Applications of Probabilistic Interpolation to Ship Tracking

T.R. Hammond[1]

[1]DRDC Atlantic Research Centre, 9 Grove Street, Darmouth, NS, B2Y 3Z7

Abstract

Ships report their own position at predictable intervals via the Automatic Identification System (AIS) and Long Range Identification and Tracking (LRIT). Those able to receive these position reports can track the movement of vessels, but using self-reported positions raises new estimation challenges. One of these is the interpolation problem, which considers what happened between two successive position reports, *A* and *B*, from the same ship. This paper illustrates the practical significance of this problem in issues from oil-spill investigation to maritime security and fisheries management. It outlines a general Bayesian approach to the problem that is based on simulating large numbers of random tracks. The approach is illustrated using a fictitious Arctic scenario in which a contact, obtained from a radar satellite system, is to be associated with one of three AIS tracks, in the presence of ice. The method shows how the ice-breaking capabilities of the vessels can be accounted for in the association problem, a challenging task for traditional methods. Finally, the paper shows how to generalize from this simple association problem to more complex cases.

Key Words: Bayesian inference, situational awareness, AIS

1. Introduction

This paper is concerned with reconstructing past events involving ships, events around which the ships were employing self-reporting systems. The motivation behind such reconstruction is typically forensic. In the classic scenario, one of the ships may have done something suspicious, but it is not clear which one. In another common application, we are reconstructing the past behaviour of a suspect, looking for accomplices or verifying alibis. We want to eliminate the impossible but also to identify the more probable courses of events. This is a situation that arises in practice because of the widespread use of self-reporting systems.

Self-reported data, such as from the Automatic Identification System (AIS), allow widely-separated position reports to be attributed to the same vessel because the vessel is uniquely identified in each report. In contrast, radar contacts can typically only be associated with a given ship track when the contacts are relatively nearby. Thus, the problem of considering the possible trajectories of a vessel between widely separated points in space-time is one that is naturally associated with self-reporting.

With AIS data, as available in a nation's common operating picture, it is typical for gaps between successive position reports from a given ship to vary from seconds, to days, or even longer, due to variability in AIS coverage over the ship's trajectory (Hammond and Peters, 2012). Long Range Identification and Tracking (LRIT), another self-reporting system, typically produces time gaps of several hours duration. The present paper is concerned with what to do when interesting events take place in the wider time intervals, those with widths measured in hours.

1.1 Example Scenario

The typical scenario is illustrated here with an example reproduced from Peters and Hammond, 2011. Figure 1 shows the geography, in arbitrary units. The five green polygons represent islands and the two red dots represent ports. Two successive passes of a satellite in low earth orbit collect AIS data. The first pass (at time t = 0) finds three vessels at the locations marked with black dots labelled “A(1)”, “A(2)”, and “A(3)”; and the second pass 12 hours later (at time t = 1) finds the same three vessels at the locations respectively marked with black dots labelled “B(1)”, “B(2)”, and “B(3)”.

The filled orange circle represents a waste spill, which took place at time t = 0.34. The spill is of such a size that the vessel generating it should have been carrying and using AIS. We want to know which of the vessels was most probably within that circle at that time.

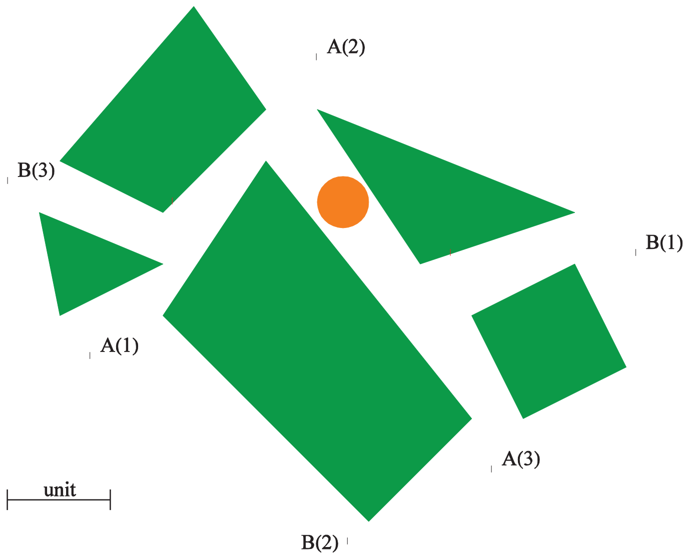

Figure 1: Representation of geographical data for the scenario reproduced from Peters and Hammond (2011). The five green polygons represent land masses; the two small red dots represent ports; the orange circle represents the waste spill; the black dots labelled A(n) and B(n) are the AIS position reports (t=0 for A(n) and t=1 for B(n), n from 1 to 3 for the three suspect vessels).

The type of output that we seek in answer to the problem is illustrated in Figure 2, which shows probability density contours for the position of each vessel at the time of the spill. With the aid of a few supplementary computations (see Peters and Hammond, 2011), these contours allow the user to identify the most likely culprit of the three, which is Ship 1 (with probability 49%).

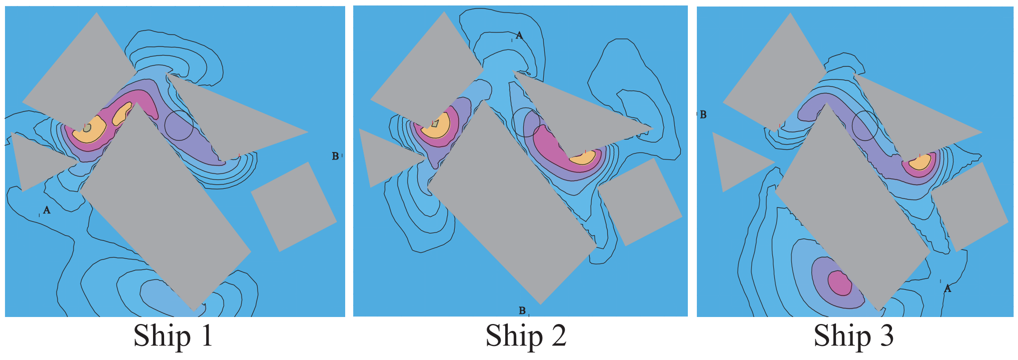

Figure 2: Probability density for the position of each vessel at the time of dumping, reproduced from Peters and Hammond (2011). The spill site is indicated with a black circle.

2. Approach

We seek to make probabilistic inference about the path each ship in the area took between its two most relevant position reports (*A* and *B*). These two reports are the ones that most tightly enclose the event of interest. Thus, this paper suggests a probability measure over the space of possible trajectories connecting *A* to *B*. It also illustrates a practical method by which probabilistic inference can be made over such a space. The Bayesian approach advocated here was first described in Hammond and Peters, 2009 and refined in Peters and Hammond, 2011. It involves generating a large set of tracks connecting *A* to *B* from a prior distribution over track space. Given relevant data (including negative data), the randomly generated tracks are then weighted by their likelihoods. Peters and Hammond (2011) illustrated how probabilistic responses to nearly any imaginable query follow straightforwardly.

The prior distribution used in Peters and Hammond (2011) is parameterized by the maximum and preferred speed of the ships ($v_{\max}$ and v_{pref}), by the minimum, peak and mean duration of port stays ($s_{\min}$, s_{peak} and, s_{mean}, respectively), by an index of each ship's tendency to visit ports rather than sailing on (the 'land lust' *L*), by a flag that permits or forbids immediate revisits to the same port (*repeat_allowed*), and by the minimum time interval over which paths are considered random ($t_{\min}$). Of these parameters, the maximum speed of the vessel $v_{\max}$ is the most important.

This paper introduces a new consideration into problems like the one illustrated in Figure 1, the presence of ice fields of the sort that impede navigation through polar waters. The main motive for this added consideration is the fact that the Northwest Passage is said to be increasingly passable to ships in summer, despite some remaining ice fields. Such ice fields are not always an absolute impediment to vessel progress, particularly for ships with ice-breaking capability. Indeed, while land masses generally impose a hard constraint of vessel motion, ice can be regarded as a soft constraint, one that may merely slow the vessel down. This paper suggests a means by which the ice-breaking capabilities of the vessels in an ice-infested area can be considered in interpolation problems.

Section 3, below, provides an overview of the existing track generation algorithm. Section 4 talks about how to identify shortest paths in the presence of ice. Section 5 points out that the ability to identify such shortest paths allows the method of Section 3 to be used to construct random tracks through ice fields. That section illustrates the new ice interpolator on a toy scenario, featuring a contact-to-track association problem. Benefits and limitations of the method are discussed in section 6.

3. Overview of Random Track Generation

This section contains a rough overview of the track generation algorithm presented in full in Peters and Hammond (2011).

Let *A* and *B* be the positions of the two reports between which we wish to interpolate the vessel's path, and let t_A and t_B denote the respective time stamps of those two reports. We want our track generation method to be capable of generating any physically feasible route that we might imagine. A simple way to accomplish this is by a recursive "divide and conquer" algorithm, in which a random intermediate point *X* is proposed for a random time t_X in the interval between t_A and t_B. If a trip from *A* to *B* via *X* is infeasible, given the land masses in the way, then the random selection of *X* is repeated. The details on how one may determine whether or not the intermediate point *X* is feasible are presented in Annex A. The implementation there is based on Dijkstra's algorithm (Dijkstra, 1969). When a feasible point *X* is found, the "divide and conquer" algorithm is called recursively, once from *A* at time t_A to *X* at time t_X and once from *X* at time t_X to *B* at time t_B. If the time interval is shorter than some pre-specified minimum (t_{min}), the vessel is instead assumed to take the shortest path, at a constant speed – thus terminating the recursion.

We use the recursive algorithm described above whenever we know (or assume, or choose to dictate) that no ports were visited in the time interval in question. A more general version of the algorithm is used when one or more ports are available. We start by randomly setting a "sojourn time" *s* – that is, an amount of time to be spent (possibly) at a port along the way. Sometimes, as decided randomly based on a probability that derives from the 'land lust' parameter (*L*) and from the sojourn time, we ignore the ports and turn to the no-port "divide and conquer" method described in the previous paragraph. Otherwise, a port is chosen randomly, among all those that are feasible, with the relative probability for each port being based on the amount of flexibility in the time of arrival at that port. In other words, ships are deemed more likely to visit ports when there is enough slack time between reports *A* and *B* that the ship can spend a "desirable" time in port (desirable sojourn times are defined with parameters s_{min} and s_{peak}). The arrival time t_X is then chosen randomly. The generalized "divide and conquer" algorithm is then called twice recursively – once from A at time t_A to the chosen port at time t_X and once from that port at time $t_X + s$ to B at time t_B.

Finally, consider the case where the vessel is (moored) in port at the start or the end of the time interval. A "sojourn time" is generated in the same way as it would be for a port to be visited along the way, but then only a randomly chosen portion of that sojourn time is considered to belong to the time interval of interest (between t_A and t_B). The generalized "divide and conquer" algorithm, as described in the previous paragraph, is called for the truncated time interval. For example, if the vessel starts in port, and a sojourn time *s* is determined for that port stay, then a random time amount *u* – typically, but not always, taken as uniform from zero to *s* – is taken out of the time interval of interest. So in this example the generalized "divide and conquer" algorithm would be called from A at time $t_A + u$ to B at time t_B.

4. Shortest Paths through Ice Fields

This section describes how to generate the shortest route between two points through a field of hard and soft polygonal obstacles (where the soft obstacles are ice fields). The presentation here assumes the reader is familiar with the graph construction process in Annex A1. This section shows how to add vertices and edges from ice field polygons into the coastline graph discussed in that annex. We introduce a new parameter that is specific to both a given ship *s* and a given ice field *f*. The parameter is called the *icedelay.* It represents the amount of additional time it takes ship *s* to traverse a given distance, when travelling through ice field *f*. The *icedelay* is used to add additional 'length' (sometimes called 'cost') to the graph edges, when these edges are entirely in ice.

The process begins by taking an ice polygon, and introducing *n* vertices at regular intervals around the boundary, as illustrated in the first panel of Figure 3 below. These boundary vertices are connected to one another with edges in the coastline graph whenever it is possible to sail between them without leaving the ice or crossing land. The process is illustrated by the next panels of Figure 3. These edges are assigned a 'length' in the graph that is artificially inflated by multiplying the actual length by the *icedelay.*

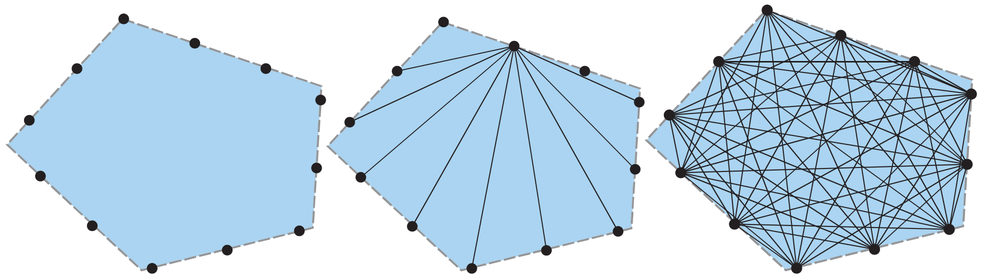

Figure 3: Three panels are used to illustrate the process of adding vertices and edges for an ice field to the coastline graph. The first panel shows the vertices introduced along the boundary of the field polygon. The middle panel shows how an individual boundary vertex is connected with graph edges to other boundary vertices, when the path to these is entirely in ice (and does not cross land). These edges have a length that is artificially inflated by the *icedelay* parameter. The right panel shows the final ice field sub graph- a clique in this case.

Figure 4, below, illustrates the process of adding the ice field sub-graph (Figure 3) to the main coastline graph produced by the process in Annex A1. Generally, an ice field vertex is connected to a coastal vertex (with an edge in the graph) whenever it is possible to sail strait between the two without crossing either land or ice. These connecting vertices have normal length, unmodified by the *icedelay*. There is one special case, however, that occurs when a coastal vertex is itself encased in ice. In that event, the coastal vertex in question is connected to those ice polygon vertices, and only those, which can be reached by travelling strait without crossing land and without leaving the ice. In the special case, the edge lengths are inflated by the *icedelay*.

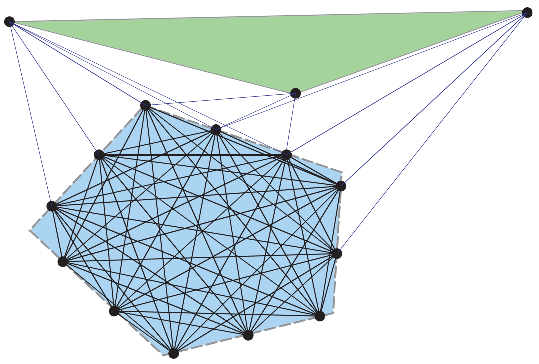

Figure 4: This figure illustrates the land in green and the ice field in blue. The ice field graph of Figure 3 is connected to the coastline graph (Annex A1) by adding an edge for each distinct (ice vertex, coast vertex) pair between which the ship can sail without crossing either ice or land. Those new edges are shown with dark blue lines.

Once the graph has been constructed, Dijkstra's algorithm allows for the computation of the shortest path between arbitrary points *A* and *B*. It suffices only to specify how to add these two points to the graph (see also Annex A2). There are two cases: the points are in ice or they are not. If either is not in ice, connect it (with edges) to all graph vertices that can be reached by crossing neither ice nor land. These edges have normal length. If either is in ice, connect it to those vertices, and only those, that can be reached by travelling entirely through ice without crossing land, inflating the edge length by the *icedelay*. Once this is done, Dijkstra's algorithm can find the shortest route from *A* to *B*, up to the precision induced by *n,* the number of ice field boundary vertices.

The shortest paths through ice fields exhibit an interesting refraction effect (mimicking the bending of light rays in a prism), an effect that gets more pronounced the greater the *icedelay.* This property is illustrated in Figure 5. To minimize travel time, it pays to turn the ship slightly on entering the ice field, so as to cross the ice faster.

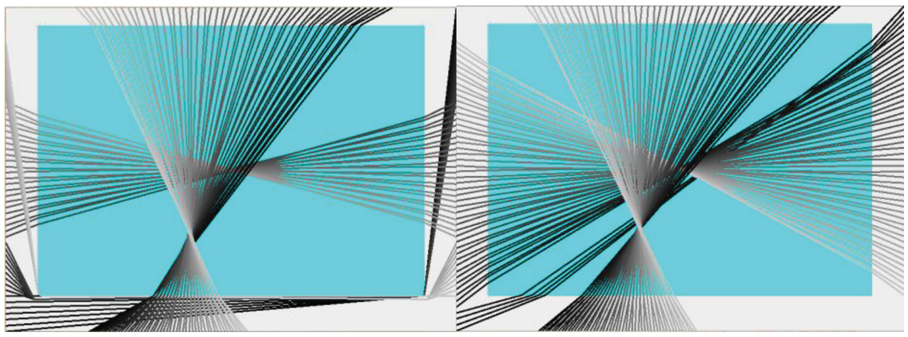

Figure 5: This figure has two panels, each showing 100 shortest paths (50 paths left-to-right and 50 paths top-to-bottom) through a rectangular ice field, which is shown in light blue. 400 vertices were placed on the ice field boundary in graph construction. In the left panel, the *icedelay* parameter is 1.5, so that it sometimes pays to go around the ice field rather than traversing it. In the right panel, the *icedelay* is only 1.2.

5. An Ice Interpolator Scenario

The ability to find shortest paths through ice, illustrated in the previous section, implies that the random track generation method (Section 3) can now be applied to polar waters. The only new consideration is the *icedelay*. Recall that this parameter was specific to a ship. As a result, each ship in the area of interest will have to have its own obstacle-avoidance graph. Otherwise, the algorithm works as before.

Figure 6 shows the ice interpolator in action, generating random tracks (gray lines) between 3 pairs of points (points are in red, with one pair per panel) on the same coastline shown in Figure 1, albeit with different ports (red A, B, C) and the addition of an ice field (shown in light blue). 40 vertices were introduced on the boundary of that ice field (see Figure 3).

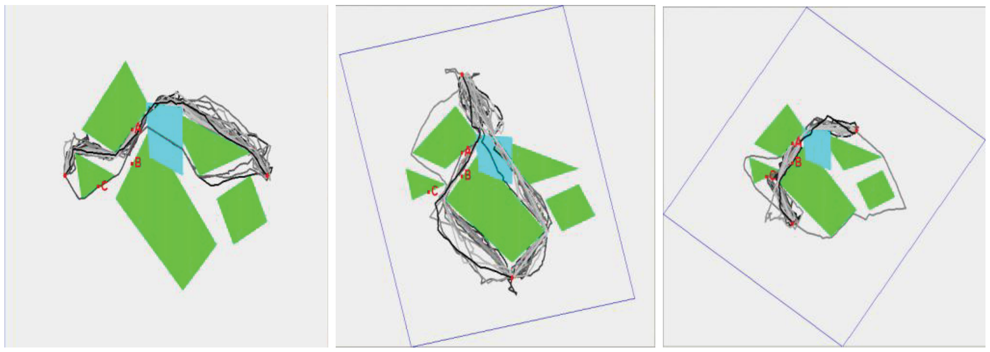

Figure 6: This figure has three panels, each showing 50 randomly generated tracks (shown in shades of gray) between a pair of reports (in red). The land is shown in green, and an ice field polygon is shown in light blue. Port locations are shown with red points, labelled A, B, C.

In Figure 6, the following parameter values were used (see section 4 and Peters and Hammond, 2011): v_{max} = 20 knots, v_{pref} = 15 knots, s_{min} = 1 hour, s_{peak} = 4 hours, s_{mean} = 5 hours, repeat_allowed = FALSE, L = 1, t_{min} = 30 minutes, $N_{frustration}$ = 20. The reports were 24 hours apart. For the ship in the middle panel of that figure, the *icedelay* was 1.2, while for the others it was 1.8. Thus, the middle ship was more capable than the others at breaking through the ice field shown. To give a better sense of scale in Figure 6, the coordinates of the small triangular island on the left were (1.0, 1.0), (0.8, 2.0) and (2.0, 1.5). Each unit represented 50 nautical miles.

Let us now add a slight twist to the scenario depicted in Figure 6. After 12 hours, a radar-equipped satellite is overhead and makes a ship detection that happens to lie inside the ice field. Which ship is it most likely to be? The answer is suggested in Figures 7 to 9, which show probability density contour maps for the location of each ship at the time. The contour lines in those figure were computed using the `bkde2D` function in the `KernSmooth` package in R (with smoothing parameter 0.15), so they are kernel density estimates. The location of the ice field contact is shown as a red x. The ice interpolator suggests that the most likely candidates are ship 1 and ship 2 (76% and 23% probability respectively leaving less than 1% for ship 3 (this last vessel was probably lurking much nearer the ports at the time). Ship 3 also has more slack time than the others, and so is better able to spend a desirable amount of time in port. In other words, the ports are more attractive for Ship 3 (see Section 3), whereas the other ships are more “pressed for time”.

These probabilities are computed by comparing the relative heights of the density surface (Figures 7-9) at the radar contact location.

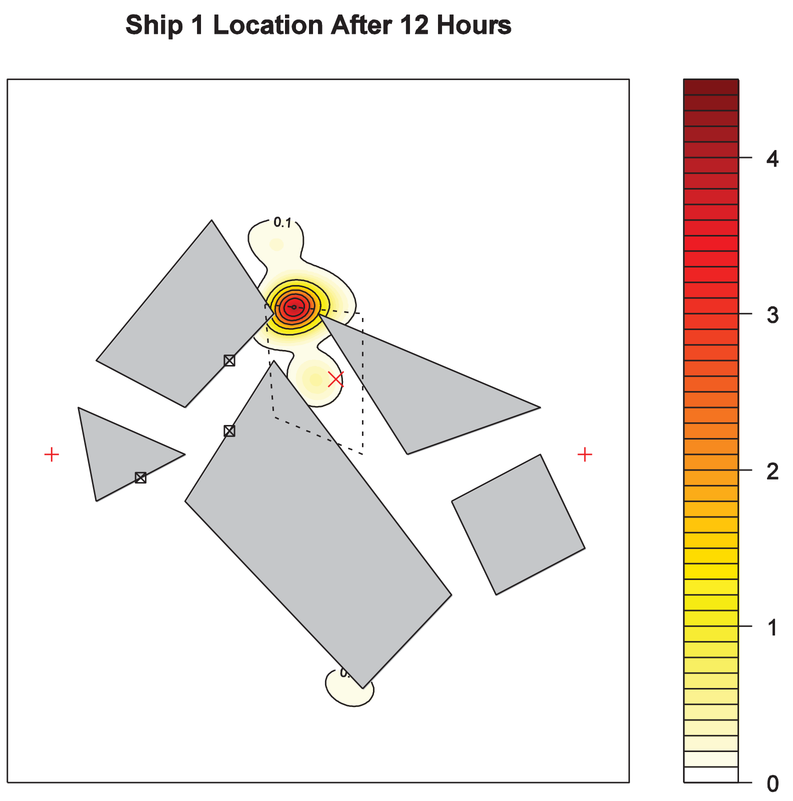

Figure 7: This figure shows density contour lines for the location of ship 1 (from Figure 6) after 12 hours. The land is shown in grey, and the ice field polygon is outlined with a dashed line. Port locations are shown with crossed black boxes. Red crosses show the start and end positions of the ship and a red x in the ice shows the radar contact position.

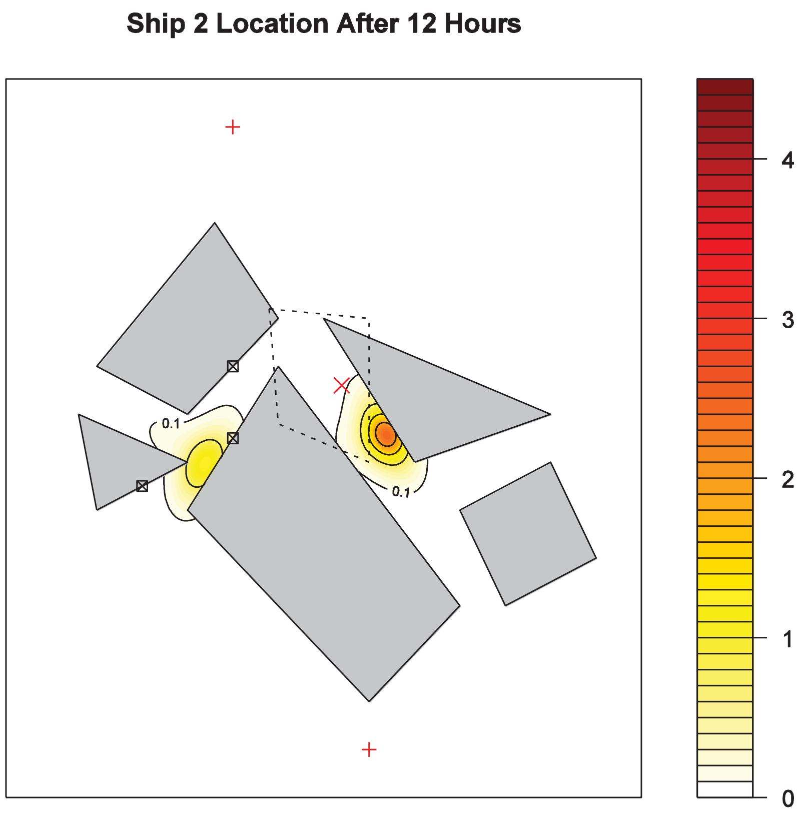

Figure 8: This figure shows density contour lines for the location of ship 2 (from Figure 6) after 12 hours. The land is shown in grey, and the ice field polygon is outlined with a dashed line. Port locations are shown with crossed black boxes. Red crosses show the start and end positions of the ship and a red x in the ice shows the radar contact position.

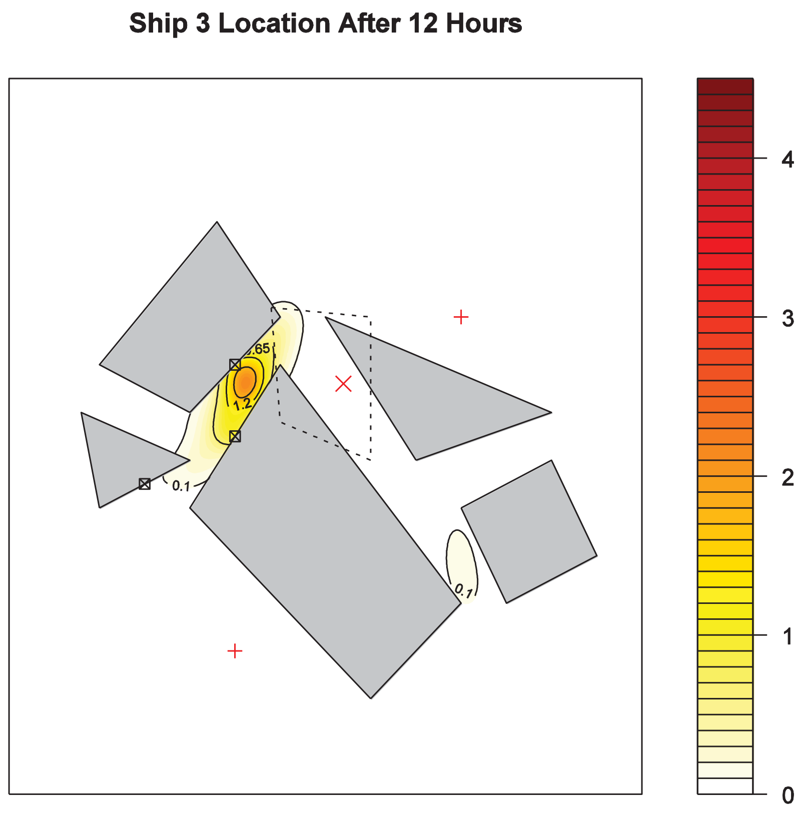

Figure 9: This figure shows density contour lines for the location of ship 3 (from Figure 6) after 12 hours. The land is shown in grey, and the ice field polygon is outlined with a dashed line. Port locations are shown with crossed black boxes. Red crosses show the start and end positions of the ship and a red x in the ice shows the radar contact position.

5. Discussion

This paper took an existing general procedure for enabling probabilistic inference over track space and modified it for use in ice-infested waters. The paper also suggested, using a fictitious example, the relevance of such queries to practical maritime situational awareness questions from oil-spill investigation, to maritime security and fisheries management. It must be emphasized that the procedure provides capabilities presently afforded by no other method. Some of the key assumptions and limitations of this approach have already been touched on in the discussion in Peters and Hammond (2011), and will not be repeated here. Instead, the reader may wish to consider an issue which that paper glossed over: the fact that graph construction is really slow.

Annex A3 reveals that finding the shortest path through a field of polygonal obstacles is not a speedy process, and that the slow part is graph construction. The figures in that annex do much to explain why the example applications shown here and in Peters and Hammond (2011) considered only simple, fictitious coastlines. If the ideas in this paper are to be applied to realistic coastlines, something will have to be done to increase the speed. We believe that it is possible to improve speed dramatically by relaxing the requirement to find the absolutely shortest path between two points. If we are willing to accept a path that is, in some useful sense, nearly the shortest path, then we should be able to improve speed by breaking the obstacle avoidance graphs into smaller pieces. This, in a nutshell, is the direction we plan to take with this work in future.

References

Devijver, P. A. and Kittler, J. (1982). Pattern Recognition: A Statistical Approach. Prentice-Hall.

Dijkstra, E. (1959). A note on two problems in connection with graphs. Numerische Mathematik, 1, 269-271.

Gelman, A., Carlin, J., Stern, H. and Rubin, D. (1995). Bayesian Data Analysis. London: Chapman and Hall.

Hammond, T., McIntyre, M., Chapman, D. and Lapinski, L. (2006). The implications of self-reporting systems for maritime domain awareness. 11th ICCRTS Symposium: Command and Control in the Networked Era, Cambridge, UK.

Hammond, T. and Peters, D. J. (2009). Probabilistic interpolation between position reports. NATO Workshop on Data Fusion and Anomaly Detection for Maritime Situational Awareness, La Spezia, Italy.

Hammond, T.R. and Peters, D.J., (2012) Estimating AIS Coverage from Received Transmissions. Journal of Navigation 2012. 65: p. 409–425.

Peters, D.J. and Hammond, T.R., (2011) Interpolation Between AIS Reports: Probabilistic Inferences Over Vessel Path Space. Journal of Navigation. 2011. 64: p. 595–607.

Nilsson, N. J. (1980). Principles of Artificial Intelligence. Palo Alto, CA: Tioga Publishing Company.

A. Shortest Paths through Polygonal Obstacles

This annex illustrates how this paper uses Dijkstra's algorithm to find the shortest route between two given points in the presence of polygonal obstacles (land masses). The method has two parts: converting a chart to a graph (described in A1) and using the graph to find the shortest route between two given points (described in A2). Section A3 gives the reader some insight into how long it might take to perform the operations identified in A1 and A2. It is not necessary to read A3 to follow the main ideas of the current paper. For that, the key thing is to understand what is meant by a 'feasible' intermediate point.

Given two position reports, A and B, an intermediate point X is considered feasible, if X can be reached from A and B can be reached from X without exceeding the assumed maximum speed of the ship (v_{max}). To verify whether X is feasible, it is sufficient to compute the shortest path between A and X and the shortest path between X and B. If both paths can be traversed in the time interval between A and B, then X is feasible.

A.1 Converting a Chart to a Graph

Start with a chart, which is a collection of polygons that separate the water from the land, as illustrated in Figure A1.

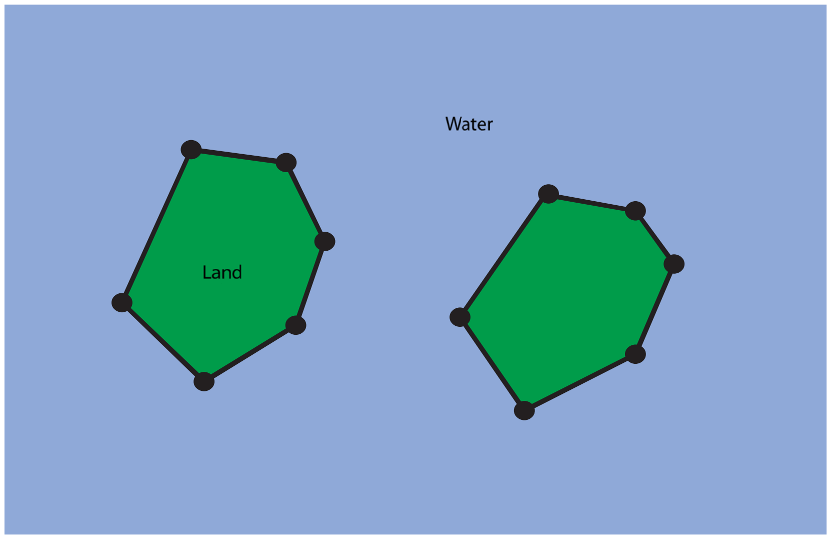

Figure A1: This figure provides an example of a simple set of polygons that divide water from land.

Construct a graph, which is a collection of vertices V (nodes) and edges E (line segments connecting the vertices, with an associated length (sometimes called cost)). The vertices of the graph will be the vertices of the coastline polygons. Connect the vertices of the graph with edges whenever it is possible to sail straight from one vertex to the other without crossing land. The graph that results from the coastline in Figure A1 is shown in Figure A2.

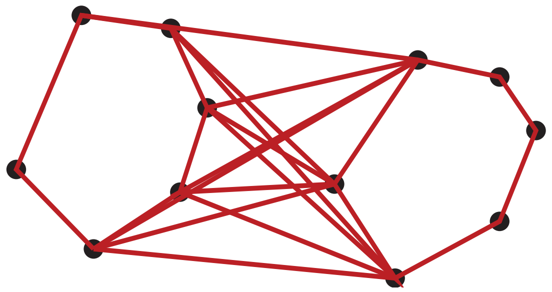

Figure A2: This figure illustrates the final graph produced by connecting all the coastline vertices of Figure A1 between which ships can sail without crossing land.

A.2 Determining Shortest Paths Using the Graph

This section describes how to use the graph, constructed in section A1 above, to find the shortest route between two given points *A* and *B*. We illustrate the general method with a representative example. Suppose that the positions of *A* and *B* are as illustrated (with pink dots) in Figure A3.

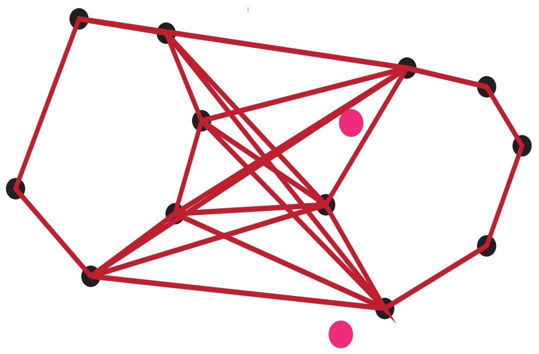
Figure A3: This figure adds two points (*A* and *B*- shown in pink) to the coastline graph of Figure A2.

The first step is always to check whether a ship could go straight between *A* and *B* without crossing land. If so, then this is the shortest route. In the case illustrated in Figure A3, the strait path is obstructed, so there is more work to do. We add *A* and *B* to the vertices of the graph, connecting these vertices to the rest of the graph with edges, as if they had been vertices of the original polygons. In other words, we connect each to all the vertices that can be reached by sailing straight. After connecting the pink dots, the graph looks like Figure A4.

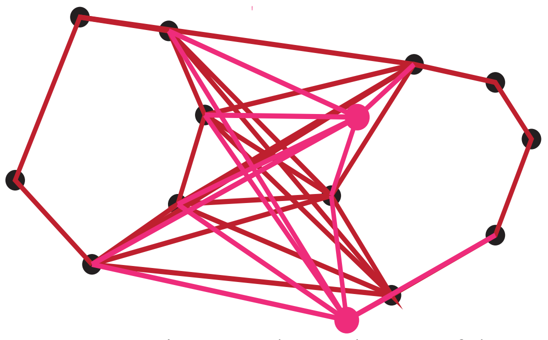
Figure A4: This figure connects points *A* and *B* to the rest of the graph.

Now we don't need the chart anymore: the shortest path connecting *A* to *B* must be made up of edges of this graph (Dijkstra, 1969). In fact, you can find the shortest path by Dijkstra's algorithm in O(|E|+|V|log|V|), where |E| is the number of edges and |V| is the number of vertices in the graph. The A* algorithm (Nilsson, 1980) is even faster.

A.2 Speed Considerations

In the process outlined above, the time-consuming part is graph construction, the part in section A.1. The good news is that this need only be done once for a given chart, and can be done in advance of shortest path finding. The task is also readily divisible between several processors. The bad news is that this is very time consuming indeed. The purpose of this section is to provide the reader with fair warning.

The section defines a sequence of gradually increasing coastline complexity and computes the graph construction time and the time needed to compute 100 benchmark shortest paths, for each entry in this sequence. The sequence of coastlines used and the benchmark paths are illustrated in Figure A5. Each entry in the sequence was constructed from its predecessor by taking each coastline edge and introducing a new coastline vertex randomly along the perpendicular bisector of the edge (the length of the perpendicular bisector segment was 20% of the length of the original edge). In other words, each entry in the sequence has twice as many coastline vertices as the previous one, and we could carry on creating more and more complex coastline fractals, to the limits of computer memory.

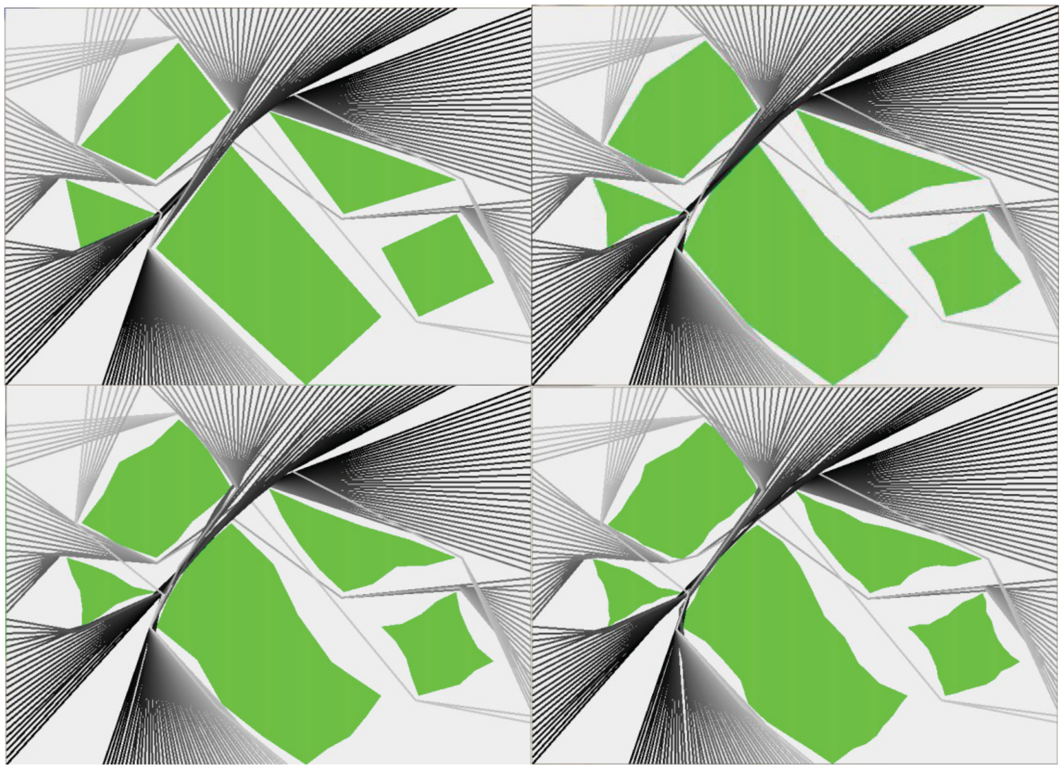
Figure A5: A sequence of coastlines (green polygons) of increasing complexity is shown in four panels, with the simplest coastline at top left and most complex at bottom right. 100 shortest paths are computed in each panel, 50 from top to bottom and 50 from left to right.

Figure A6 gives the time required to construct the coastal graph as a function of the number of coastline vertices in the sequence defined above. Note the logarithmic scale on the vertical axis. The long run times occur despite a number of speed enhancements and tricks that were not documented above, in the interest of brevity. Once the graph is constructed, however, the news is much better. Figure A7 gives the time required to find 100 shortest paths as a function of coastline complexity. In that figure, the run time is almost linear in the number of vertices.

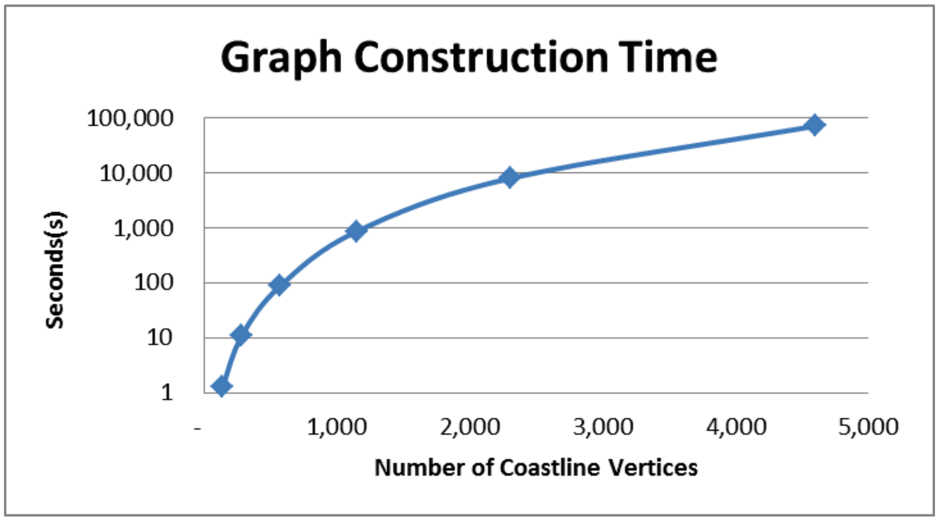

Figure A6: The time needed to construct the coastline graph is shown as a function of the number of coastline vertices in the sequence of coastlines of Figure A5.

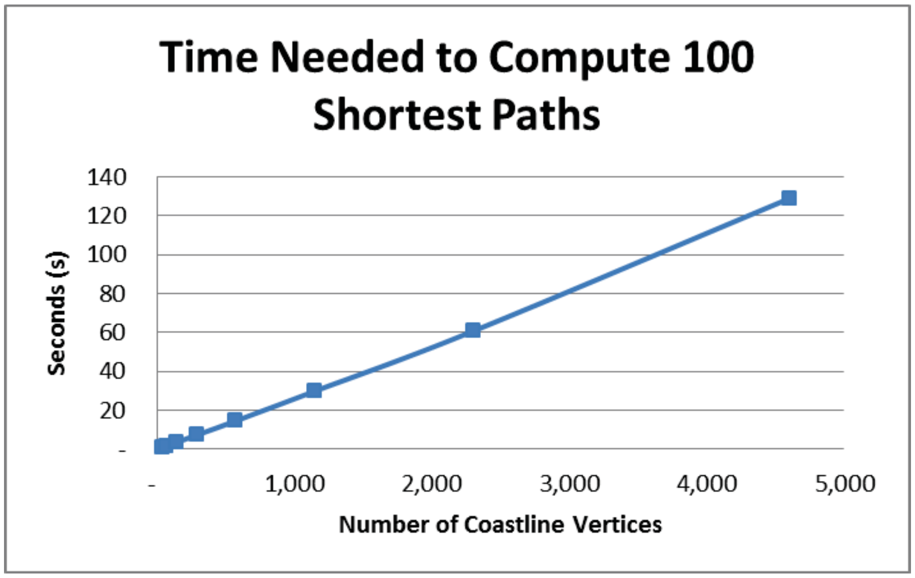

Figure A7: The time needed to construct the coastline graph is shown as a function of the number of coastline vertices in the sequence of coastlines of Figure A5.