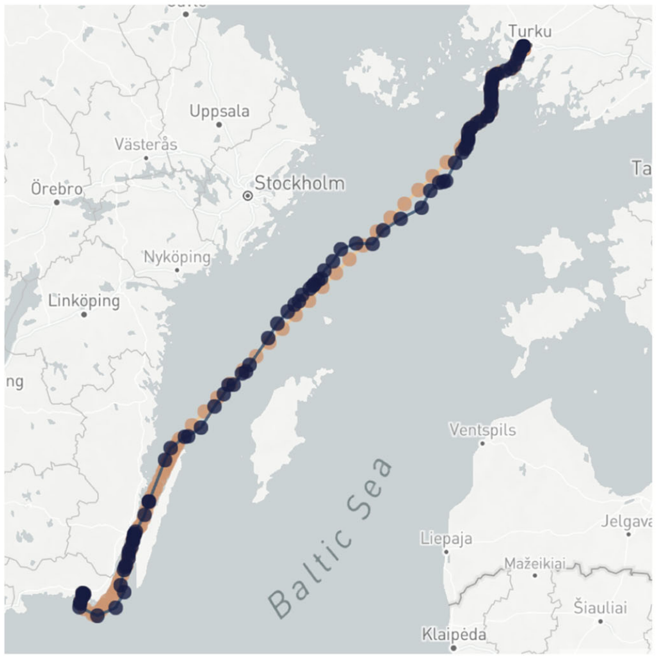
Several findings of this study initiate further discussion. Since the method is prone to incorrect or missing AIS messages, there is a room for improvement in these aspects. For the areas that are relatively scarcely filled with AIS data, currently the method does not yield a good quality graph. A potential alleviation for this issue is to use AIS trajectory reconstruction methods for filling the “gaps”. This problem can be tackled by a number of methods. Linear interpolation is one example – it was used, e.g., by Mao et al. (2018). Lz et al. (2015) presented a more sophisticated data interpolation method, which uses line, arc, and curve trajectories and is dedicated to inland AIS data. In other example, Nguyen et al. (2015) used piecewise cubic Hermite interpolation.



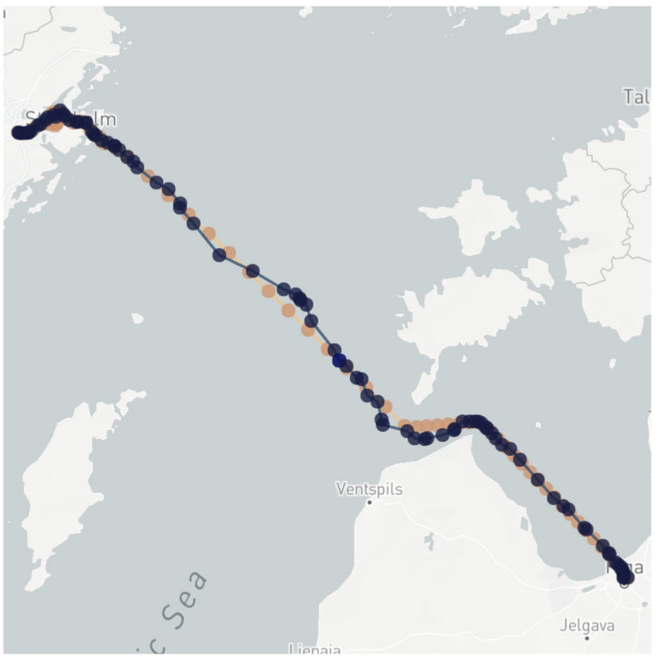
(a) PLGDN-RULED



(b) RUKGD-LVRIX



(c) SEKAA-FITKU



(d) SESTO-LVRIX

Fig. 6 Our results (blue) compared to `searoutes.com` (orange) for the graph generated from tanker data (256 partitions, large population) (color figure online)

In this article, we focused solely on the graph creation that is used later for finding an optimal route. This means that it does not consider fuel efficiency or weather conditions. We have already conducted further research that takes into account weather conditions. However, due to the limited volume of this paper, its result will be published as a separate work. Lastly, since the topological representation of a graph tends to result in a *zig-zag* route, a method for generating more smooth paths could be considered.

Acknowledgements This work was supported by a grant provided for the project HANSA: Retrospective Analysis of Historical AIS Data for Navigational Safety Through Recommended Routes.

HANSA is funded by the MarTERA partners: German Federal Ministry of Economic Affairs and Energy (BMWi), Polish National Centre for Research and Development (NCBR) and Research Council of Norway (RCN) and co-funded by the European Union.

CRediT Authorship Contribution Statement DF: conceptualization, data curation, formal analysis, investigation, software, validation, visualization, writing – original draft, writing – review and editing. KW: conceptualization, data curation, formal analysis, investigation, software, validation, visualization, writing – original draft, writing – review and editing. MS: conceptualization, funding acquisition, methodology, project administration, supervision, writing – original draft, writing – review and editing. MM: formal analysis, investigation, software, writing – original draft, writing – review and editing. WA: funding acquisition, project administration, supervision, writing – review and editing.

References

Arguedas VF, Pallotta G, Vespe M (2017) Maritime traffic networks: from historical positioning data to unsupervised maritime traffic monitoring. IEEE Trans Intell Transp Syst 19(3):722–732

Basseville M, Nikiforov IV (1993) Detection of abrupt changes: theory and application. Prentice Hall, Englewood Cliffs

Bellman R (1952) On the theory of dynamic programming. Proc Natl Acad Sci 38(8):716–719

Besse P, Guillouet B, Loubes JM, François R (2015) Review and perspective for distance based trajectory clustering. arXiv preprint arXiv:150804904

Bijlsma S (2001) A computational method for the solution of optimal control problems in ship routing. Navigation 48(3):144–154

Cai Y, Wen Y, Wu L (2014) Ship route design for avoiding heavy weather and sea conditions. TransNav Int J Mar Navig Saf Sea Transp 8:551–556

Calvert S, Deakins E, Motte R (1991) A dynamic system for fuel optimization trans-ocean. J Navig 44(2):233–265

Chen Z, Guo J, Liu Q (2017) DBSCAN algorithm clustering for massive AIS data based on the Hadoop platform. In: 2017 International conference on industrial informatics-computing technology, intelligent technology, industrial information integration (ICIICII). IEEE, pp 25–28

De Wit C (1990) Proposal for low cost ocean weather routeing. J Navig 43(3):428–439

Dobrkovic A, Iacob ME, van Hillegersberg J (2015) Using machine learning for unsupervised maritime waypoint discovery from streaming AIS data. In: Proceedings of the 15th international conference on knowledge technologies and data-driven business. ACM, p 16

Dobrkovic A, Iacob ME, van Hillegersberg J (2018) Maritime pattern extraction and route reconstruction from incomplete ais data. Int J Data Sci Anal 5(2–3):111–136

Ester M, Wittmann R (1998) Incremental generalization for mining in a data warehousing environment. In: International conference on extending database technology. Springer, Heidelberg, pp 135–149

Ester M, Kriegel HP, Sander J, Xu X et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd'96: Proceedings of the second international conference on knowledge discovery and data mining. pp 226–231

Faithfull W (2017) Change detection for software engineers part I: introduction and CUSUM. https://faithfull.me/change-detection-for-software-engineers-part-i-introduction-and-cusum/

Fang MC, Lin YH (2015) The optimization of ship weather-routing algorithm based on the composite influence of multi-dynamic elements (ii): optimized routings. Appl Ocean Res 50:130–140

Gustafsson F (2000) Adaptive filtering and change detection. Wiley, Hoboken

Hagiwara H, Spaans J (1987) Practical weather routing of sail-assisted motor vessels. J Navig 40(1):96–119

Haltiner G, Hamilton H, Arnason G (1962) Minimal-time ship routing. J Appl Meteorol 1(1):1–7

Hevner A, Chatterjee S (2010) Design research in information systems: theory and practice, vol 22. Springer, Berlin

Hevner AR, March ST, Park J, Ram S (2008) Design science in information systems research. Manag Inf Syst Q 28(1):6

James RW (ed) (1957) Application of wave forecasts to marine navigation. https://trid.trb.org/view/388400

Kosmas O, Vlachos D (2012) Simulated annealing for optimal ship routing. Comput Oper Res 39(3):576–581

Lamm A, Hahn A (2017) Detecting maneuvers in maritime observation data with CUSUM. In: 2017 IEEE international symposium on signal processing and information technology (ISSPIT). IEEE

Maki A, Akimoto Y, Nagata Y, Kobayashi S, Kobayashi E, Shiotani S, Ohsawa T, Umeda N (2011) A new weather-routing system that accounts for ship stability based on a real-coded genetic algorithm. J Mar Sci Technol 16(3):311

Mannarini G, Pinardi N, Coppini G, Oddo P, Iafrati A (2016) VISIR-I: small vessels-least-time nautical routes using wave forecasts. Geosci Model Dev 9(4):1597–1625

Mao S, Tu E, Zhang G, Rachmawati L, Rajabally E, Huang GB (2018) An automatic identification system (AIS) database for maritime trajectory prediction and data mining. In: Proceedings of ELM-2016. Springer, pp 241–257

Marie S, Courteille E et al (2009) Multi-objective optimization of motor vessel route. In: Proceedings of the international symposium on TransNav, vol 9. pp 411–418

Mazzarella F, Vespe M, Damalas D, Osio G (2014) Discovering vessel activities at sea using AIS data: mapping of fishing footprints. In: 17th international conference on information fusion (fusion). IEEE, pp 1–7

Montes AA (2005) Network shortest path application for optimum track ship routing. PhD thesis, Monterey, California. Naval Postgraduate School

Nguyen VS, Im M, Lee S (2015) The interpolation method for the missing AIS data of ship. J Navig Port Res 39(5):377–384

Page ES (1954) Continuous inspection schemes. Biometrika 41(1/2):100

Pallotta G, Vespe M, Bryan K (2013) Vessel pattern knowledge discovery from AIS data: a framework for anomaly detection and route prediction. Entropy 15(6):2218–2245

Panigrahi J, Padhy C, Sen D, Swain J, Larsen O (2012) Optimal ship tracking on a navigation route between two ports: a hydrodynamics approach. J Mar Sci Technol 17(1):59–67

Robinson JT (1981) The KDB-tree: a search structure for large multidimensional dynamic indexes. In: Proceedings of the 1981 ACM SIGMOD international conference on management of data. ACM, pp 10–18

Samet H (1984) The quadtree and related hierarchical data structures. ACM Comput Surv (CSUR) 16(2):187–260

Sang L, Wall A, Mao Z, Xp Yan, Wang J (2015) A novel method for restoring the trajectory of the inland waterway ship by using AIS data. Ocean Eng 110:183–194

Schøyen H, Bråthen S (2015) Measuring and improving operational energy efficiency in short sea container shipping. Res Transp Bus Manag 17:26–35

Sen D, Padhy CP (2010) Development of a ship weather-routing algorithm for specific application in north indian ocean region. In: The international conference on marine technology. Dhaka, Bangladesh, BUET, pp 21–7

Shao W, Zhou P, Thong SK (2012) Development of a novel forward dynamic programming method for weather routing. J Mar Sci Technol 17(2):239–251

Sivanandam S, Deepa S (2008) Genetic algorithms. In: Introduction to genetic algorithms. Springer, Heidelberg, pp 15–37

Szłapczynska J, Smierzchalski R (2009) Multicriteria optimisation in weather routing. p 423

Tan WC, Weng CY, Zhou Y, Chua KH, Chen IM (2018) Historical data is useful for navigation planning: data driven route generation for autonomous ship. In: 2018 IEEE international conference on robotics and automation (ICRA). IEEE, pp 7478–7483

Tsou MC, Cheng HC (2013) An ant colony algorithm for efficient ship routing. Polish Marit Res 20(3):28–38

Tu E, Zhang G, Rachmawati L, Rajabally E, Huang GB (2018) Exploiting ais data for intelligent maritime navigation: a comprehensive survey from data to methodology. IEEE Trans Intell Transp Syst 19(5):1559–1582

Vettor R, Soares CG (2016) Development of a ship weather routing system. Ocean Eng 123:1–14

Wang HB, Li XG, Li PF, Veremey EI, Sotnikova MV (2018) Application of real-coded genetic algorithm in ship weather routing. J Navig 71(4):989–1010

Yu J, Zhang Z, Sarwat M (2019) Spatial data management in apache spark: the geospark perspective and beyond. Geoinformatica 23(1):37–78

Zhang SK, Shi GY, Liu ZJ, Zhao ZW, Wu ZL (2018) Data-driven based automatic maritime routing from massive AIS trajectories in the face of disparity. Ocean Eng 155:240–250