Journal of Artificial Intelligence Research 82 (2025) 1845-1907 Submitted 06/2024; published 03/2025

Path-Planning on a Spherical Surface with Disturbances and Exclusion Zones

Jonathan Smith JONSMI@BAS.AC.UK
Samuel Hall SAMHALL@BAS.AC.UK
George Coombs GECOOMB@BAS.AC.UK
Harrison Abbot HABBOT@BAS.AC.UK
Ayat Fekry AYKRY@BAS.AC.UK
Michael A. S. Thorne MIOR@BAS.AC.UK
British Antarctic Survey, Cambridge, UK

Derek Long DEREK.LONG@KCL.AC.UK
King's College London, Strand, London WC2R 2LS, UK

Maria Fox MARIA.S.FOX@GMAIL.COM
British Antarctic Survey,
High Cross, Madingley Road, Cambridge CB3 0EX, UK

Abstract

An algorithm is presented for path-planning in a non-uniform spheroid mesh containing exclusion zones, vector and scalar fields. The mesh models physical environments such as ocean regions, together with a variety of environmental phenomena such as wind, current and ice conditions which impact on routing decisions. The path-planning method can be used to optimise the travel time of journeys between points in the mesh. We provide the algorithmic details and the mathematical foundations of the algorithms. To demonstrate that the method has basic desirable properties, we show that long paths in unconstrained regions of the mesh closely approximate great circle arcs. We go on to show that the method path-plans efficiently in environments with complex interacting conditions.

1. Introduction

In this paper we address the problem of path-planning for a vessel on a spherical surface with obstacles and an environment. The environment is modelled by a scalar field representing combined resistances and a vector field representing directional constraints on the progress of the vessel. The spherical surface and environmental features represent the surface of the Earth, including polar regions where challenging sea ice conditions apply. This work has application in path-planning for ships and autonomous marine platforms in a range of environmental conditions. The problem is a variant of the well-known Zermelo's navigation problem. Zermelo (1931) studied the problem of finding the fastest path for a vehicle navigating in a vector field on an uncluttered planar surface.

The path-planning method that we present is called Polar Route. Input to the system is a gridded model of a region within the global ocean aggregating a range of different environmental data sources that impact on path-planning for a given vessel specification. We first demonstrate the features and performance of Polar Route on randomly generated abstract models, then we present a number of real navigation examples. In order to show that the method is well-founded,

we demonstrate that paths generated in unconstrained environments are close approximations of the corresponding great circle arcs[1]. On top of satisfying this basic requirement, we further show that Polar Route generates efficient plans in the presence of scalar and vector field environmental conditions including in situations in which obstacles are present.

As well as path-planning in the presence of fields and exclusion zones, one of the main technical contributions of this paper is the approach we present to overcoming the limitations of mesh-based planning. Polar Route features a novel smoothing method that can be used to refine and improve the efficiency of a grid-based path produced by any of the well-known shortest path methods.

The paper is organised as follows. Section 2 specifies the problem being addressed. Section 3 motivates the problem and highlights some of the challenges that shape the solution we propose. Section 4 gives an overview of the route planning method used by Polar Route. Section 5 provides an overview of related work in path planning, particularly in the context of environmental features and on spherical surfaces. Section 6 describes Polar Route in detail, including the mathematical derivation of the solution to the fundamental problem: that of finding the fastest path across a boundary between areas with two different prevailing environmental conditions. We show how this problem extends to the spherical surface and explain how Polar Route exploits this in constructing the final path. Section 7 presents an evaluation of the performance of Polar Route on both synthetic and natural data sets, and a comparison between Polar Route and a Probabilistic Road Maps method (PRMs) (Kavraki & Latombe, 1998) in an example featuring obstacles but no fields. PRMs are widely used for path planning in robotics so it is of interest to understand how they can be used to address our problem. Section 8 extends our evaluation to compare with IcePathFinder (Lehtola, Montewka, Goerlandt, Guinness, & Lensu, 2019), a system designed for maritime navigation in icy waters. Finally, Section 9 concludes the paper.

2. Problem Definition

In this section we specify the form of the navigation problem solved by Polar Route. Informally, the problem is to find the fastest path for a vessel moving between two waypoints, through an environment on the surface of a sphere. The vessel is modelled as a particle moving under its own power, with no momentum or constraints on turning behaviour, and subject to the effects of the environment. A particle assumption is reasonable since the vehicle is insignificant in size at the scale considered, and the time to change velocity is also insignificant. The environment is modelled by a scalar field, which acts to constrain the maximum speed that the vessel can generate and a vector field that acts summatively on the velocity of the vessel moving in it.

2.1 The Environmental Mesh

The environment through which the vessel travels is an abstraction constructed using geospatial data. Suitable datasets are available from a variety of GIS services[2]. Surface sea ice concentration

1. A great circle arc between two points is the shorter segment of the circle defined by the intersection of the sphere and the plane containing its centre and the two points.
2. Geographical Information Systems Services include Copernicus, the National Snow and Ice Data Centre (NSDIC), the European Centre for Medium-range Weather Forecasting (ECMWF) and the General Bathymetric Chart of the Oceans (GEBCO).

and bathymetry[3] are examples of scalar fields, while surface currents and winds are examples of vector fields.

A dataset might consist of observations, or it might be a model-generated best fit to observations, providing a mean-state estimate. Datasets have temporal resolution as well, varying from days to months or years. Complex applications of a digital model of an environment require information about many environmental conditions, and the different datasets must be abstracted and combined in some way to provide a characterisation that is adequate for the intended application.

To characterise this data, we abstract the surface into tiles bounded by lines of constant latitude and longitude. The resulting grid of cells is called a mesh. Each cell in the mesh has a centre that is geo-located at a unique latitude and longitude, and contains a constant scalar field and constant vector field. These are arrived at by combining the temporally and spatially located data using a suitable aggregation method.

A subset of the cells are further subdivided into equally sized quarters in order to better characterise the details of the environmental features in areas of environmental complexity (for example, around land and on the edges of sea ice). Subdivision leads to a non-uniform mesh in which all cells have equal angular aspect ratio[4] This subdivision can be performed on cells arbitrarily deeply, although few subdivisions are needed for Polar Route to achieve a very high degree of accuracy in the great circle approximation (see Figure 1).

The mesh abstraction treats cells as rectangular, projecting the spherical segment onto the rectangular field. The width of each cell is approximated by taking the distance spanned by its longitudinal angle at the equator and multiplying it by the cosine of the latitude of the centre of the cell. Its height is the distance spanned by its latitudinal angle (along the great circle through the poles). This approximation is successful because land masses at the poles prevent access to cells in which the distortion of the projection would become prohibitive.

We use a deterministic modelling approach, in common with other long-distance navigation methods (Bast, Delling, Goldberg, Müller-Hannemann, Pajor, Sanders, Wagner, & Werneck, 2016; Lehtola et al., 2019; Bijlsma, 1975; Dijkstra, 1959; Kotovirta, Jalonen, Axell, Riska, & Berglund, 2009; Zermelo, 1931). The temporal and spatial scales of the navigation problem considered render stochastic path-planning infeasible. Our strategy is to plan using a snapshot of conditions, and replan as conditions change and are re-meshed.

The entire mesh can be conveniently captured as a quad-tree data structure (Finkel & Bentley, 1974). Cells are classified as entirely passable (albeit under different constraints) or entirely impassable. Since scalar and vector fields are constant within each cell, fields are discontinuous at the boundaries between cells.

As well as capturing and representing the information contained in each of the datasets, the meshing of datasets serves as a compression method, with the file size of resulting meshes being orders of magnitude smaller than the source data sets from which they were created.

Definition 1. *A* **mesh** *partitions a region of the environment into different-sized cells of equal angular aspect ratio, bounded by lines of constant latitude and constant longitude, in which each cell contains a constant scalar and constant vector field.*

3. A bathymetric dataset such as GEBCO provides high resolution measurements of sea bed elevation (and hence depth of water).
4. The angular aspect ratio is the ratio of the angles subtended by the horizontal and vertical edges of the cell.

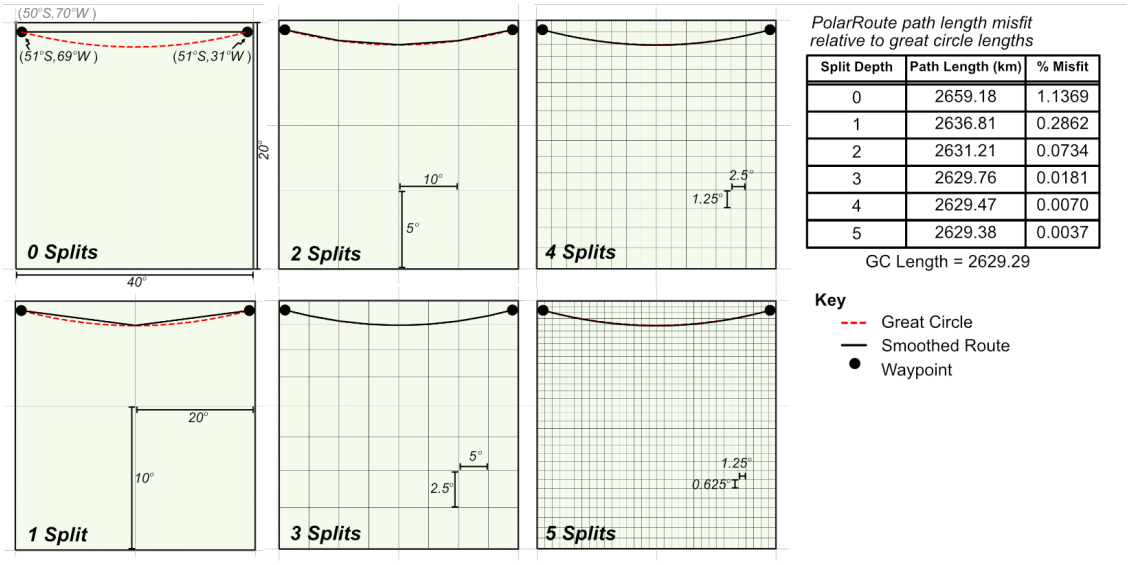

Figure 1: Polar Route plans a path between two waypoints that are 2629.29 km apart within a cell of dimensions 20° latitude by 40° longitude. In a single cell representation, the straight line path is 1.14% longer than the great circle computed on a sphere. One cell subdivision reduces this error to less than 0.3%. Three cell subdivisions are sufficient to reduce the cell sizes down to 2.5° latitude by 5° longitude and allow Polar Route to construct a great circle approximation to within an error of 0.02%. Five subdivisions reduce the cell sizes to 0.625° latitude by 1.25° longitude, and achieve an error within 0.004%.

Details of the construction of the mesh data structure are outside the scope of this paper. A mesh satisfying the properties described in Definition 1 can be arrived at in different ways. Polar Route receives such a mesh as an input, and generates paths that are constrained by its structure.

2.2 Properties of Paths

A vessel travelling across a cell travels with its velocity determined by the sum of the vector field effect in that cell and the vector of magnitude equal to its maximum speed (determined by the scalar field in the cell) at its chosen bearing. Note that the fastest path across a cell will always be found by using the maximum speed possible, according to the scalar and vector fields. A vessel travelling along the boundary between two cells is assumed to be affected by the more favourable conditions of the two adjacent cells.

Definition 2. *Cells are considered closed at the boundaries. Two cells, A and B, are* **adjacent** *if neither is blocked and their intersection is non-empty. They are diagonally adjacent if their intersection consists of a single point. The space inside a cell, C, (including its boundary) is referred to as $Space(C)$.*

In general, a path is a continuous trajectory from a start waypoint to an end waypoint, passing through cells in the mesh. A path in a single cell, C, can be modelled as a continuous parametric equation, $\mathscr{C} : [0, 1] \to Space(C)$, defining the points on the curve (which may be a straight line), in

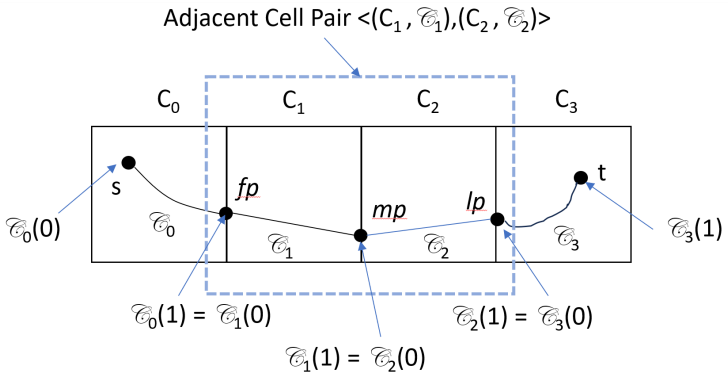

Figure 2: An example of a general path from s to t in a mesh, passing through four cells and consisting of three adjacent cell pairs. The points on the boundaries of the cells are the ends of the curves of the path in the two adjacent cells. The three significant points, fp, mp and lp, are shown for the adjacent cell pair C_1 and C_2.

C as a function of a parameter $x \in [0, 1]$. The curve connects the points $\mathscr{C}(0)$ and $\mathscr{C}(1)$, which may lie inside C or on its boundary (see Figure 2). Although Definition 3 describes the fully general form of arbitrary continuous paths in a mesh, Polar Route only constructs paths in which the curves are straight lines.

Definition 3. *A* **path**, P, *in a mesh,* $\mathcal{M}$, *between start waypoint,* s, *and end waypoint,* t, *is a sequence of pairs,* $(C_i, \mathscr{C}_i)$, $i = 0, .., n$, *of cells and curves in those cells, such that* $\mathscr{C}_0(0) = s$, $\mathscr{C}_n(1) = t$ *and, for each* $i = 1, .., n$, $\mathscr{C}_{i-1}(1) = \mathscr{C}_i(0)$.

The final clause of Definition 3 ensures that the path is connected from one cell to the next.

Definition 4. *An* **adjacent cell pair** *in a path,* $P = \{(C_i, \mathscr{C}_i) : i = 0, .., n\}$ *is a pair of cells* (C_j, C_{j+1}), *and the first point,* fp, *mid-point,* mp, *and last point,* lp, *of the path in this adjacent cell pair are* $\mathscr{C}_j(0)$, $\mathscr{C}_j(1)$ *and* $\mathscr{C}_{j+1}(1)$, *respectively.*

Three types of adjacent cell pairs, defined in Definition 4, can arise: those in which all three points lie on cell boundaries (the most common type in a smoothed path), those in which either the fp or the lp lie inside the space of the corresponding cells (which only occurs at the start and end of a path) and, finally, those in which fp are lp are the specific cell centres of their respective cells (occurring in the grid-based paths constructed prior to smoothing).

The problem we address can now be formally stated as follows:

> INSTANCE: Two points, s and t on a spherical surface, a mesh $\mathcal{M}$ defining a grid-structured partition of the surface, assigning scalar, $S(C)$, and vector, $V(C)$, field effects to each cell $C \in \mathcal{M}$ in the mesh, and an error tolerance $\epsilon > 0$.
>
> QUESTION: Find a path, P, within $\mathcal{M}$ from s to t whose total travel time is ϵ-optimal among all paths from s to t in $\mathcal{M}$.

This format mirrors that used by Mitchell and Papadimitriou (1991) in their paper on a closely related problem. Their work is further discussed in Section 5.

3. Motivation

The shortest path between two points on a sphere is a great circle arc. Modern GPS-based methods can follow a great circle arc in equatorial waters, although these methods are much less reliable in areas closer to the poles. Global navigation continues to depend primarily on compass bearings. Ships follow rhumb lines[5], changing bearing at appropriate points in order to closely follow great circle arcs. This practice reduces reliance on continuous GPS coverage and also allows interventions to adjust course around local weather conditions or other obstacles. This motivates constructing paths as sequences of waypoints connected by rhumb lines, where a key challenge is the problem of finding the most efficient way to connect rhumb lines in the presence of obstacles and varying environmental conditions.

Figure 3 shows examples of the problem we are addressing, constructed by randomly generating the scalar and vector fields as Gaussian Mixture Models. These examples are displayed in Mercator[6] projections. In these horizontal and vertical path examples, the waypoints are in fixed horizontal positions and fixed vertical positions respectively. Fixing these positions makes it clear how the paths change under the different conditions. It can be seen that the paths in the vertical examples cross the 0 latitude boundary, showing that Polar Route correctly manages the reflection of the surface through the equator. In the absence of constraints, the shortest paths follow approximate great circle arcs, as can be seen in the top left of the vertical examples, and the top of the horizontal examples. When environmental conditions are present, modelled by Gaussians, the paths are distorted from great circle arcs as necessary to accommodate them.

Figures 4 and 5 show paths planned by Polar Route in real environmental settings in the Arctic and in equatorial waters. The Arctic example is shown in a northern polar stereographic projection[7]. These examples show that Polar Route proposes efficient ways to navigate modelled conditions across long distances, adhering to common navigation practice by following rhumb lines and changing bearings infrequently. The computational requirements are moderate so routes can be regenerated when environmental conditions change.

4. Methods

Polar Route plans in two stages. First, the mesh is abstracted into an *accessibility graph*, defined in Definition 5.

Definition 5. *An* **accessibility graph** *is defined as* $G = \langle V, E, w \rangle$*, where* V *is the set of cell centre points and* E *is the set of edges connecting adjacent pairs in* V*, with weights* $w(e) \in \mathbb{R}^+$ *for each edge* $e \in E$*.*

Dijkstra's algorithm (1959) (presented in Appendix E) is used to construct shortest paths, which we call *mesh-optimal*, in the accessibility graph. To avoid unnecessary computation we only

5. A rhumb line, also known as a loxodrome, is a path of constant bearing.
6. A Mercator projection of a convex curved surface is a projection in which lines of constant longitude run orthogonally to lines of constant latitude, forming a grid. The parallels of latitude are straight lines spaced increasingly far apart towards the poles. The projection is named after the cartographer Gerardus Mercator who first defined this projection in 1569.
7. A polar stereographic projection projects points on a sphere onto the plane from a point of perspective at the pole. Great circle arcs passing through the pole are then seen as straight lines on the projected surface. All other great circles project to circles, but arcs of great circles passing close to the pole are seen as approximately straight.

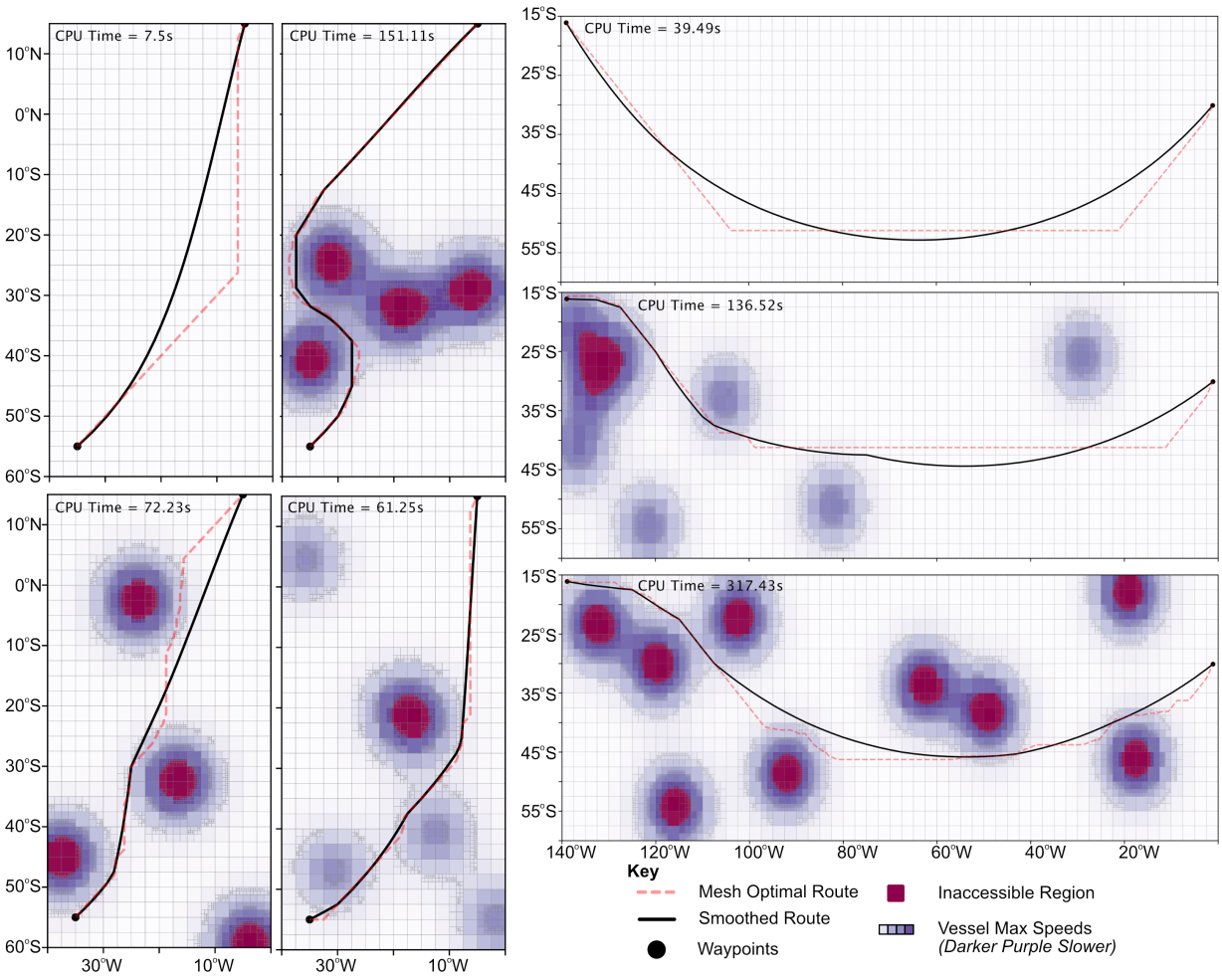

Figure 3: Examples of paths planned in Gaussian Mixture Model environments, showing the mesh-optimal paths (red dotted lines) and paths obtained after smoothing (in black). As can be seen from the axes, these are southern hemisphere and equatorial examples. Further analysis of these examples is included in Section 7.

evaluate edges as they are required by the Dijkstra search. Dijkstra's algorithm could be replaced by a version of A^* or a dual search from both the start and end points in the accessibility graph to achieve equivalent mesh-optimal paths.

The edge weights, w, are the fastest travel times between the centres of the adjacent cells in G. If the two cells model different environmental conditions, the shortest travel time will be the conjunction of two straight-line segments meeting at the boundary of the cells. The position of the crossing point on the boundary is the single degree of freedom in this non-linear optimisation problem. The impact of the position of the crossing point on the total travel time, in a variety of different current conditions, is shown in Figure 6. The graph shows the travel time between two points equally spaced across a boundary that is orthogonal to the straight line connecting them, so that, in the absence of a vector field, the shortest travel time is when the crossing point is at

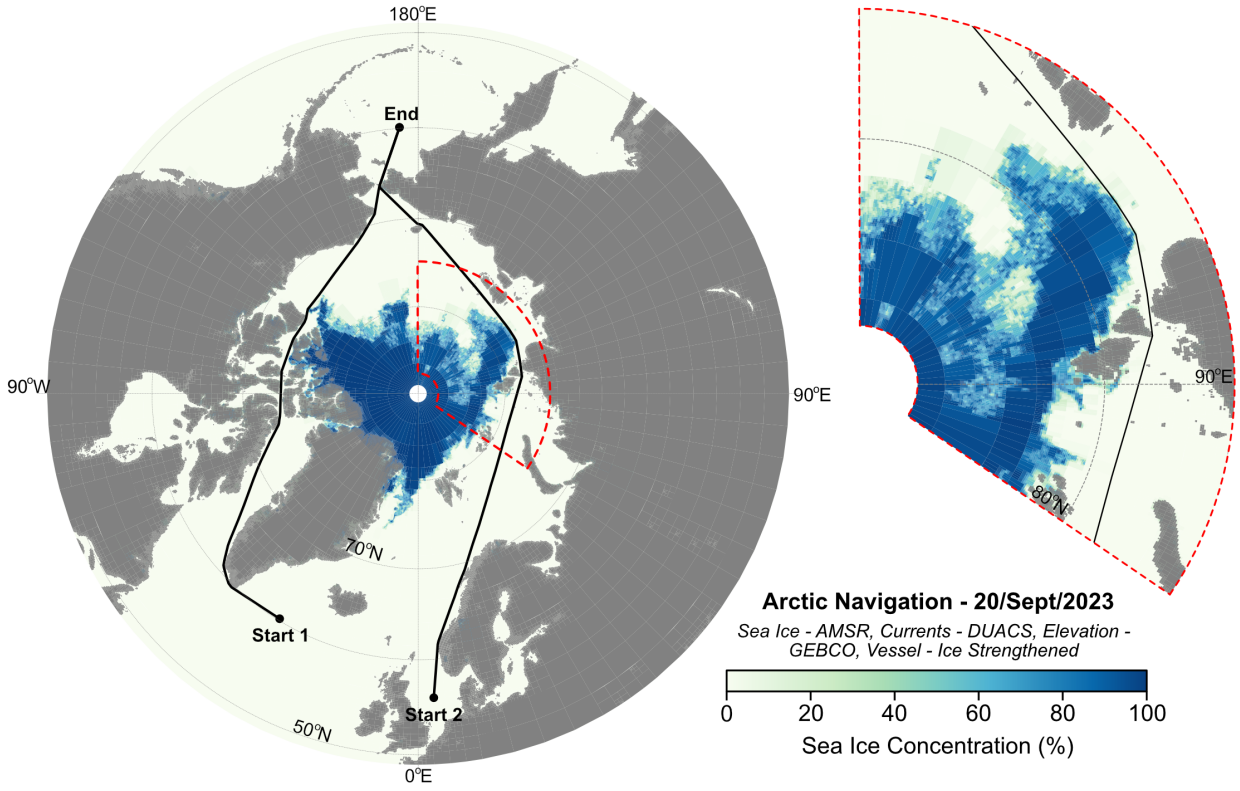

Figure 4: Paths planned in the Arctic using surface ice concentration data from September 2023. The red dotted area is expanded on the right, to show the mesh decomposition on the ice edge, and how Polar Route skirts around inaccessible cells.

0 displacement from the straight line between the points. As can be seen, travelling away from the line, to either side, increases the travel time. The curve is a quadratic (as we show in detail in Section 6.1.1) and, more precisely, a hyperbolic, in which the asymptotes are determined by the maximum speed on each side of the boundary.

Figure 6 shows that, when a vector field is present, the curve is displaced, so that the solution can vary between positive and negative displacements. In this example, the environment is modelled with currents and a scalar field fixing the maximum speed of the vehicle to be 20 in both cells (units are not important in this abstract example). The currents are described by pairs of vectors $(a, b); (c, d)$ for the two cells in the order of travel, with a and c the currents orthogonal to the crossing line and b and d oriented in the direction of the straight line from the starting point to the end point. In Section 6.1.1 we show the derivation of this curve as an implicit function of the displacement and explain how its minimum can be found numerically using the Newton-Raphson method.

Curvature affects the choice of crossing points. Figure 7 shows the effect of latitude on the choice of crossing point in the southern hemisphere. In this example, the travel time is a function of angular displacement from the origin at 63.75° S. The start and end points are both at this latitude, a constant scalar field of 20 is used and no vector field applies. The distance between the

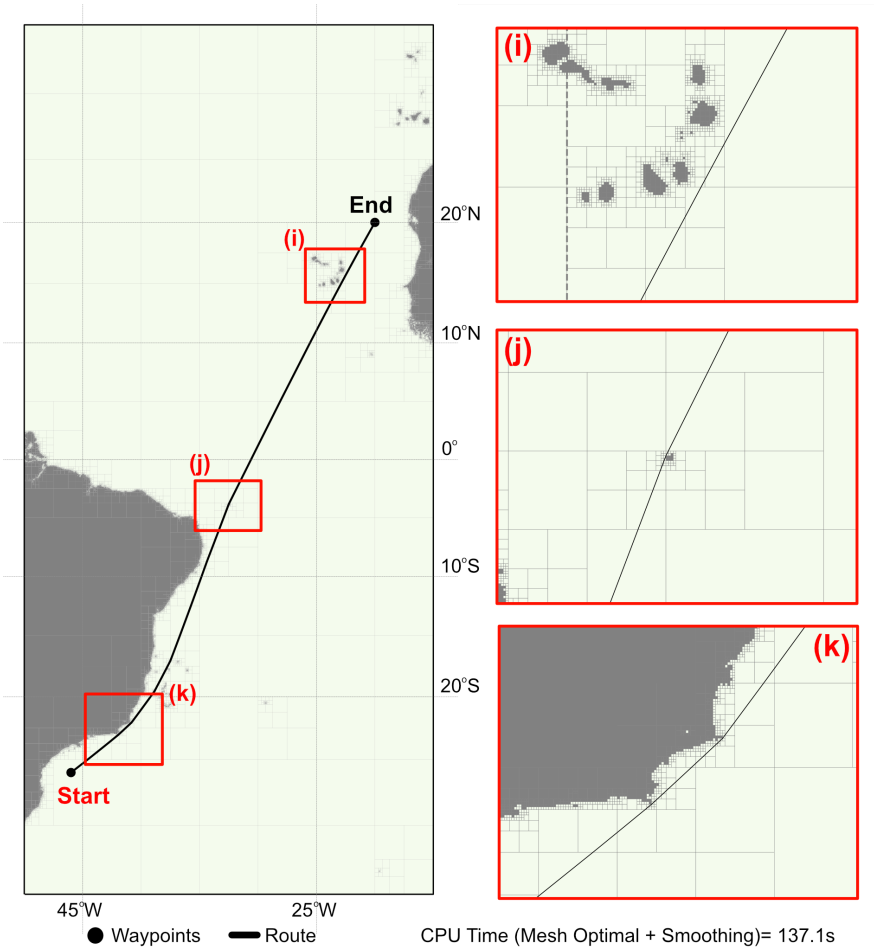

Figure 5: A planned path crossing the equator (at 0° latitude) from *Start* to *End*. The red boxes indicate expansions showing how Polar Route navigates around land cells and shallow bathymetry.

waypoints has been increased to show the curvature effect. The orange curve shows the travel times for the planar model, in which the fastest path crosses at zero displacement. The blue curve shows the effect on travel times of modelling the spherical surface. In this case, it is faster to cross at a latitude displacement of -0.31°. Clearly, similar results, but reflected, are obtained in a northern hemisphere example. Only at the equator will the planar and curved surface paths be the same.

The curve in Figure 7 is not a quadratic, because the choice of crossing point has a cosine-based effect on the scaling of distance. The details of the derivation of the implicit function determining travel time over the curved surface as a function of the choice of crossing point are given in Appendices C and D, and the solution can be found by an application of Newton-Raphson's method similar to that used in the planar case (the common structure makes it sensible to use a common solution framework for the two cases).

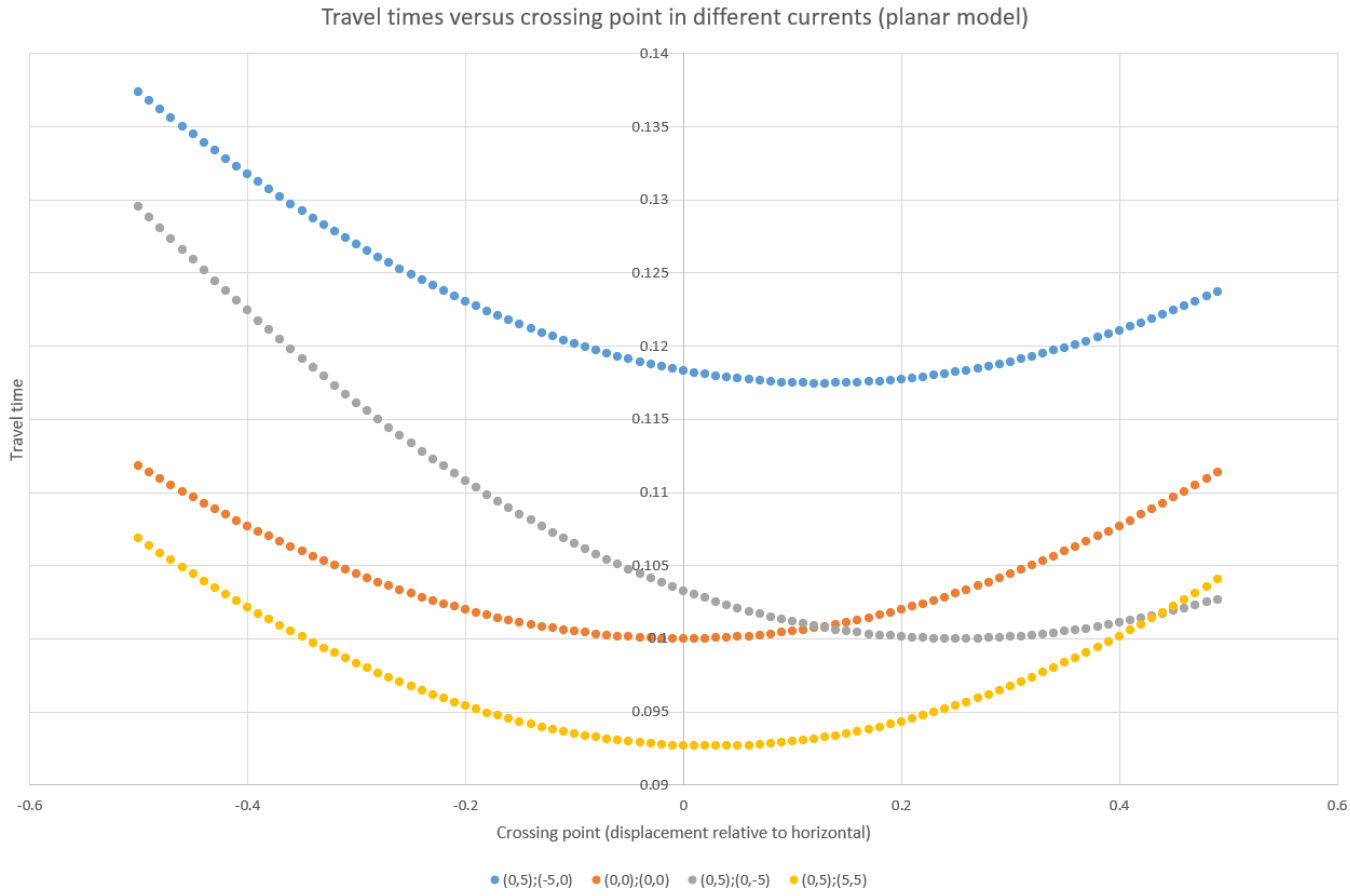

Figure 6: Travel times between two points at 63.75° S latitude under the effects of different currents (colour-coded). The x-axis shows the crossing point as displacement in degrees from the straight line between the two points. The y-axis shows the travel time at each of these displacements. The line through zero gives the optimal travel time in the absence of a vector field. The environment is modelled as a planar surface. The non-zero currents are all magnitude 5 northerly in the left cell and of different magnitudes and orientations on the right (as indicated by the legend).

The navigation problem requires not only that the path respects the vector and scalar fields in the cells it crosses, but also that it avoids cells that are blocked (by land or other exclusion zones). The compromise between the effects of curvature and the impact of blocked cells, as well as the graduated effects of the scalar and vector fields, makes the problem challenging. Finding shortest time paths through scalar fields has been explored, and the solution proposed by Mitchell and Papadimitriou (1991) illustrates the significance of the identification of the crossing points between boundaries of regions with different scalar characteristics. The combination of this problem with vector fields has received less attention. As discussed in Section 5, Garrido *et al.* (2020) do address scalar and vector fields in the context of marine navigation, but only with regard to short-distance

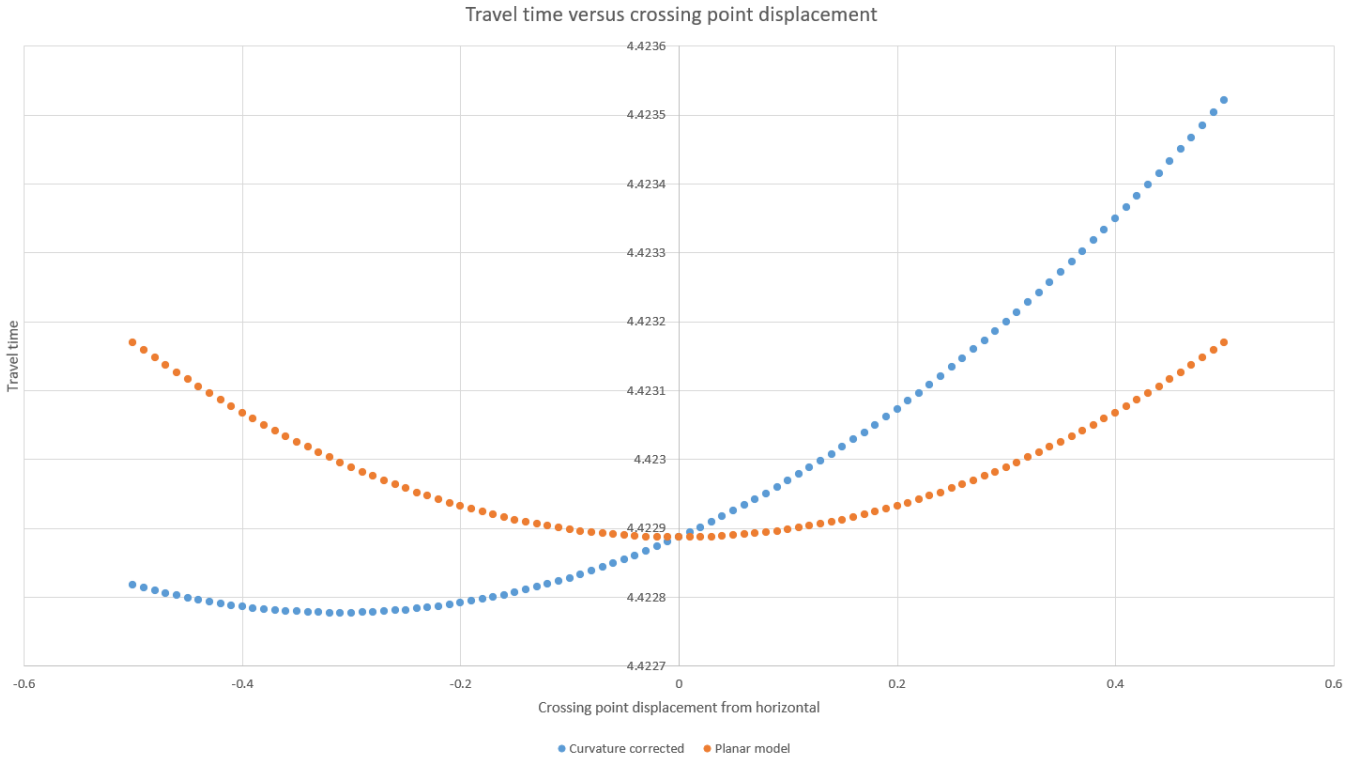

Figure 7: Travel times between two points at 63.75° S latitude, showing the effect of modelling the spherical surface. The orange curve is the optimal solution in the planar model. The blue curve shows that optimal travel time is achieved on the spherical surface by crossing at a -0.31° displacement.

navigation on a high resolution grid of a planar surface. The additional impact of using a spherical surface is most relevant in long-distance maritime or aerial navigation and this problem remains extremely challenging. We discuss some related work in this area in Section 5.

We therefore formulate the equations that determine the travel time for a given crossing point and then minimize this time numerically, using Newton's method to solve the corresponding minimisation problem. Special cases arise if the problem does not have a stable solution or if the solution falls outside the boundaries of the cells. If we face numerical instability, we attempt various recovery methods for the numerical process, and otherwise set the edge as infinite weight.

As can be seen in Figure 8, the paths in the two directions between two waypoints are asymmetric due to the directional effect of the vector field. The path from West to East exploits the strong Antarctic Circumpolar Current (ACC) flowing between the tip of South America and the Northern Peninsula of Antarctica. The East to West path avoids the ACC by travelling closer to the Northern Peninsula and exploiting the East Wind Drift Current. This example emphasises the importance of modelling the vector field as it has significant effects on path quality.

Smoothing iterates over the entire path until convergence is reached. This process refines a mesh-optimal path into one that follows the curvature of the sphere as closely as possible given the

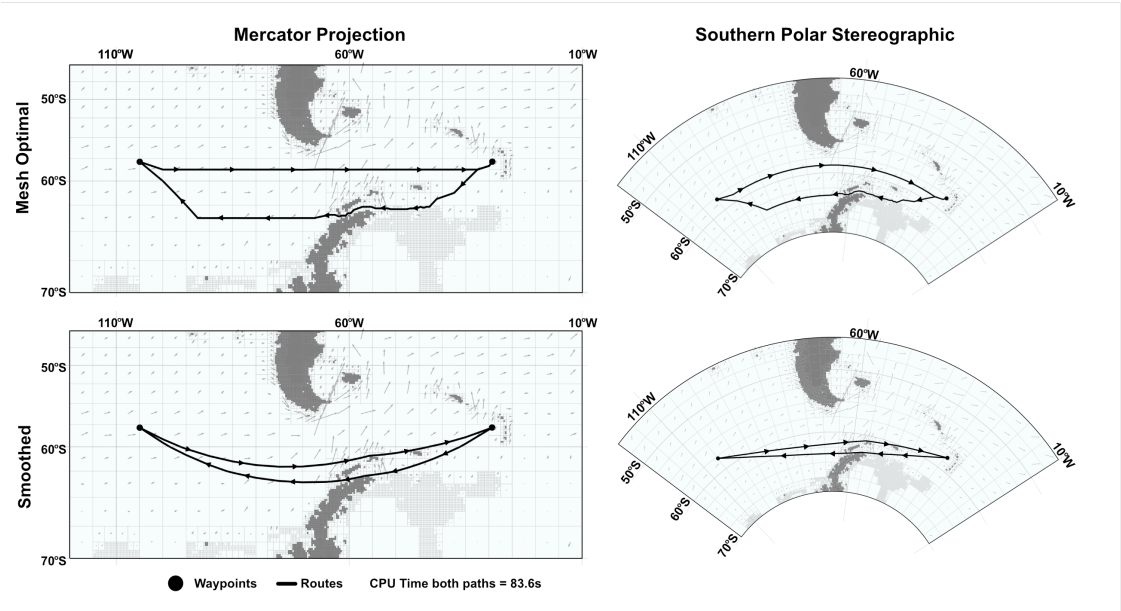

Figure 8: On the left, mesh-optimal and smoothed paths in a Mercator projection. On the right, the same paths in a southern polar stereographic projection. In all cases, the paths in both directions are shown, generated in an environment constrained by a scalar field representing land and sea ice, and a vector field representing currents. The vector field significantly increases the East to West travel times.

modelled disturbances and accessibility constraints. Figure 8 illustrates the impact of combining navigation on a sphere with the problem of obstacle avoidance and field effects, showing each of the two stages of the Polar Route method. The details of both the mesh-optimal path construction and the smoothing method are presented in Section 6.

4.1 Optimality

Polar Route constructs paths that represent a compromise between travel time and the number of bearing changes. The latter feature is important to navigators, but hard to evaluate versus travel time. The formal problem description in Section 2 is expressed only in terms of minimising travel time. The number of bearing changes on a path is dependent on the number of grid cells the path crosses, but the error between the great circle arc that crosses a cell, and the rhumb line distance across that cell, depends on the size of the cell, as shown in Figure 1. If we are willing to accept more bearing changes in order to reduce travel time, we can increase the number of cells to improve the accuracy of the paths. If cells are further subdivided in order to achieve this, the properties of the original cells remain unchanged. Subdivision can therefore be performed arbitrarily often in order to improve the path travel time.

The rhumb line distance converges approximately linearly towards great circle arc distance in the number of cell divisions. The number of cells crossed by the path approximately doubles for every subdivision. Thus, convergence to optimality is approximately logarithmic in the number of

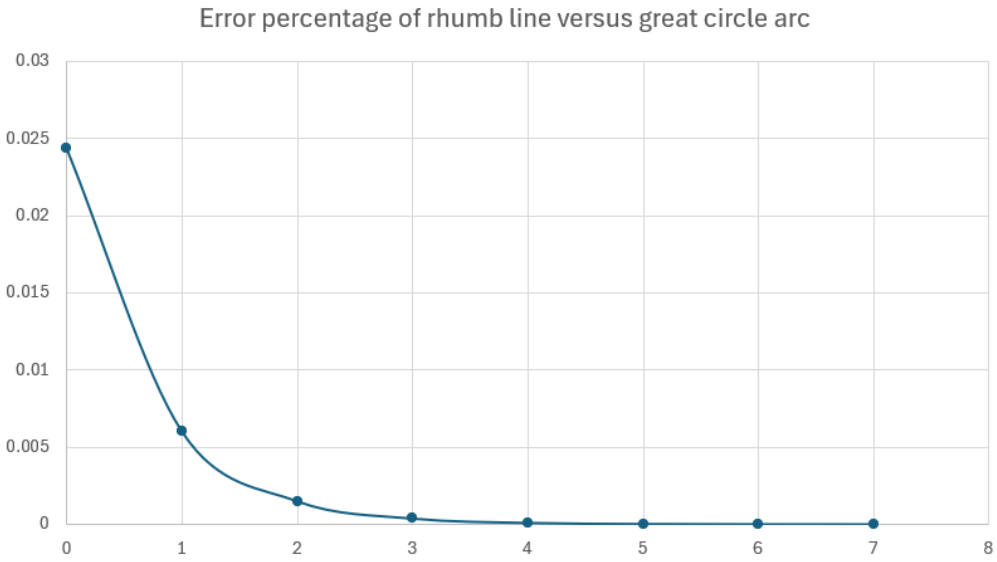

Figure 9: Percentage error in rhumb line versus great circle arc between points at 60° latitude, $\frac{2.5}{2^n}$° latitude and $\frac{5}{2^n}$° longitude apart, for increasing n.

cells on the path. Figure 9 shows the error between rhumb line distance and great circle arc for an example crossing (corner to corner) in a cell at 60° latitude. As can be seen, the error is less than 1 in 10,000 after just one subdivision.

The approach used in Polar Route has a singularity in its behaviour at the poles. As a consequence, paths that pass exactly through the poles cannot be constructed using this method. Behaviour close to the poles remains stable, but the algorithm for handling converging points at the boundaries of cells during smoothing does not apply if the polar cells are allowed to converge at a single point. The error in paths that should otherwise traverse the pole can be made arbitrarily small by using increasingly fine-grained subdivision of the cells around the pole, allowing their base edges to get arbitarily close to the pole without reaching it. Since marine navigation at the poles is not a real problem, we do not attempt any more specific solution to navigation in polar regions.

A more significant issue for optimality of Polar Route paths is the possibility that mesh optimal paths pass around obstacles on the opposite side relative to the optimal path in reality. In this case, our smoothing algorithm cannot lift the path across the obstacle. This situation can always be resolved by finer subdivision of the cells around the obstacle. More generally, the length of the mesh-optimal path also converges towards the optimal path length as the cells are subdivided.

5. Background

Path planning has been considered in a variety of environments, particularly in situations constrained by the presence of obstacles or the existence of predefined accessible paths (eg roads). Some researchers have explored the problem of robots moving through different terrains, taking into account the impact on their maximum speed according to the terrain conditions. For example, Mitchell and Papadimitriou (1991) found an efficient solution to path planning through planar

polygonally segmented regions, in which different polygons are assigned different weights representing the maximum speed of the vehicle in each region. They solve the following problem:

> INSTANCE: Two points, s and t in the plane, a finite triangulation γ in the plane, an assignment of weights, a_e and a_f, to edges and faces, and an error tolerance $\epsilon > 0$.
>
> QUESTION: Find a path within γ from s to t whose *weighted Euclidean length* is ϵ-optimal among all paths from s to t in γ.

The solution they explore allows paths to follow the edges forming boundaries between the regions, which are weighted separately from the faces. The authors employ a continuous Dijkstra's algorithm method, in which the paths maintained in the priority queue managed by the algorithm are extended by selecting intervals of reachable space, separated by *events* at which the paths cross boundaries or enter edges between faces.

While triangulations and other tilings, such as hexagonal, can be used to form a mesh, grids formed from rectilinear polygons are often used to represent terrain in different situations where path-planning is required. Methods for path-planning in rectilinear models find shortest paths under accessibility constraints requiring grid artifacts such as cell centres, corners or edges to be followed. These include Dijkstra's algorithm (1959) and variants of $A^\star$ search (Koenig, Likhachev, & Furcy, 2004; Koenig & Likhachev, 2002; Ferguson & Stentz, 2006). These constraints mean that shortest paths found by these methods are longer than the actual shortest paths in the environments being modelled. This was shown by Daniel *et al.* (2014), who show how $A^\star$ can be used to find an initial path that can then be "smoothed" by relaxing some of the grid-following components of the path and allowing grid edges to be crossed at any angle, to make those parts of the path shorter. In their approach, called Theta*, this is done by using lines of sight to replace a sequence of lines that visit grid points, as long as these lines of sight do not cross obstacles. The resulting paths have fewer changes of heading and are closer to optimal paths in reality. An alternative approach is considered by Rivera *et al.* (2020), who reduce the impact of the grid-based restrictions by considering larger neighbourhoods. This leads to a larger range of possible angles for the path segments and reduces the occurrence of grid artifacts in the paths. Both works were carried out on a planar surface and environmental factors, such as wind, were not considered. Rospotniuk and Small (2022) extend an any-angle approach due to Harabor and Grastien (2013), designed for Euclidean planes, to compute optimal paths on a spherical surface with obstacles. Their system, called Spherical Anya, performs an any-angle style search in a grid (based on an equirectangular projection), but uses great circle arcs to project the lines of sight. This leads to optimal paths as combinations of great circle arcs and boundary-following behaviour around obstacles. The authors do not consider scalar or vector fields.

5.1 Algorithmic Approaches

The Fast Marching Method (Sethian, 1996) uses a variant of Dijkstra's algorithm to find paths in a graph constructed on a fine-scaled mesh, using numeric methods to determine the travel time between points in the mesh. This approach allows both scalar and vector fields to be considered (Garrido et al., 2020; Garrido, Álvarez, & Moreno, 2016). Sethian (1996) has observed that the topology of the surfaces over which it is applied can include spherical or ellipsoidal surfaces. As Garrido *et al.* (2020) observe, the accuracy of their solutions is a compromise between the grid resolution and the computation time. Although the cost of their algorithm is $O(n)$ for a grid of

n points, this is dependent on using an implementation of Dijkstra's algorithm that exploits an *untidy priority queue* (Yatziv, Bartesaghi, & Sapiro, 2006). The same data structure could be used to speed up the performance of Dijkstra's algorithm in Polar Route, but the untidy priority queue introduces an approximation that can, in principle, lead to inaccurate outcomes.

The Fast Marching approach proposed by Garrido *et al.* (2020) involves first expanding a wave front from the source to identify obstacles, further propagating waves from the obstacles themselves, to identify costs with points on the grid that compromise between time-to-travel and distance to obstacles. Bijlsma (1975) uses a variant of Fast Marching in which the time increment is maintained, so that instead of calculating the time to reach positions, he calculates the positions reached after specific times. Petres *et al.* (2007) have also explored the use of Fast Marching to plan navigation in underwater environments on a local scale where spherical geometry is not relevant.

Other methods discretise the environment without imposing a regular grid structure. A detailed survey by Kavraki and LaValle (2016) highlights some of the main approaches that have been taken in path and motion planning for robotics. One of the most influential approaches is the Probabilistic Road Map (PRM) method (Kavraki & Latombe, 1998). This approach builds a graph representing the underlying space and the impact of the kinodynamic constraints on the robot moving in that space. It does this by using sampling to select waypoints in the space (filtering these to remove points that lie inside obstacles) and then connecting close neighbours by using simple edges that represent feasible solutions to the local kinodynamic problem of moving the robot from one configuration to the other. This approach has an important property: assuming that the sampling strategy has appropriate qualities, the shortest path between waypoints in the graph generated by this method will converge to optimal as the sampling density increases. By appropriately selecting the metric for determining the edges between waypoints, this approach can be applied to spherical surfaces. Sampling can be performed probabilistically or deterministically. LaValle *et al* (2004) discuss the impact of alternative strategies, including importance sampling, quasi-probabilistic and mixed deterministic-probabilistic sampling. The performance of a PRM approach on problems addressed by Polar Route is further discussed in Section 7.

5.2 Mathematical Methods

The navigation problem of finding the fastest path for a vessel navigating in a vector field on a surface, was first formalised by Zermelo (1931). He offered as an example the problem of navigating a ship in wind, which he considered on a planar surface. In a conference in 1929 he also proposed a version of the same problem with an airship moving in a 3-dimensional vector field. Zermelo and other mathematicians approached the problem through the calculus of variations. Certain cases are solvable analytically (for example, a vector field of constant magnitude and direction is easily solvable) and navigation on surfaces of revolution, such as a globe or an ellipsoid, have been explored (Bonnard, Cots, & Wembe, 2021; Aldea & Kopacz, 2020), with some simpler cases also being solvable analytically. In most cases, the problem can only be solved numerically. One way to approach path planning is as a control optimisation problem as it was posed by Zermelo (1931).

Variational methods treat navigation problems as continuous control problems, with solutions developed as continuous functions specifying control responses along the path of the vessel. For example, Bijlsma (1975, 2001) uses this approach to solve marine navigation problems in scalar fields, while Aldea and Kopacz (2020) use it to solve navigation on a spheroid surface in wind. In these methods, the theoretical underpinnings rely on assumptions that the fields are continuous

and differentiable. The solutions can then be found, in certain cases, using variational principles. The computational process is then to find values of these solutions, which is typically achieved numerically, by closely approximating the necessary derivatives using local gradient calculations and integration (to determine travel time, distance travelled or fuel consumed) by similar numerical methods, such as a simple predictor-corrector method.

Discretised data does not necessarily prevent the use of a continuous model. Bilinear interpolation can be used between discrete values, effectively turning the discontinuous problem into a continuous approximation which may be considered no worse an approximation of the underlying reality than is a discretised mesh. Nevertheless, the optimal control approaches rely on discretising the space once again in order to apply numerical methods and this discretisation is typically done at very high resolution. For example, Bijlsma (2001) uses a mesh with a resolution similar to the distance a ship can travel in 6 hours. This would be approximately 150 km[8] which is a little more than a degree of latitude. A uniform mesh at this resolution would introduce at least an order of magnitude more grid points than is typical in a mesh generated for Polar Route in which non-uniform sized cells greatly reduce the mesh complexity. For finer scaled navigation, such as that considered by Garrido *et al.* (2020), the very high resolution grids required would make it computationally very expensive to plan paths over long distances.

Post-processing paths to smooth them is a common stage in the construction of robot paths. Typically, this is to account for the kinematics of the robot and to overcome the discretised structure of the underlying graph used in the initial path construction. In Polar Route, smoothing accounts for the spherical surface over which the vessel is navigating. Smoothing paths using splines (Sprunk, 2008) has been used in robotics, where the emphasis is on smoothly changing momentum, but is not relevant to the problem of navigating on a sphere, where the geometric constraints are well-understood, but the impact of environmental conditions challenges the discovery of short paths.

5.3 Shortest Path Methods

A different context for the construction of shortest paths (by distance or travel time) is in the graphics community, where geodesics are of fundamental importance in characterising, rendering and manipulating images of surfaces and models of structures. A thorough survey of methods for finding geodesics in this context is that of Crane *et al.* (2020), who characterise solutions within two broad groups: those rooted in computational geometry, treating polyhedral surfaces as exact representations of the structures, and the scientific computing approaches that treat polyhedral meshes as approximations of smooth surfaces. Our work falls somewhere between these two, since we treat the environment as fundamentally discretised by the mesh, but with a model of the surface as spherical. Similarly, our approach combines methods from computational geometry (Dijkstra's algorithm is a common building block for many of these approaches) and scientific computing (using a numerical method to account for the spherical surface).

In their survey, Crane *et al.* highlight the MMP algorithm (Mitchell, Mount, & Papadimitriou, 1987) as a landmark approach to the geodesic problem. This is a forerunner of the work by Mitchell and Papadimitriou (1991) previously discussed, in which they extend the problem to include scalar fields. Lanthier *et al.* (1997) also explore this problem. Lanthier (1999) is also identified as one of the first to attempt to construct geodesics using a graph-based method and the subsequent work

8. Marine navigation typically uses nautical miles (nm) as the unit of distance. The conversion is 1 km equals 0.54 nm.

is a strategy to make the underlying graph better represent the cost of traversal of faces. Crane *et al.* observe that pure graph-based approaches in which vertices are added (Steiner points (Lanthier et al., 1997) or spanner points (1997)) suffer from too poor scaling behaviour in exchange for real accuracy in finding good crossing points for edges of the polyhedral representation of the underlying surface. Crane *et al.* also note that approaches based in iterative path improvement, in which graph-based paths are improved by using the local topology of the paths typically require many iterations, but global approaches to path improvement have been successful at improving performance (Liu, Chen, Xin, He, Liu, & Zhao, 2017).

The methods used in Polar Route share much with the overall frameworks of many of these approaches (mesh-based graph abstractions, iterated path improvement and the impact of scalar weighted regions), but there do not appear to have been efforts to solve the problems in the context of vector fields, perhaps because these do not have natural interpretations in the context of graphics problems. Furthermore, Crane *et al.* do not identify examples of prior work combining the challenges of scalar field resistances with spheroid surface topologies.

While much of the relevant work in path-planning has been done in the context of local, small-scale and often indoor navigation for mobile robots, several automated methods have been developed for solving the navigation problem for ships and other vehicles in different ocean settings (Bijlsma, 1975; Kotovirta et al., 2009; Lehtola et al., 2019; Li, Ringsberg, & Rita, 2020; Mishra, Alok, Rajak, Beg, Bahuguna, & Talati, 2021; Sen & Padhy, 2015; Topaj, Tarovik, Bakharev, & Kondratenko, 2019; Walther, Rizvanolli, Wendebourg, & Jahn, 2016; Zermelo, 1931). The most common methods use uniform mesh-based approximations of the environment (Garrido et al., 2020; Bijlsma, 2001; Kotovirta et al., 2009; Lehtola et al., 2019; Mishra et al., 2021; Sen & Padhy, 2015), combined with the use of heuristic search (Hart, Nilsson, & Raphael, 1968) or greedy methods (Garrido et al., 2020; Lehtola et al., 2019; Mishra et al., 2021; Sen & Padhy, 2015) and some sort of post-processing approach to remove mesh artifacts. In the meshes used, edges or cells (or both, as in the case of Mitchell and Papadimitriou (1991)) may be supplemented with scalars modelling environmental impacts on vessel performance. Mishra *et al.* (2021), compute weights using a mathematical model parameterised by vessel-specific information, and precompile them as travel times on the edges connecting nodes in a uniform mesh. Dijkstra's algorithm is then used to compute the mesh-optimal path. Alternatively, cells are augmented with performance data that affects route efficiency (Garrido et al., 2020; Lehtola et al., 2019). The cost of computing appropriate edge and cell weights is dependent on the complexity of the environment being modelled. Dellin and Srinivasa (2016) have considered shortest path problems in which edge costs are expensive to find and have examined lazy edge computation strategies and the impact they can have on the path finding algorithm.

Kotovirta *et al.* (2009) plan routes in a mesh constructed using a lossy compression technique. To address dependence on the underlying mesh, Powell's conjugate direction method (1964) is used to search for a route between two points taking into account the impact of different environmental conditions on speed. The result is a path that is not constrained by mesh artifacts so is not restricted to the construction of rectilinear paths. The curvature of the Earth is not taken into account, and the number of waypoints needed for route construction must be determined in advance. The distance to be travelled must be split into segments to keep the overall computational cost realistic. Long paths comprising great circle arcs cannot be constructed since curvature is not modelled.

5.4 Methods in Marine and Land Transport

Lehtola *et al.* (2019) describe IcePathFinder, a graph-based optimal path-finding method with a post-processing step to remove some of the resulting mesh artifacts from the paths and improve their geodesic validity. This relaxation monotonically improves the cost of the path by successively removing points using a method similar to the line of sight substitution of Theta* (Daniel et al., 2014). When the cost can no longer be improved, the costs of edges between pairs of points are replaced by the costs of traversing geodesic curves. The resulting paths have smoothed out some of the mesh effects and take into account the curvature of the Earth. The order of removal of points is fixed, but removal of different subsets could lead to better paths: a search over alternatives would be highly computationally costly. We describe some detailed comparisons with IcePathFinder in Section 8.

Transportation path-planning in a quad-tree mesh has been studied (Bast et al., 2016) and used as the core of well-known in-car navigation algorithms. In road navigation, a pre-existing network of accessible roads is used to constrain the routes that can be taken by vehicles between waypoints, with real-time traffic updates used to re-plan paths. In a maritime situation, the routes available to vehicles are constrained by the environmental conditions pertaining in otherwise open water, and paths can be similarly updated as environmental conditions change.

6. Polar Route

In this section we give details of the route planning method used in Polar Route. The first step in the construction of the mesh abstraction is to determine the edge weights of the edges in the accessibility graph defined in Section 4, Definition 5.

6.1 Mesh-Based Route Construction

In the mesh-optimal planning phase of path-construction, a route between waypoints is found by determining the path through the mesh that minimises travel time in the cell array representation of the vector and scalar fields. When an adjacent pair of cells is diagonally arranged, the optimal route between them is initialised to be the path between their centres using straight line connections between the centres and the common corner. The non-diagonal cases to be considered are adjacent arrangements where the destination cell is at the left, right, above or below the source cell. Without loss of generality, we consider an adjacent pair of cells where the task is to go from the left centre to the right centre, taking into account the vector fields in the two cells. In fact, we implement the computation this way, rotating the cells to this configuration and reinterpreting the x and y axes and the environment vectors appropriately. The adjacent cell pair arrangement is shown in Figure 10, highlighting the fact that the two cells might be of different sizes (recall that this can be due to both cell subdivision and the scaling of cell size by cosine of the latitude).

Figure 10 refers to the following quantities: (u_1, v_1) and (u_2, v_2) are the vector field components in the left and right cells respectively; Y is the vertical separation of the two centre points; x is half of the width of the left cell; a is half of the width of the right cell; y indicates the position of the crossing point and b is the vertical distance travelled in the right hand cell.

This adjacent cell pair has the special property that it is context-free, since it does not matter where the left cell is entered or where the right cell is exited, as these positions will not affect the selection of the crossing point between the centres of the adjacent cells. The context-free pattern

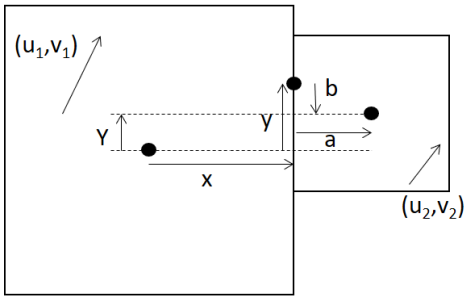

Figure 10: A pair of adjacent cells sharing part of the boundary: Y is the vertical displacement from the left cell centre to the right cell centre, y the vertical displacement of the crossing point relative to the left cell centre and b the displacement of the crossing point from the right cell centre, but measured in the same direction as y (so here it would be negative).

consists of the two adjacent cells and the selected crossing point between them. These patterns are stored and can be retrieved and assembled into paths during mesh-optimal path construction. As explained in Section 4, we perform this process lazily, only determining the values when an edge is accessed, but then storing them for future look-up.

The mesh-optimal path structure puts the adjacent cell pairs in context by specifying, for each pair, the point at which the adjacent pair is entered (the entry point) and the point at which it is exited (the exit point). An abstracted example is shown in Figure 11. Once two adjacent cell pairs are chosen to form part of a path, pairs A and B in the figure, the two curves in C_1 will meet at the centre, allowing them to be combined to form a single path across the cell. This path enters at the crossing point between C_0 and C_1 and exits at the crossing point between C_1 and C_2. As stated in Definition 4, these points are named *fp* (the entry point), *mp* (the midpoint) and *lp* (the exit point).

Figure 12 shows a situation in which a given cell, A, contains an extremely strong vector field effect and the crossing point computed between A and B, using the solution described in Section 6.1.1, lies outside the common boundary of the two cells. In this case, the direct edge from A to B is removed from the graph and any path that connects A and B will route through other cells (the B to A computation is carried out independently). In cases where all adjacent cells contain opposing vector fields with greater magnitude than the achievable speed of the vessel there may be no path from A to B.

The implementation of Dijkstra's algorithm within Polar Route uses a weight function input, w. The function, w, returns the shortest travel time between the cell centres for the given edge. The optimisation process used to determine the best crossing point is described in the next section.

6.1.1 Crossing Point Optimisation

The method for choosing the crossing point receives parameters that have been scaled by the cosines of the latitudes of the two cells, but does not make any further allowance for curvature of the surface. This model is an equirectangular approximation (Veness, 2002) in which cells are treated as having a flat surface and a constant width throughout the cell. This allows us to compute the shortest

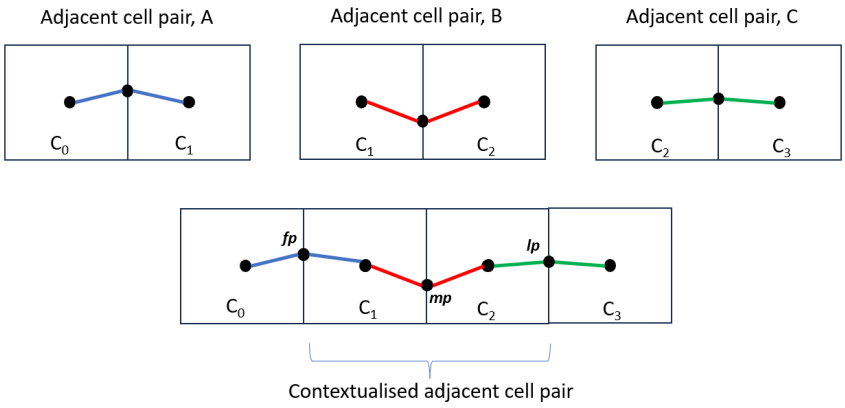

Figure 11: A path is a sequence of connected adjacent cell pairs. As A, B and C are linked to form a path, a contextualised adjacent cell pair is formed by merging the curves in the common cells, C_1 and C_2.

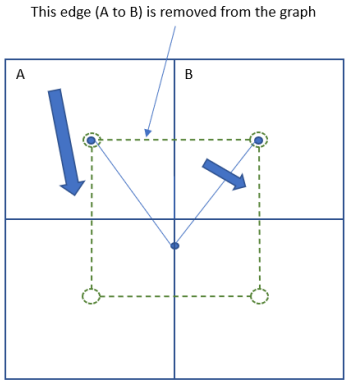

Figure 12: The crossing point falls out of range of the common boundary when travelling from the centre of A to the centre of B, due to the strong vector field in A.

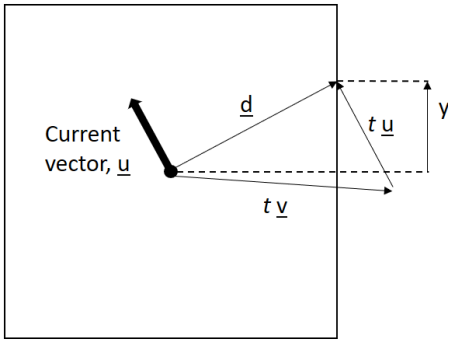

Figure 13: Travel within a single cell: in time t, the vessel will travel $\underline{d}$ due to its velocity, $\underline{v}$, and current, $\underline{u}$. The crossing point is at position y relative to the centre of the cell.

distance across a cell using Pythagoras' theorem, as a simple approximation of the distance across the curved surfaces of the cells.

The effect of the environmental vector field on the vessel is to change its effective velocity on its trajectory, and it will have to choose a heading that takes into account the vector field affecting it. To achieve the straight line route the vessel must maintain a constant net velocity along that line, which will be the sum of the vector field effect and the velocity of the vessel. In our model, the speed of the vessel in a cell is dependent on both the vector and scalar fields that apply in that cell.

If the vessel crosses the central boundary between cells at a selected point, y (measured relative to the horizontal axis through the centre of the left cell – see Figure 10), then the time, t, it takes to travel from the centre of the left-hand cell to the crossing point (or from the crossing point to the centre of the right-hand cell) satisfies:

$$t(\underline{u} + \underline{v}) = \underline{d} \tag{1}$$

as shown in Figure 13, where $\underline{u}$ is the environment vector in the cell, $\underline{v}$ is the velocity of the vessel and $\underline{d}$ is the vector from the centre of the cell to the crossing point on the boundary. The cell-dependent speed of the vessel is constant on this path.

We proceed as follows. First we find the solution for t as a function of the crossing point, defined in terms of the distance y as shown in Figure 10. Then we find the crossing point, y, that minimises the sum of the travel times in the two adjacent cells. As illustrated in Figure 6, this function (in the equirectangular case) is a quadratic and we use Newton's method to optimise the choice of y. Newton's method finds successively better approximations to the roots of a continuous function, iterating until a convergence condition is achieved. The single variable function case is defined as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

where $f'(x)$ denotes the first derivative of $f(x)$.

The travel times in the two cells we denote $tt_l(y)$ (the time to travel to the crossing point from the centre of the left cell) and $tt_r(y)$ (the time to travel to the centre of the right cell from the crossing point). We denote the speed of the vessel in the left cell as s_l and its speed in the right cell as s_r.

Rearranging Equation 1, the travel time in the left cell is derived as follows.

$$t\underline{v} = \underline{d} - t\underline{u} \tag{3}$$

Considering the left cell, in which the speed of the vessel is s_l, the vector from the centre to the crossing point is $\underline{d_1}$ and the environment vector field is $\underline{u_1}$, the square of Equation 3 is:

$$s_l^2 t^2 - |\underline{u_1}|^2 t^2 + 2D_1 t - |\underline{d_1}|^2 = 0 \tag{4}$$

where $D_1 = \underline{u_1}.\underline{d_1}$. This equation can be solved for t:

$$t = \frac{-2D_1 \pm \sqrt{4D_1^2 + 4|\underline{d_1}|^2(s_l^2 - |\underline{u_1}|^2)}}{2(s_l^2 - |\underline{u_1}|^2)} \tag{5}$$

$$= \frac{-D_1 \pm \sqrt{D_1^2 + |\underline{d_1}|^2(s_l^2 - |\underline{u_1}|^2)}}{s_l^2 - |\underline{u_1}|^2} \tag{6}$$

Since we are only interested in positive values of t, we have:

$$t = \frac{\sqrt{D_1^2 + |\underline{d_1}|^2(s_l^2 - |\underline{u_1}|^2)} - D_1}{s_l^2 - |\underline{u_1}|^2} \tag{7}$$

with a special case when $s_l^2 = |\underline{u_1}|^2$ which is when the magnitude of the vector field matches the speed of the vessel in the left cell. In this case, $t = \frac{|d_1|^2}{2D_1}$, by equation 4, being undefined when $D_1 = 0$ (the case in which the vector field is orthogonal to the intended direction of travel). If D_1 is negative, this latter case is degenerate, with the vessel not fast enough to overcome the vector field effect in the intended direction. The solution for t defines the function $tt_l(y)$ for a given vector field, cell size and speed. The function $tt_r(y)$ is defined similarly, using s_r, $\underline{u_2}$, D_2 and $\underline{d_2}$.

Our function, $f(y)$ is then the sum of the two travel times in each of the left cell, $tt_l(y)$, and right cell, $tt_r(y)$, $tt_l(y) + tt_r(y)$, which we want to minimise by choosing the best y possible. This is found by iterating over equation 8 until a convergence condition is reached. The ticks denote the first and second derivatives and the subscripted l and r indicate whether travel is through the left or right cell respectively.

$$y_{k+1} = y_k - \frac{tt'_l(y_k) + tt'_r(y_k)}{tt''_l(y_k) + tt''_r(y_k)} \tag{8}$$

The squared distance travelled in the right box is $a^2 + b^2$, where b is equal to $Y - y$. The distance from the centre of the left cell to the central boundary is x (see Figure 10). The distances travelled in the two cells are therefore defined using Pythagoras as follows.

$$d_1^2 = x^2 + y^2 \tag{9}$$

$$d_2^2 = a^2 + b^2 = a^2 + (Y - y)^2 \tag{10}$$

The following terms are helpful in simplifying the derivations below. We use the shorthand ∂ to refer to the operator $\frac{d}{dy}$.

$$C_1 = s_l^2 - |\underline{u_1}|^2 \tag{11}$$

$$C_2 = s_r^2 - |\underline{u_2^2}| \tag{12}$$

$$D_1 = u_1 x + v_1 y \tag{13}$$

$$\partial D_1 = v_1 \tag{14}$$

$$D_2 = u_2 a + v_2 (Y - y) \tag{15}$$

$$\partial D_2 = -v_2 \tag{16}$$

$$X_1 = \sqrt{D_1^2 + C_1 d_1^2} \tag{17}$$

$$X_2 = \sqrt{D_2^2 + C_2 d_2^2} \tag{18}$$

Equations 11 to 18 name the left and right cell components of the expressions in Equation 7. Equation 7 has an analogous form for the right cell, and the pair of equations can be abbreviated as Equations 19 and 20. These expressions are the travel time functions, $tt_l(y)$ and $tt_r(y)$ respectively, taking values t_1 and t_2 as follows:

$$t_1 = \frac{X_1 - D_1}{C_1} \tag{19}$$

$$t_2 = \frac{X_2 - D_2}{C_2} \tag{20}$$

Differentiation of Equation 4, using Equation 11, yields:

$$\partial t_1 2 C_1 t_1 + \partial t_1 2 D_1 + 2 v_1 t_1 - 2y = 0 \tag{21}$$

Factorising gives:

$$\partial t_1 (C_1 t_1 + D_1) = y - t_1 v_1 \tag{22}$$

and by a similar derivation for the right cell:

$$\partial t_2 (C_2 t_2 + D_2) = y - Y + t_2 v_2 \tag{23}$$

The minimum travel time is achieved when $\partial t_1 + \partial t_2 = 0$. By Equation 19 we have that:

$$C_1 t_1 = X_1 - D_1 \tag{24}$$

so

$$X_1 = C_1 t_1 + D_1 \tag{25}$$

and, similarly:

$$X_2 = C_2 t_2 + D_2 \tag{26}$$

The derivatives of X_1 and X_2 are obtained as follows. Squaring both sides of Equation 17 and differentiating using Equation 14, leads to:

$$\partial X_1 = \frac{D_1 v_1 + C_1 y}{X_1} \tag{27}$$

and by similar steps:

$$\partial X_2 = \frac{-D_2 v_2 - C_2(Y - y)}{X_2} \tag{28}$$

Using Equations 22 and 25 we have that

$$\partial t_1 = \frac{y - t_1 v_1}{X_1} \tag{29}$$

and, by similar reasoning from Equations 23 and 26:

$$\partial t_2 = \frac{y - Y + t_2 v_2}{X_2} \tag{30}$$

To obtain the minimum travel time, we require that the derivative, $\partial t_1 + \partial t_2$, is zero:

$$\frac{y - Y + t_2 v_2}{X_2} + \frac{y - t_1 v_1}{X_1} = 0 \tag{31}$$

Rewriting this expression using Equations 19 and 20, we define the function $F(y)$:

$$F(y) = X_2(y - \frac{v_1(X_1 - D_1)}{C_1}) + X_1(y - Y + \frac{v_2(X_2 - D_2)}{C_2}) \tag{32}$$

It can be seen that this equation corresponds to $F(y) = X_1 X_2(tt'_l(y) + tt'_r(y))$, which is zero precisely when $f(y) = \partial t_1 + \partial t_2$ is zero. By writing $F(y)$ in the following form:

$$F(y) = (X_1 + X_2)y - \frac{(X_1 - D_1)X_2 v_1}{C_1} + \frac{(X_2 - D_2)X_1 v_2}{C_2} - Y X_1 \tag{33}$$

it can be seen that the derivative of $F(y)$ is:

$$\begin{aligned} \partial F(y) = {} & (X_1 + X_2) + y(\partial X_1 + \partial X_2) \\ & - \tfrac{v_1}{C_1}(\partial X_2(X_1 - D_1) + X_2(\partial X_1 - \partial D_1)) \\ & + \tfrac{v_2}{C_2}(\partial X_1(X_2 - D_2) + X_1(\partial X_2 - \partial D_2)) \\ & - Y \partial X_1 \end{aligned} \tag{34}$$

Using the definitions of $F(y)$ and $\partial F(y)$, we can apply Newton's method to arrive at a value of y that makes $F(y) = 0$.

6.2 Route Smoothing

The smoothing process smooths the adjacent cell pairs in the order in which they appear in the mesh-optimal path. The smoothed passage between the cells must pass between the entry and exit points connecting the adjacent cell pairs without the requirement to visit the centres of the cells. Starting from the mesh-optimal path, route smoothing iterates over the path structure until convergence. On each iteration, i, successive pairs of adjacent cells are considered, using the boundary points of these cells in the mesh-optimal path to constrain and inform the positions of these points in the resulting partially smoothed path, p_i (Definitions 6 and 7).

Definition 6. *Given a mesh-optimal path p, a* **partially smoothed path**, *p_i, is a path found after i iterations of smoothing starting from the path p_0, obtained from p by removing, for each adjacent cell pair, the cell centre points and connecting both fp and mp, and mp and lp, by straight lines.*

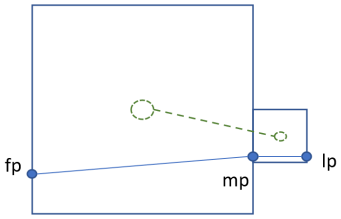

Figure 14: The relationship between the *fp, mp* and *lp* points in an adjacent pair, also showing the graph edge and centre points (dotted) used to construct the mesh-optimal path.

Definition 7. *A partially smoothed path, p_i, has* **converged** *if, when compared with the partially smoothed path p_{i-1}, there is a 1:1 correspondence between all of the waypoints in p_i and p_{i-1}, and none of the waypoints in p_i differ in position by more than a given tiny constant δ from their positions in path p_{i-1}.*

A single iteration of smoothing therefore entails choosing, for each adjacent cell pair in sequence, the optimal *mp*, given its fp and lp (see Figures 14 and 15). These points are collectively referred to as the *tracking points* as they allow the process of smoothing to be tracked along the sequence of adjacent cell pairs. This requires a second invocation of Newton's method, as the optimal *mp*, given fp and lp, will not be the same as the optimal crossing point between the cell centres chosen during mesh-optimal path construction. The new crossing point calculation no longer uses the accessibility graph because the crossing point decision is no longer context-free. The mesh-optimal path has provided the context in which the new crossing point is selected.

On this second invocation, we introduce a more sophisticated approximation of the curvature of the sphere that models the change in width *within* a cell by introducing a latitude correction. This was not needed in the use of Newton's method during mesh-optimal path construction because all paths are forced to go between the centres of the two cells, dominating any sensitivity to latitude changes within a cell, and making the fixed cell-width assumption adequate in that context.

The computation of *mp* differs for horizontal and vertical orientations of the cell pair, as described in Appendices C and D respectively.

When the new *mp* falls outside the shared boundary of the two cells, as is shown in Figure 12, new cells are introduced into the path, in the direction of the *mp*, to allow the eventual smoothed path to take its more efficient route. The introduction of new cells requires that they be checked to ensure they are not blocked. Over multiple iterations the addition of new cells can change the course of the path. New cells are introduced in several different contexts during smoothing, depending on the configuration of the adjacent cell pair itself and the position of the new *mp*. We cover all of these cases in Section 6.3.

6.3 Smoothing Algorithm

In this section we present the details of the smoothing algorithm. Twelve auxiliary functions are referred to in the algorithm components, and are defined in Appendix B.

The inner loop of the smoothing algorithm shown in Algorithm 1 consists of a data structure and seven functions. Communication between these functions is managed via the data structure

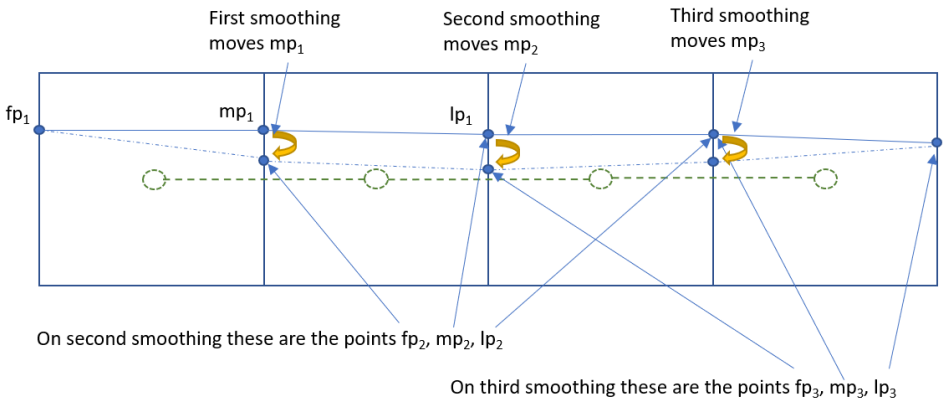

Figure 15: How the smoothing process proceeds through a sequence of adjacent cell pairs, smoothing the selection of *mp* in each adjacent cell pair in turn.

```
1  Record {
2  |  aps ;                  // The list of adjacent cell pairs in the path
3  |  pathIndex ;            // Index of the cell pair currently being processed
4  |  pathSize ;             // Length of aps
5  |  ap ;                   // The current adjacent cell pair of interest
6  |  ap′ ;                  // The next cell pair of interest
7  |  fp;
8  |  mp;
9  |  lp;
10 |  converged;
11 |  complete ;             // The status of the current loop iteration
12 } SmoothingRecord;
```

Data Structure 1: The data structure used in the smoothing functions.

which carries the context that is maintained and used by each of the functions during the execution of the main loop. Convergence is checked for each time round the main loop. The context, called the *smoothing record*, consists of the list of adjacent cell pairs making up the path (which changes as cells are added to and removed from the path during smoothing), the index of the cell pair about to be smoothed and the tracking points associated with that pair. At the start of the inner loop, the smoothing record is initialised for the current iteration and information about the way the loop proceeded is collected and passed back from the functions as the loop executes.

The smoothing record data structure is defined as shown in Data Structure 1. After setting up the context, the five main stages of Algorithm 1 are: initialisation, removal of reversed edges, merging of close points, handling diagonal edges and the general case in which Newton's method is used to solve the equation that determines the new crossing point between adjacent cells on the path.

```
   input  : Mesh Optimal Path, P
   output: Smoothed path
 1 begin
 2     startwp := the start waypoint of P;
 3     endwp := the end waypoint of P;
 4     sr := new SmoothingRecord();
 5     sr.aps := the list of adjacent cell pairs in P;
 6     sr.converged := false;
 7     while not sr.converged do
 8         sr.fp := startwp;
 9         sr.mp := null;
10         sr.lp := null;
11         sr.pathIndex := 0;
12         sr.pathSize := length(sr.aps);
13         sr.converged := true ;            // Assume converged until shown otherwise
14         while sr.pathIndex < sr.pathSize do
15             sr := initialise(sr, endwp);
16             sr := removeReversedEdges(sr);
17             sr := closePoints(sr);
18             sr := handleDiagonalPath(sr);
19             sr := smoothCrossing(sr, P);
20     return P;
```

Algorithm 1: The Smoothing Algorithm. The inner loop is a sequence of function calls. If any one of them completes the handling for this iteration, the remaining functions will do nothing, as shown in their definitions.

Initialisation, shown in Algorithm 2, simply receives the smoothing record and sets up the adjacent pair, the *mp* and the *lp* of the context to be used for the next iteration of the smoothing process. When initialising, the *fp* will be the start waypoint.

The removal of reversed edges, shown in Algorithm 3, is done at the beginning of every iteration. Reversed edges can occur as a result of adding edges during smoothing, because the smoothing process only has a local view of the choices of cells reachable from the current adjacent cell pair. It is not always clear where reversed edges might arise, so it is best to remove reversals systematically. We therefore check for, and remove, reversed edges before going into the next iteration of the smoothing process.

The next stage is to merge points that have moved sufficiently close together to be considered the same point. As smoothing progresses, the *fp* and *mp*, or the *mp* and *lp*, might approach one another to within our minimum separation (defined using the constant *MERGESEP*). An example is shown in Figure 17. This situation can arise as the path smoothing moves points on two sides of a cell towards a common corner. When *fp* and *mp* approach one another, the algorithm simply delays handling the situation until the next iteration of the smoothing, and moves on to the next edge. If, however, the *mp* and *lp* approach too closely, the situation is handled by removing the redundant ith and $i+1$th edges and replacing them with a direct edge between the start of the

```
1  Function initialise(sr, endwp):
       input  : SmoothingRecord, sr, final waypoint, endwp
       output: modified SmoothingRecord
2      begin
3          sr.ap := sr.aps[sr.pathIndex] ;            // The current adjacent cell pair
4          sr.mp := sr.ap.crossing;
5          if sr.pathIndex + 1 < sr.pathSize then
6              sr.ap′ := sr.aps[sr.pathIndex + 1];
7              sr.lp := sr.ap′.crossing;
8          else
9              sr.ap′ := null;
10             sr.lp := endwp;
11         return sr;
```

Algorithm 2: Initialisation function that sets the various cells and points of interest in a single iteration of smoothing.

```
1  Function removeReversedEdges(sr):
       input  : SmoothingRecord, sr
       output: modified SmoothingRecord
2      begin
3          if not sr.ap′ == null and sr.ap.start == sr.ap′.end then     // Reversed edges
4              remove(sr.pathIndex, sr.aps);  // Remove the indicated element from aps
5              remove(sr.pathIndex, sr.aps);  // And again, to remove the reversed edge
6              sr.pathSize := sr.pathSize − 2;
7              sr.converged := false ;        // Any change in path breaks convergence
8              sr.complete := true;
9          return sr;
```

Algorithm 3: Removal of a reversed edge pair: this phenomenon can occur as a result of edge insertions during smoothing and the pair is removed and the path linked directly between the two ends of the pair.

former to the end of the latter. It is certain that this edge must exist because the cells it connects share the common corner at which the points are converging, and are both accessible. The merging method is shown in Algorithm 4.

Once these tidying-up steps are complete, we move on to the next iteration of smoothing of the partially smoothed path.

Algorithm 5 describes the process for smoothing a diagonal edge. It can arise that the optimal path actually passes through a corner, but typically the smoothed route through a diagonally adjacent cell pair will need to pass through either the top or bottom cell on the missing diagonal, as shown by the dotted lines in Figure 18. In Algorithm 5, the correct cell to introduce is found by $selectSide(fp, mp, lp, ap)$, which uses spherical geometry to determine on which side mp lies relative

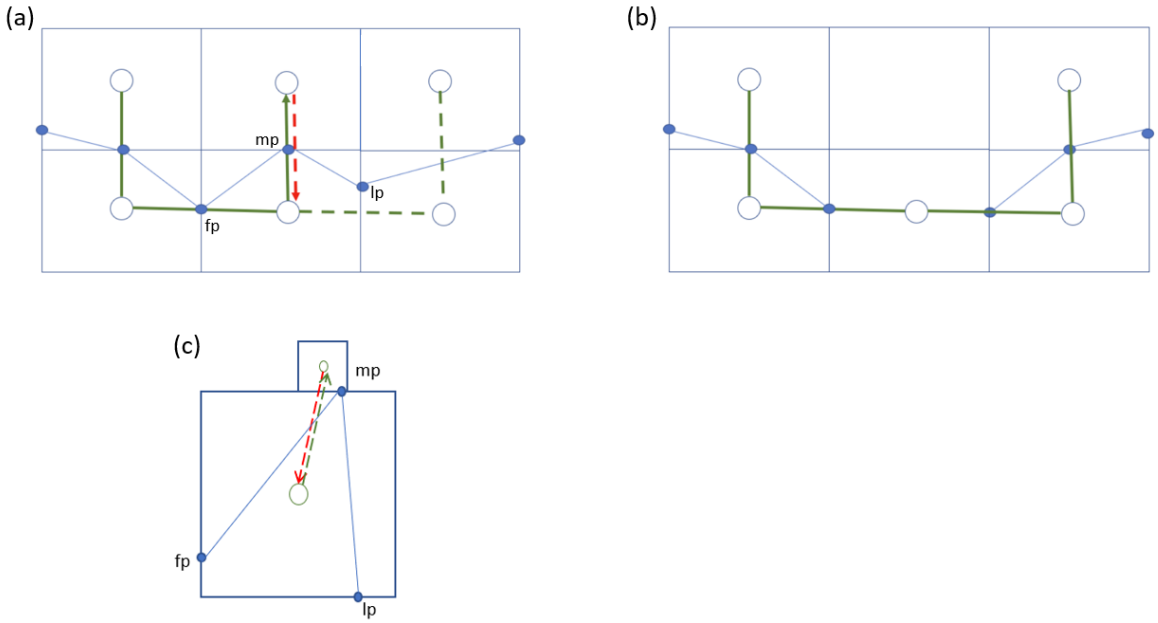

Figure 16: (a) A case in which reversed edges can be produced (the red dotted line and the green line in the middle cell reverse one another). (b) These are removed. (c) The general case of the central panel of part (a), in which the two cells are of different sizes.

to the great circle arc segment between *fp* and *lp*. Provided that the selected cell is accessible, new edges are added to include it in the path.

The final stage is the main smoothing stage, in which Newton's method is used to choose the new crossing point. This stage is described in Algorithm 6. Significant changes to the mesh-optimal path can now arise, as new cells will need to be entered if this new crossing point is out of range of either of the adjacent cells in the pair. In this case, the algorithm identifies, using the $nearestNeighbour$ function, any cells that the path might have to newly enter, called the *target* cells, and finds the edges that will need to be added. If either of the target cells is blocked, the chosen crossing point must be clipped back to the closest corner. These cases are exemplified in Figures 19 and 20, which show the *targetA* and *targetB* cases detailed in Algorithm 6.

The edges connecting the newly added cells to the path are identified from the underlying neighbourhood graph, which respects the inaccessibility of cells and the impact of high magnitude vector fields. The decision about which of the identified edges to add is then made by Algorithm 7. When the new crossing point falls outside the range of the start cell, but inside the range of the end cell, or vice versa, only one new target cell needs to be added, and hence only two new edges are required. These cases are referred to as *V-type* cases, since the two edges are connected to form a V. When the new crossing point falls outside of the boundaries of both the start and end cell, two new target cells are added, and hence three new edges are identified and inserted. This is a *U-type* case. In both the V-type and U-type cases, the original edge connecting the start and end cells is removed. Since the three edges in the U-type case form a characteristic horseshoe shape, this case

```
1  Function closePoints(sr):
       input  : SmoothingRecord, sr
       output: modified SmoothingRecord
2      begin
3          if not sr.complete then
4              if dist(sr.fp, sr.mp) < MERGESEP then
5                  ++sr.pathIndex;
6                  sr.fp := sr.mp;
7                  sr.complete := true;
8              else if dist(sr.mp, sr.lp) < MERGESEP and sr.pathIndex < sr.pathSize − 1
                   then                                   // Final waypoint cannot merge
9                  newEdge := findEdge(sr.ap.start, sr.ap′.end);
10                 remove(sr.pathIndex, sr.aps);          // Remove ap
11                 remove(sr.pathIndex, sr.aps);          // Remove ap′
12                 insert(sr.pathIndex, [newEdge], sr.aps);
13                 sr.pathSize := sr.pathSize − 1;
14                 sr.converged := false;
15                 sr.complete := true;
16         return sr;
```

Algorithm 4: Handling situations in which the midpoint is within MERGESEP of either the first or last point.

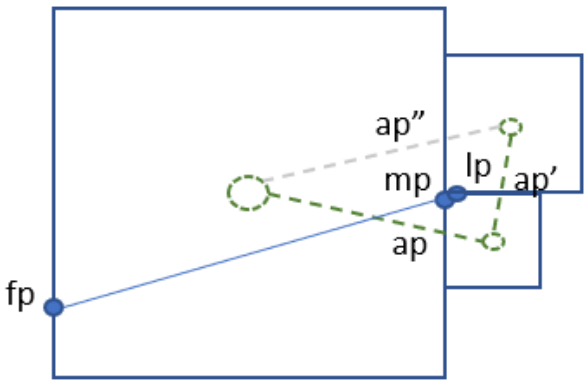

Figure 17: The midpoint, *mp*, and lastpoint, *lp* are within *MERGESEP* of one another. The tracking points, *fp*, *mp* and *lp* are associated with the adjacent cell pair consisting of the two cells linked by the graph edge *ap*. The subsequent adjacent cell pair in the path consists of the cells connected by graph edge ap'. The edge, ap'' completes the graph of these three cells, but this edge does not connect adjacent cells on the path.

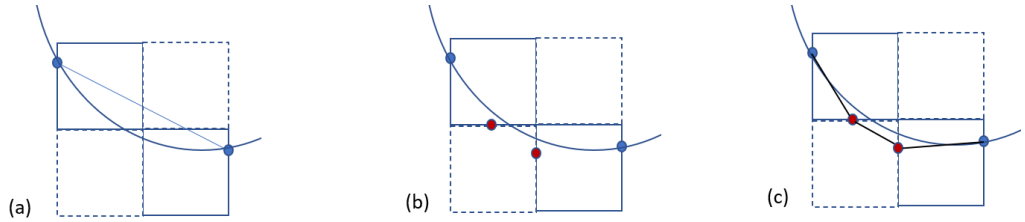

Figure 18: Smoothing a diagonal section of a path. (a) The arc segment is used to identify the cell to include. (b) The two new adjacent cell pairs are added, and the crossing points (marked in red) are added. (c) The arc is approximated by a series of constant bearing legs between these pairs of points.

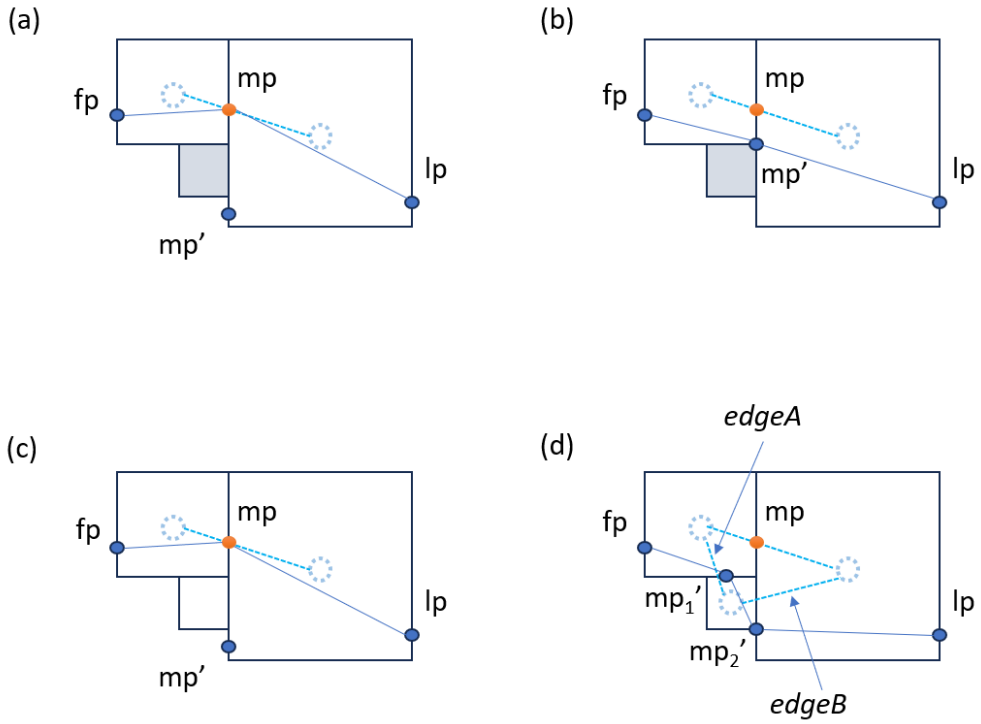

Figure 19: The *targetA* case: (a) New cell is blocked (shaded) and the proposed new midpoint, mp', lies outside it. (b) The mp point is clipped back to the original cell pair and the new position of mp' lies on the corner of the blocked cell. (c) New cell is accessible, but mp' lies outside its boundary. (d) New edges are introduced, *edgeA* and *edgeB*, to replace the original adjacent cell pair. These have separate midpoints, mp'_1 for the first pair and mp'_2 for the second, clipped back to lie inside the boundary of the newly entered cell. Successive iterations of smoothing will move mp'_1 west, possibly out of the new cell, adding a further cell to the path. In all cases, the original mp is no longer in the path.

```
1  Function handleDiagonalPath(sr):
       input  : SmoothingRecord, sr
       output: modified SmoothingRecord
2      begin
3          if not sr.complete then
4              if isDiagonal(sr.ap) then
5                  target := selectSide(sr.fp, sr.mp, sr.lp, sr.ap);
6                  if not blocked(target, sr.ap.start, sr.ap.end) then
7                      edgeA := findEdge(sr.ap.start, target);
8                      edgeB := findEdge(target, sr.ap.end);
9                      remove(sr.pathIndex, sr.aps);
10                     insert(sr.pathIndex, [edgeA, edgeB], sr.aps);
11                     ++ sr.pathSize;
12                     sr.converged := false;
13                 else
14                     ++sr.pathIndex;
15                     sr.fp := sr.mp;
16             sr.complete := true;
17         return sr;
```

Algorithm 5: Smoothing diagonal edges.

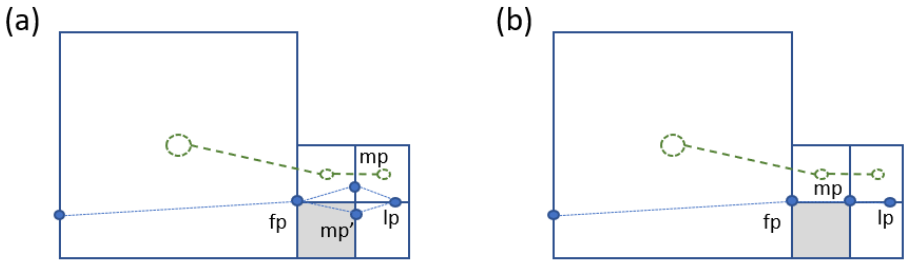

Figure 20: In a *targetB* situation: (a) After clipping to the corner on the left of the blocked (shaded) cell, we move to the next triple in which *fp* is the clipped point. It can then happen that an attempt is made to move the position of *mp* to *mp*′, a position also blocked by the same cell. (b) The crossing point *mp*′ is clipped back to the original cell corner, becoming the new value of *mp*. This leads to the edge of the blocked cell being tracked.

is referred to as a horseshoe (line 23 of Algorithm 7). When all necessary edges have been added, the updated smoothing record is returned (line 39 of Algorithm 7).

Figure 21 shows the addition of a U-type horseshoe in the horizontal case, when both cells are the same size and when they are of different sizes as a result of cell-subdivision.

```
1  Function smoothCrossing(sr, P):
       input  : SmoothingRecord, sr, mesh-optimal path, P
       output: modified SmoothingRecord
2      begin
3          if not sr.complete then
4              mp' := NewtonSmooth(sr.fp, sr.mp, sr.lp, sr.ap);
5              if mp' == null then
6                  throw P;                // Newton call failed to converge or recover
7              targetA := nearestNeighbour(ap.start, ap.end, mp');
8              if blocked(targetA,ap.start,ap.end) then
9                  mp' := clipTo(ap.start, mp');
10                 targetA := null;
11             else
12                 edgeA := findEdge(sr.ap.start, targetA);
13                 mp' := clipTo(targetA, mp');
14             targetB := nearestNeighbour(sr.ap.end, sr.ap.start, mp');
15             if blocked(targetB,sr.ap.start,sr.ap.end) then
16                 mp' := clipTo(sr.ap.end, mp');
17                 targetB := null;
18             else
19                 edgeC := findEdge(targetB, sr.ap.end);
20                 mp' := clipTo(targetB, mp');
21             if inside(mp', sr.ap.start) then
22                 targetA := null;
23             return addEdges(targetA, targetB, sr);
24         return sr;
```

Algorithm 6: Find the new *mp*, using Newton's method to solve the equation, and then check whether this requires any additional edges to be added. The actual checks and addition of edges are handled in the auxiliary function, *addEdges*.

Situations can arise in which the smoothing process can lead to oscillating patterns in the insertion and removal of edges: this is caused when the spherical surface leads to a shorter physical distance being traversed if a path pushes closer to a pole, but the conditions in the newly entered cell or cells created by the vector field lead the smoothing to seek to cross outside the newly entered cells, pushing the path back into the cells it left. These situations can be recognised by the repeated visits to the same context and are trapped in the tests on lines 26 and 31 of Algorithm 7. These two functions, *newPair* and *newTriplet*, test whether the edge combination (pair or triple, respectively) has arisen before and, if so, whether the calculated crossing point, mp', is close to (within *MERGESEP* of) a crossing point previously calculated in this context. If these conditions are met, then the situation is considered a repetition and handled by *clipping* the crossing point to the appropriate corner of the start or end cell. In all cases, the pair or triplet, together with

```
1  Function addEdges(sr, P):
       input  : Cells targetA, targetB; SmoothingRecord, sr
       output: modified SmoothingRecord
2      begin
3          if targetA == null then
4              if targetB == null then
5                  if dist(sr.mp, mp′) > MERGESEP then
6                      sr.converged := false;
7                  // Simple case:  smoothing stays inside cells
8                  sr.ap.crossing := mp′;
9                  ++ sr.pathIndex;                                  // Step along path
10                 sr.fp := mp′;
11             else                                          // Outside destination cell
12                 edgeB := findEdge(sr.ap.start, targetB);
13                 remove(sr.pathIndex, sr.aps);
14                 insert(sr.pathIndex, [edgeB, edgeC], sr.aps);
15                 ++ sr.pathSize;
16                 sr.converged := false;
17         else
18             sr.converged := false;
19             if targetB == null then                          // Outside origin cell
20                 edgeB := findEdge(targetA, sr.ap.end);
21                 remove(sr.pathIndex, sr.aps);
22                 insert(sr.pathIndex, [edgeA, edgeB], sr.aps);
23                 ++ sr.pathSize;
24                                       // Else:  outside both cells - horseshoes
25             else if targetA == targetB  and
26                     newPair(edgeA, edgeC, mp′) then
27                 remove(sr.pathIndex, sr.aps);
28                 insert(sr.pathIndex, [edgeA, edgeC], sr.aps);
29                 sr.pathSize := sr.pathSize + 1;
30             else if not targetA == targetB and
31                     newTriplet(edgeA, edgeB, edgeC, mp′) then
32                 edgeB := findEdge(targetA, targetB);
33                 remove(sr.pathIndex, sr.aps);
34                 insert(sr.pathIndex, [edgeA, edgeB, edgeC], sr.aps);
35                 sr.pathSize := sr.pathSize + 2;
36             else
37                 sr.ap.crossing := clipTo(sr.ap.start, mp′);
38                 sr.ap.crossing := clipTo(sr.ap.start, sr.ap.crossing);
39                 sr.converged := dist(sr.mp, sr.ap.crossing) < MERGESEP;
40         return sr;
```

Algorithm 7: Addition of new edges in the U-type (horseshoe) and V-type patterns.

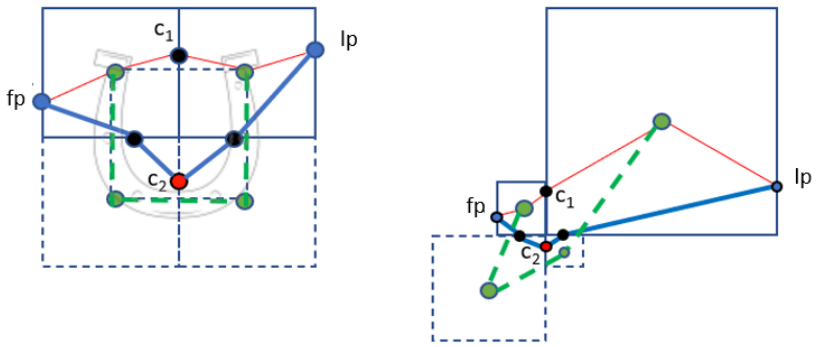

Figure 21: The horseshoe pattern (a reference to the green-dashed edges newly added to the path) for smoothing a horizontal adjacent cell pair. On the left, the case for two equal-sized cells. On the right, a more general case in which the cell sizes in the original pair differ. The original crossing point in the mesh-optimal path is c_1. The edges of the mesh-optimal path are shown in red. The new crossing point chosen during smoothing is c_2. In both cases, c_2 drops out of range of the bottom boundaries of the two cells, due to the improved curvature approximation used in the smoothing process, requiring the introduction of a horseshoe pattern. The green nodes and dotted lines are not part of the smoothed path, but indicate the three new adjacent cell pairs that comprise the horseshoe (U-type) pattern. The edges of the smoothed path (at this iteration) are shown in blue.

the calculated crossing point, are added to a record of such contexts to ensure that subsequent repetitions are trapped.

In multiple places in the overall smoothing algorithm, a check is required to determine whether a new cell is accessible. This test checks that edges exist to and from the cell to form the edges of the path. However, an additional requirement is that the impact of the scalar and vector fields within the new cell will not outweigh the potential benefit of taking the shorter route by slowing the vessel too much. Until the final path is known, it is impossible to know how much distance the vessel will have to traverse through the new cell and, therefore, the extent of the impact of the fields on its performance. This can lead to the path being moved into a new cell, only to be moved back out again as the smoothing process shortens the segment across a costly cell until it clips a corner. This behaviour ultimately resolves, using the context-recording mechanism just described, but in order to reduce computation time we abbreviate this process by the use of a simple heuristic implemented in the *blocked* auxilliary function. The heuristic checks that the conditions in the target cell are not much worse relative to those in either of the two original cells on the path.

When choosing the mp between two horizontally adjacent cells, the latitude of mp varies. The journeys to and from mp are determined by the latitudes of the fp and lp of the adjacent cell pair. When travelling vertically, the latitude of mp is fixed, and it is the longitude that varies. Also, travel to and from mp is in the longitudinal direction so is not affected by the latitudes of fp and

lp. Therefore the vertical case requires a different method for computing the travel time between the cells.

In the vertical case, described in Appendix D, mp must lie on the fixed latitude boundary between the cells, but if fp and lp are close to one of the longitudinal boundaries mp might be pushed horizontally beyond the vertical boundaries of the two cells. This results in the need for the addition of either a U-type or a V-type pattern. The way that the new cells and required edges are identified is exactly analogous to the horizontal case.

As the smoothing process nears convergence, no further pairs will be introduced but crossing points will continue to be moved until the shortest path (according to our curvature approximation) has been found. As the empirical results shown below illustrate, convergence is typically achieved after fewer than 3000 outer-loop iterations, with some paths converging after a few hundred, and some unusual cases requiring as many as 20,000 iterations to converge (see the artificial examples in Figure 3).

The algorithms of Polar Route are sound (any path found by the approach is a valid path according to Definition 3). This follows from the facts that Dijkstra's shortest path algorithm is sound and that the smoothing algorithm cannot break any of the properties of a path in its modifications.

By suitable selection of mesh sizes the accuracy can be improved to achieve an arbitrary ϵ-precision for any selected $\epsilon > 0$. This implies a form of completeness of the Polar Route algorithm, since it implies that, by a suitable choice of mesh resolution, one can always generate a path within ϵ-precision of the optimal path. A sketch of the proof for this claim is as follows. Smoothing can never make the path longer than the underlying Dijkstra path, so it is sufficient to show that the Dijsktra path itself converges to the optimal path as the mesh resolution is increased. The distance between points in the graph used for the Dijkstra path overestimates the true distances between the cell centres by approximating the effect of curvature. The local distance metric used to define the edge lengths in the graph uses rhumb line distances, while the optimal path has a length greater than or equal to the great circle arc (it might be longer due to obstacles or the effects of environmental fields). As mesh resolution increases, the rhumb line distance between points converges towards the great circle arc distance (see, for example, Vezie's report (2016)). Thus, the local metric converges to the true distance and, since the Dijkstra path is globally optimal within the graph, the Dijkstra path converges on the optimal smooth path.

7. Evaluation and Results

All of the results reported in this paper were generated on a 32 Gb Macbook Pro M2 Max laptop computer.

A basic requirement of a path-planner on a spherical surface is that it closely approximates great circles in the absence of environmental features. Therefore, to begin the evaluation of our method, we compare the lengths of smoothed paths in unconstrained environments with the lengths of great circle arcs generated between the same pairs of waypoints. A smoothed path of the same length as the great circle arc must be identical to it.

We use the haversine distance measurement and rhumb line distance measurement desribed in the online resource entitled *Calculate distance, bearing and more between Latitude/Longitude points*[9]. As discussed above, the route planner generates smoothed paths as sequences of rhumb

9. http://www.movable-type.co.uk/scripts/latlong.html

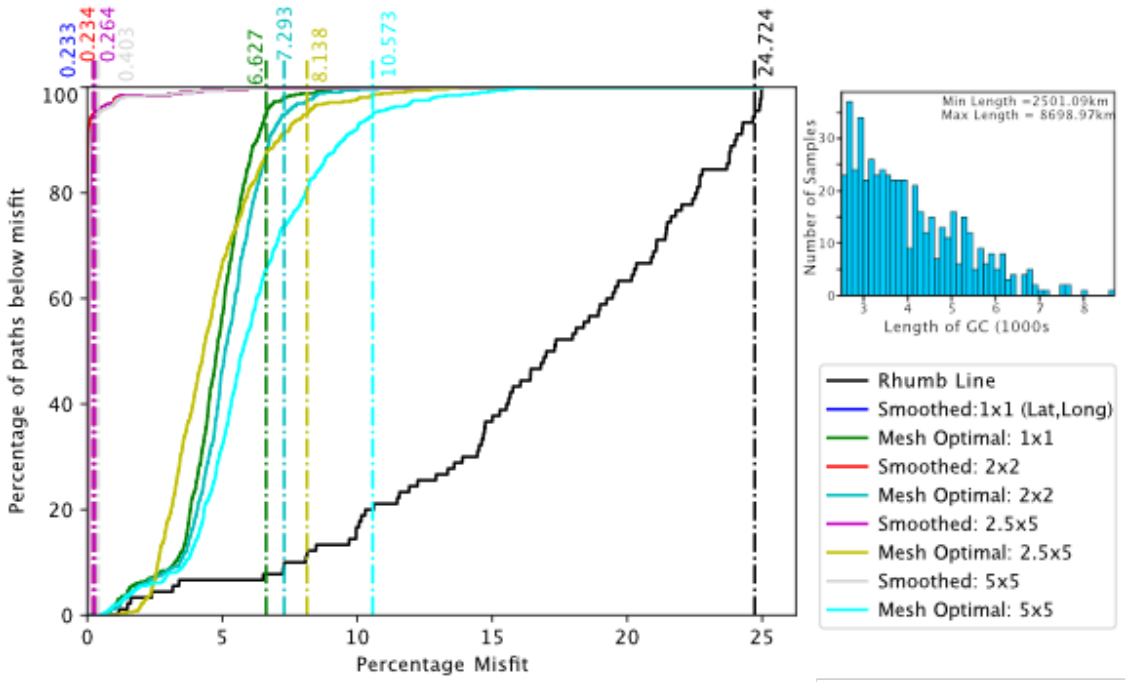

Figure 22: Percentage misfits with great-circle arcs, for a collection of 522 randomly selected start and end waypoint pairs within a domain spanning longitude -140° to 0° and latitude -75° to -45°. The histogram shows the distribution of waypoint pairs considered, in terms of great circle distances between them.

lines, so we expect a small error, with respect to the length of the corresponding great circle, to accumulate over long paths.

We first show that Polar Route generates highly accurate approximations of great circle arcs in unconstrained environments. We randomly generate pairs of waypoints at least 2,500 km apart, located at a range of latitudes, and path-plan between them. We set a limit of 20,000 outer-loop path-smoothing iterations per path. All of the paths reported in our experiments converged and, in practice, about 80% of the paths converged in fewer than 3,000 iterations. The number of iterations required to smooth a path is correlated with the distance between the waypoints.

As shown in the legend of Figure 22, three types of paths are considered: mesh-optimal paths and smoothed paths, between the randomly selected pairs of waypoints and, for comparison, the single rhumb lines between the pairs of points. The mesh-optimal and smoothed paths are generated using uniform meshes of different cell sizes, as indicated.

We measure a length "misfit" in kilometers between each of the paths in the three different path types and the great circles between the same pairs of waypoints. We then compute a percentage misfit for the three types of path relative to the great circle arc.

Figure 22 shows the error in path length, relative to the length of the great circle arc, that we accumulate using our approximation in different uniform mesh sizes. We consider meshes comprising cells of sizes 1° latitude by 1° longitude, 2° latitude by 2° longitude, 2.5° latitude by 5° longitude and 5° latitude by 5°longitude. As expected, the finer the mesh the better the approximation to

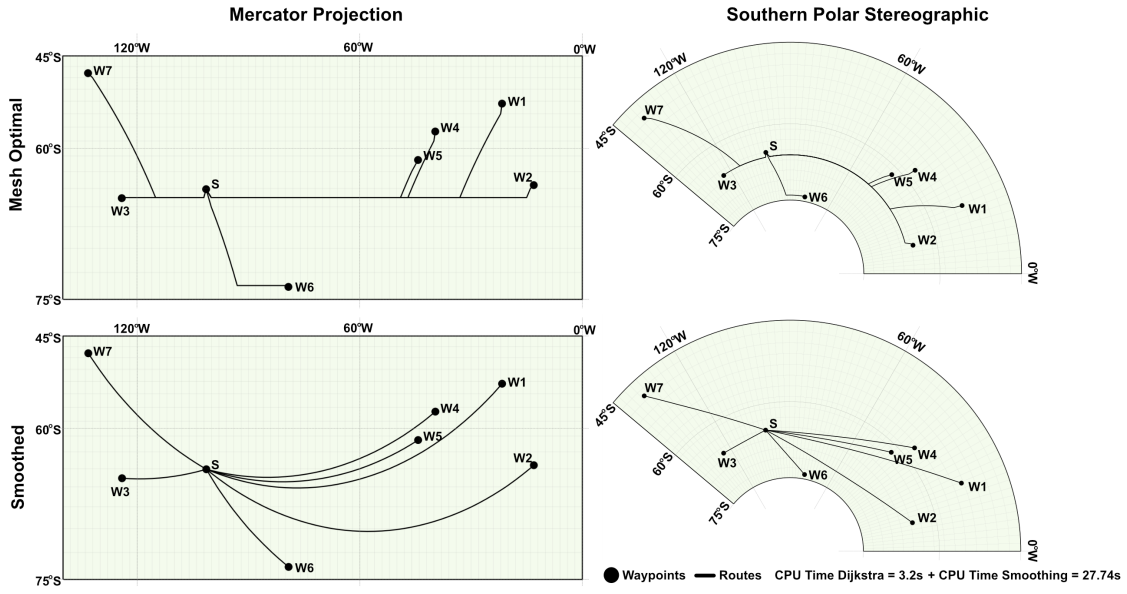

Figure 23: Mesh-optimal and smoothed paths approximating great circle arcs in an unconstrained environment (the apparent curvature of the mesh-optimal path to w7 in the left column is caused by the Mercator projection distortion).

the spherical surface and hence the closer our smoothed paths are to the great circle arcs. The plot shows that for all of the meshes, 95% of the smoothed paths accumulate an error lower than 0.403%, and all of the smoothed paths are within 6% error. By contrast, 95% of rhumb line paths accumulate up to 25% error, with fewer than 5% of rhumb line paths accumulating less than 6.6% error. Considering the mesh-optimal paths constructed in these meshes, the 1x1 mesh results in the best mesh-optimal paths, with 95% of them achieving less than 6.6% error. The 2x2 meshes result in similar quality mesh-optimal paths, with 95% of them lying within 7.3% error. 95% of the paths in the 2.5x5 meshes are within 8.1% error. The 5x5 meshes lead to much higher error: only 65% of the paths are within 6.6% error and at least 2% of the paths accumulate as much as 15% error. All of the Polar Route paths, both mesh-optimal and smoothed, are within 17% of the great circle arc length.

The difference between the error accumulated by smoothed paths versus the mesh-optimal paths in this plot shows the importance of our smoothing method for achieving high quality paths. Smoothing achieves a very close approximation indeed to the corresponding great circle arcs. Figure 23 shows a collection of paths, starting from a start waypoint, s, and visiting a set of 7 other waypoints, constructed in the same unconstrained environment and hence uniform mesh. Mesh-optimal paths are shown in the first row, and their smoothed versions in the second row. Both types of path are shown in Mercator and polar stereographic projections.

As a second comparison with Polar Route we consider the use of Probabilistic RoadMaps (PRMs) (Kavraki & Latombe, 1998), a common method for path-planning in robotics. PRMs construct a graph abstraction of the environment based on a sample of waypoints that is generated

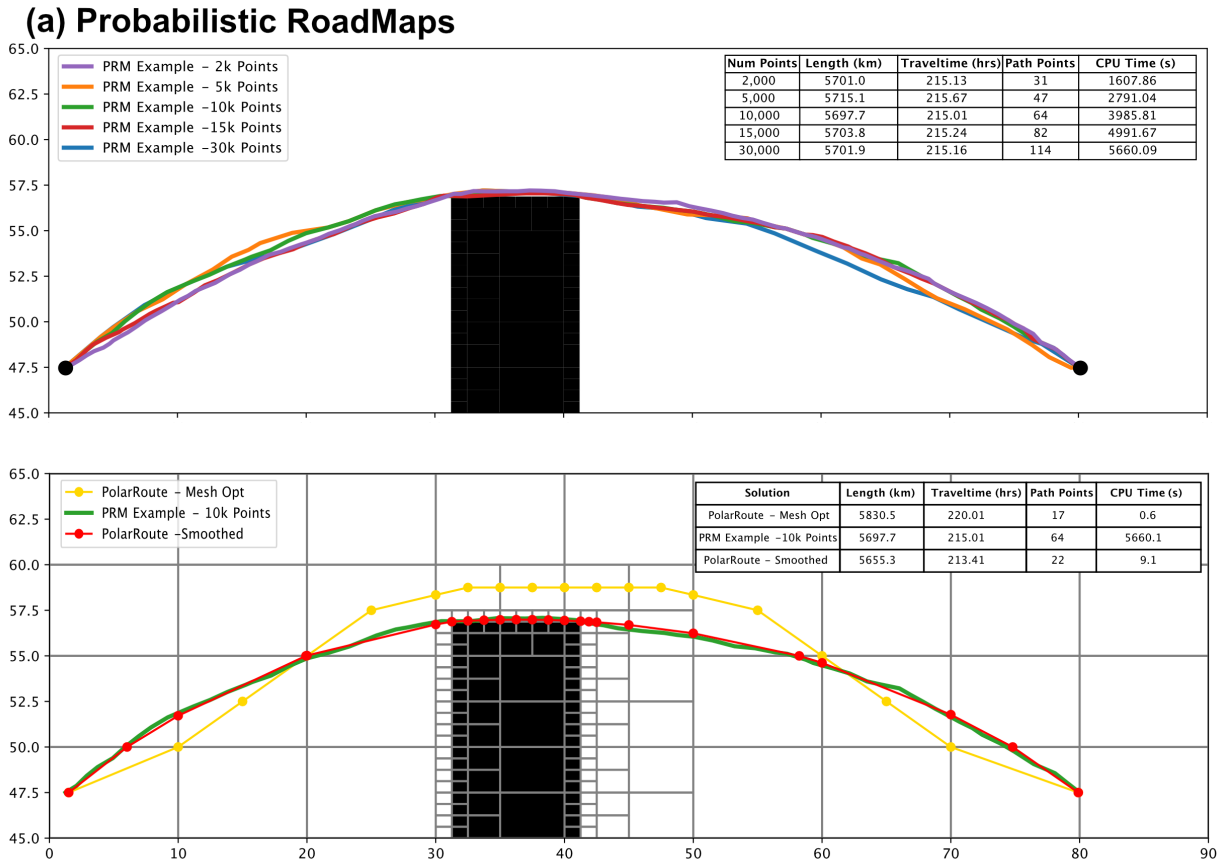

Figure 24: Comparison between Polar Route (mesh-optimal and smoothed routes) and a PRM-generated route in an artificially generated environment. (a) shows PRM paths generated using different numbers of samples. As the number of samples increases, the approximation to the great circle arc improves. (b) The mesh-optimal and smoothed routes generated by Polar Route in a non uniform mesh are shown. The best of the PRM paths is shown (in blue) for comparison.

from a sampling distribution. For each sampled point, w, a neighbourhood is considered and direct edges are created between w and each other point in the neighbourhood where possible. The start and end waypoints are included in the PRM and then a shortest-path algorithm can be applied to extract the optimal path in the PRM.

In the limit, a PRM has two important mathematical properties: sampling covers the entire navigable space and, as a consequence, the path found is the optimal path. Probabilistic roadmaps are suitable for modelling binary environments. These are environments consisting of entirely accessible and entirely inaccessible regions, with no field effects. Standard PRM approaches lack methods to model field effects that impede or resist the progress of the robot leading to varying achievable robot speeds within an accessible region.

It is straightforward to see that a PRM can be used to solve the problem of navigating between the start and destination waypoints in the binary artificial environment shown in Figure 24. The questions of interest are: how many sampled points will be needed for the PRM to construct as good an approximation to the optimal route as Polar Route can find in the same environment; how much CPU time is required by the PRM method and how many changes of bearing will the PRM path comprise by comparison with the path found by Polar Route.

To address these questions, we ran a PRM method with 2000, 5000, 10,000, 15,000 and 30,000 samples from a uniform distribution, and graph construction based on k-nearest neighbour with $k = 30$. The distance metric used is a geodesic calculation on a spherical surface. Polar Route plans in the non-uniform mesh shown in Figure 24 and the smoothed path, shown in red, contains 21 changes of bearing (points on the route excluding the end waypoint) and took 9.1 seconds to generate. The non-deterministic behaviour of PRMs means that the path does not improve monotonically as the number of samples increases.

To construct the best PRM route shown, 10,000 samples are required. The route found contains 63 changes of bearing and takes 3985.81 seconds to generate. Even then, the PRM path is 42.4 km longer than that obtained by Polar Route. Of course, the PRM we have used for this experiment is simple, using the most basic sampling distribution. Better results in the binary case could be obtained using a more sophisticated model, but deriving a distance metric and sampling strategy that could efficiently cope with directional effects and variable achievable speeds would be a research problem on its own.

As well as closely approximating great circle arcs when fields are not present, Polar Route must be able to generate efficient paths when there are strong vector fields and resistances impeding or enhancing the progress of the vessel. Figures 25 and 26 show examples of paths in environments that are increasingly constrained. The figures show non-uniform meshes generated using Gaussian Random Fields. In the unconstrained environment with exclusion zones (the top row in both figures) the vessel can travel at 27.5 km/h in non-excluded cells. The figures show the successive additions of vectors (second row), scalar fields modelling non-uniform resistances (third row) and both scalar and vector fields (final row). The total CPU times reported are the times required to compute all 7 paths shown in each of the plots and are the sum of the mesh-optimal and smoothing times (unless stated otherwise). It can be observed that the construction of the mesh-optimal paths accounts for approximately 10% of the total CPU time and, in these examples, smoothing is about 2.5 times more expensive in the complex environments than in the simplest environment.

In order to support the discovery of optimal paths that coincide with optimal paths in reality, a desirable property of the mesh is that splitting around land or other exclusion zones does not result in land cells bordering very large cells. Examples can be constructed in which Polar Route chooses a mesh-optimal path that travels on the wrong side of the exclusion zone (relative to where the optimal path lies in reality) to avoid the cost of navigating to the centres of very large cells. This phenomenon can dominate the latitude effect so that the discovered path is sub-optimal. Figure 27 (a) shows how this can occur. The task is to plan the path from a point to the West of the figure at around 12° latitude, to a point on the East side of the large land mass shown. The mesh-optimal path passes to the North of the land mass because travel to the centre of the cell lying South of the land mass would be too expensive to be considered during mesh-optimal path construction. Part (b) shows that, if this cell is subdivided so that the splitting gradient (defined in Definition 8), between the southerly tip of the landmass and the cell to the South of it, is shallow, then the path corresponding to the best path in reality can be found.

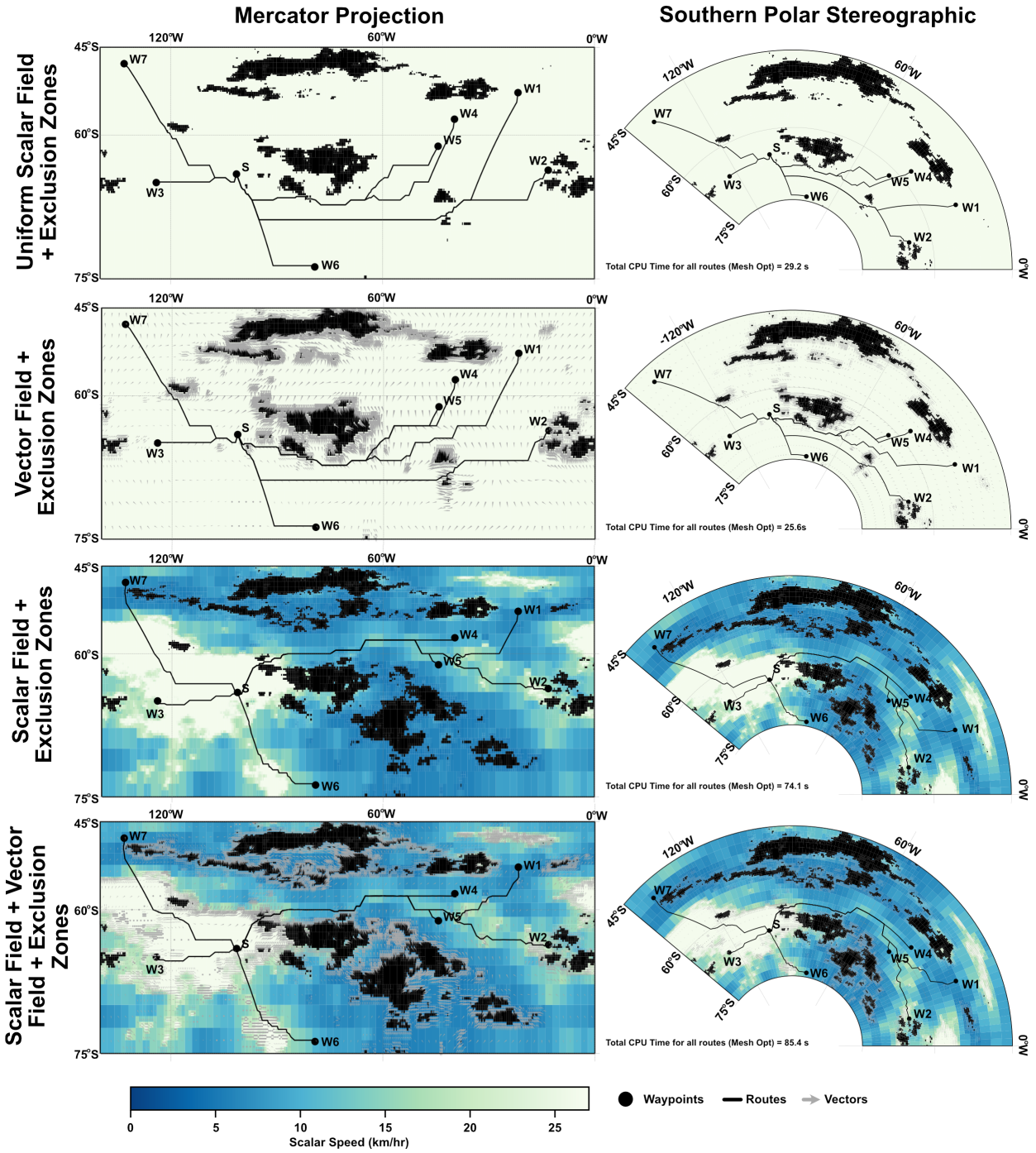

Figure 25: Examples of mesh-optimal routes in different environments. Black lines represent the routes. Black dots the waypoints. Grey regions represent the vector field, and black regions represent exclusion zones. The colour map represents the non-uniform scalar field, with faster speeds in light green and slower in darker blue.

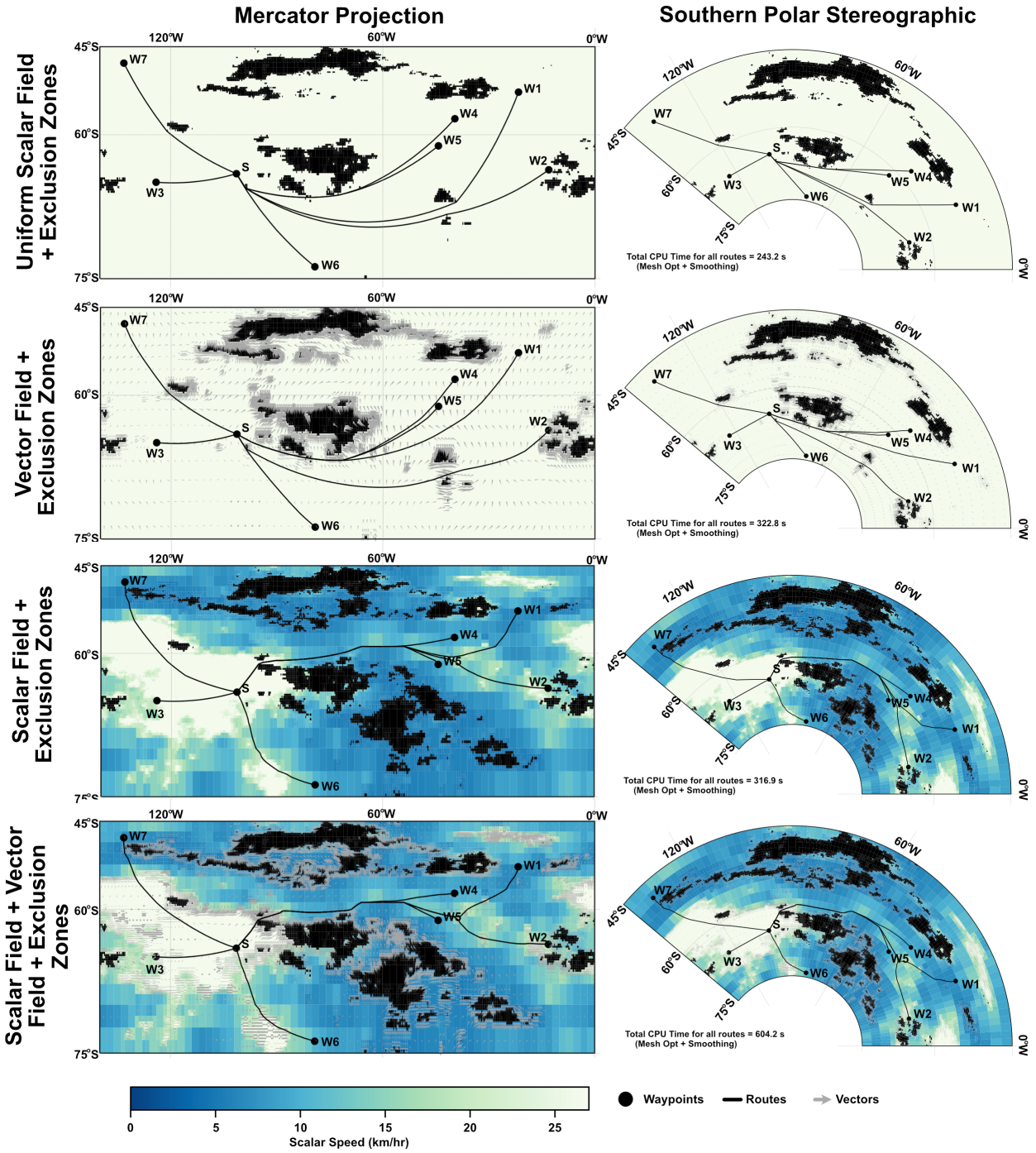

Figure 26: Examples of smoothed routes in different environments. Black lines represent the routes. Black dots the waypoints. Grey regions represent the vector field, and black regions represent exclusion zones. The colour map represents the non-uniform scalar field, with faster speeds in light green and slower in darker blue.

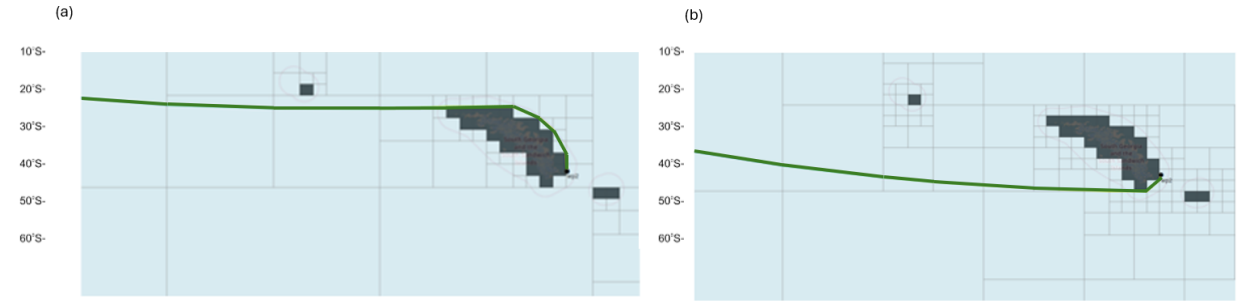

Figure 27: An example in which a mesh with a suitable splitting gradient supports Polar Route in finding optimal paths. As this is a Southern hemisphere example, the optimal path in reality is the one shown in part (b). In (a), the splitting gradient to the South of the land mass is so steep that the mesh-optimal path differs from the optimal path in reality.

Definition 8. *A splitting gradient between adjacent cells is a ratio determining the relative sizes of the two cells.*

With this proviso, mesh-optimal paths successfully navigate around the exclusion zones and fields, and provide the skeleton of the path that is then smoothed to best exploit or mitigate the environmental conditions. The effects of smoothing can be seen in Figure 26. It can be seen that, without the guidance of the mesh-optimal paths, the great circle arcs would cut through the exclusion zones. Finding the compromises that are made in the mesh-optimal path, between distance and environmental conditions, is what makes this optimisation problem so difficult to solve manually.

It can be observed in Figure 26 that the resulting smoothed paths sometimes touch the corners or edges of cells that are inaccessible or difficult to travel through. These are the consequences of clipping to corners or edges caused by the discontinuities in the environment fields that arise at cell boundaries. This feature of visiting corners and edges distinguishes Polar Route from other path-finding methods, such as IcePathFinder (Lehtola et al., 2019), as is seen in Section 8.

Figure 28 shows smoothed paths generated for vessels capable of different maximum speeds, travelling in the presence of exclusion zones, a uniform scalar field determining a constant speed for the vehicle.

The final abstract evaluation focuses on the performance of Polar Route on a sequence of increasingly complex, randomly generated, non-uniform environment meshes generated as Gaussian mixture models, of the kind seen in Section 3. This allows us to investigate the scaling behaviour of Polar Route as environments become more complex and smoothing becomes more challenging. These meshes feature an increasing number of icy atolls, scattered randomly in open water. A scalar field defines the speeds achievable by the vessel in each cell, with maximum speed being attainable in open water and zero speed attainable in ice concentrations above a given threshold of 80%. We use a domain of 0° latitude by 140° longitude, and a starting cell size of 2.5° latitude by 2.5° longitude. For each k in $k = 1..10$ we generated 10 meshes of k atolls with start and end waypoints horizontally displaced, and 10 meshes of k atolls with the waypoints vertically displaced. We then used Polar Route to find paths between each of the pairs of waypoints in these 20 cases. In total,

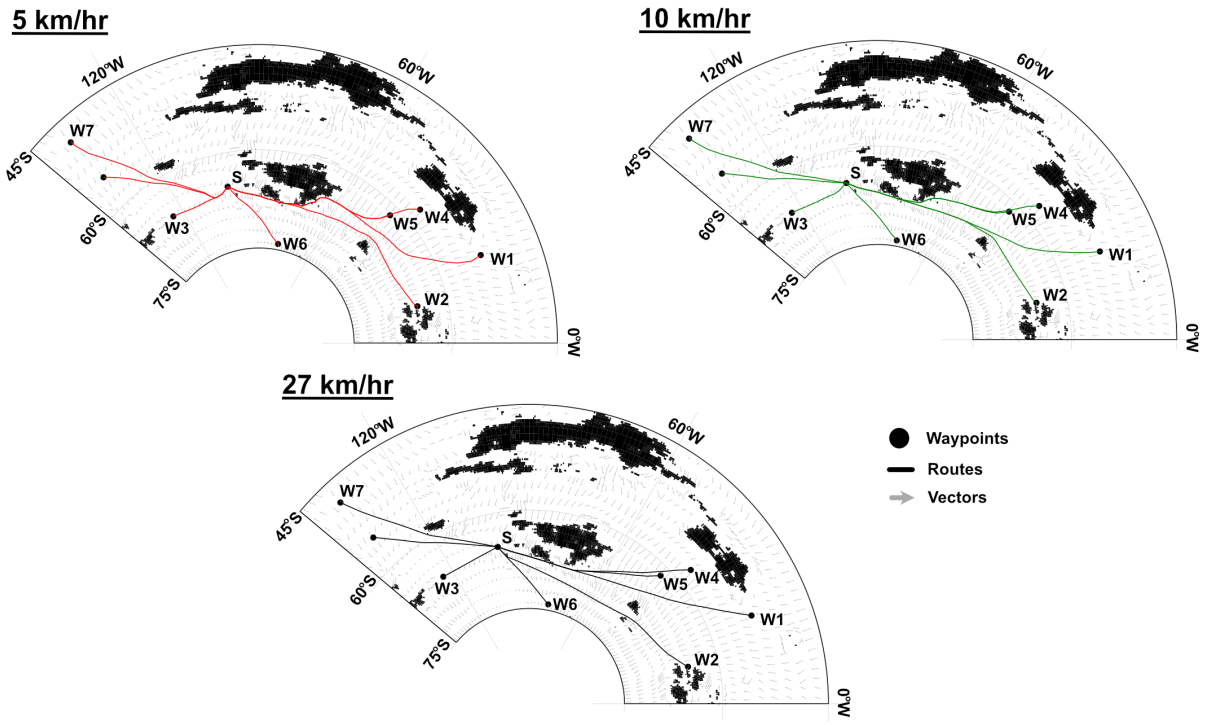

Figure 28: Examples of smoothed routes for vessels travelling at different speeds. Black lines represent the different routes taken at different speeds (5, 10 and 27.5 km/h). Black dots represent the waypoints. Black regions represent exclusion zones. Grey arrows represent the vector field. A uniform scalar field is used. It can be seen that the 5 km/h vessel is much more susceptible to the vector field than the faster vessels.

200 paths were generated and the CPU times to generate the paths, averaged over each group of k atolls, is reported in Figure 29.

Figure 29 shows that, as k increases, the problems in this particular collection become harder for Polar Route until $k = 4$. As k increases beyond 4, the icy atolls tend to clump together, creating easier routing problems. This phenomenon can be seen in natural navigation problems, where the most challenging routing problems arise in areas containing many different obstacles and conditions, requiring the route to weave between them in an efficient way. An example is seen in Figure 4, where the North West passage presents this kind of challenge. All of the problems in the Gaussian Mixture Model collection are solved within 600 seconds, with average performance across the set being less than 100 seconds.

8. Comparison with IcePathFinder

Lehtola *et al.* (2019) describe a path-planning method, called IcePathFinder, for ships operating in constrained regions in the presence of other marine traffic. The only region of operations considered in their paper is the Baltic Sea. The path-planning method consists of an A* search within a uniform, high resolution, grid, followed by a post-processing method to improve the geodesic reality

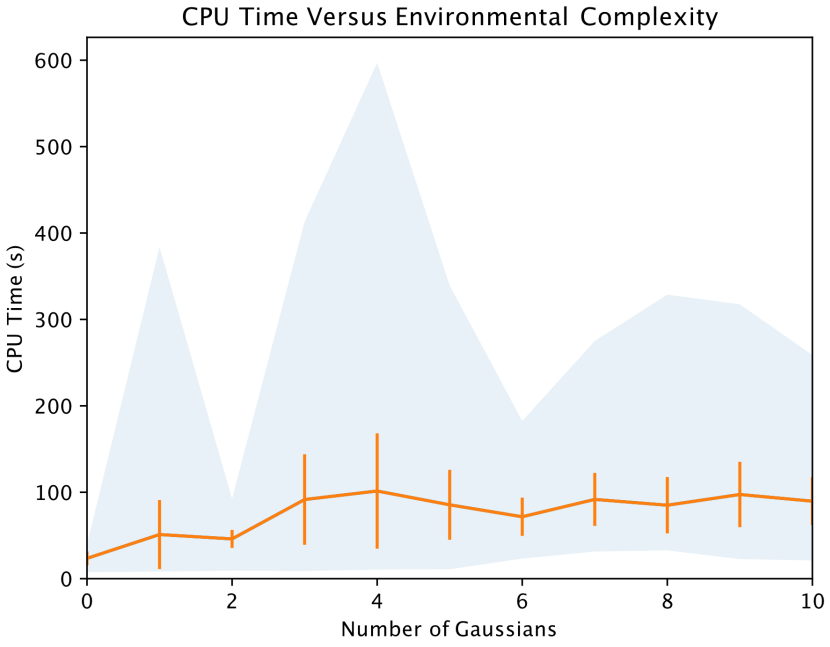

Figure 29: The orange curve represents average CPU time for each of the numbers of Gaussians used. The blue curve supplies the min and max CPU times within each group of 20 path problems. The orange bars show the standard deviations in each group.

of the paths. The method takes into account the impact of sea ice on the speeds achievable by the vessel. This is modelled by associating a maximum attainable speed with each cell in the grid, resulting in a speed map similar to the scalar field information that we record in our mesh. The speed map specifies how the speed of the vessel is modified in a given cell due to ice conditions and the presence of other ships.

Given a speed map represented as a uniform grid, the A* stage of path-planning generates fastest paths between specified waypoints in the grid. In common with other grid-based planning methods, the resulting paths contain grid artifacts which make the paths impractical for navigation. A post-processing phase then passes over the path, considering each triple of adjacent points $\langle s, i, d\rangle$, and removing the point i if it is not needed for obstacle avoidance and a strictly shorter path is possible by going directly from s to d. This process iterates over the whole path until no further efficiency improvements are possible. Finally, the length of the path is calculated as the sum of the *geodesic* distances between the remaining pairs of points.

In order to compare IcePathFinder with Polar Route, we conducted experiments in both an artificial environment and in the Baltic Sea. We generated IcePathFinder solutions using the authors' github codebase[10]. The artificial example consists of open water and land, with start and end waypoints positioned at lat-long coordinates on either side of the land feature. Land is shown as a black rectangle in the figures. The artificial environment is uniformly gridded at the resolution

10. https://github.com/vlehtola/icepathfinder

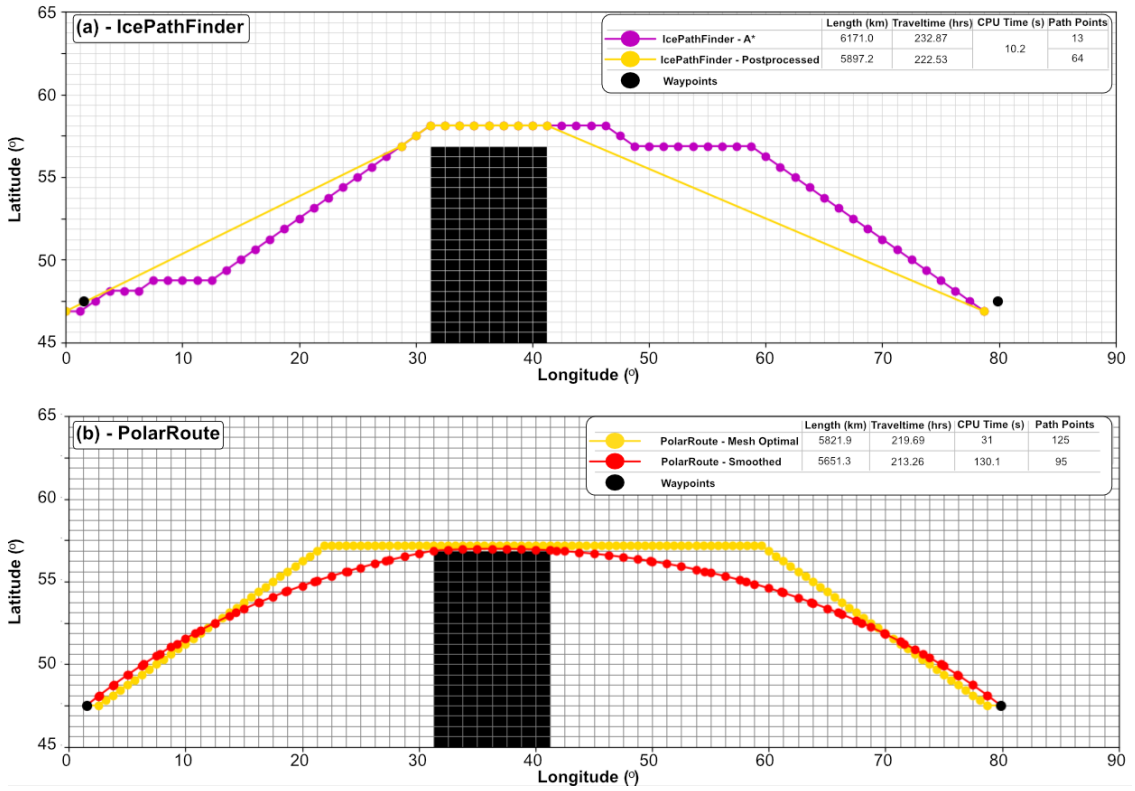

Figure 30: (a) Comparison between the post-processed and A* paths of IcePathFinder in an artificial environment using a uniform mesh.(b) Comparison between the smoothed and mesh-optimal paths of Polar Route in the same uniform mesh.

of 0.625° latitude by 1.25° longitude. In this example, all non-land cells are navigable at a uniform speed (set at 8 knots[11] = 14.816 km/h for the purpose of these experiments).

Figure 30 part (a) shows the IcePathFinder A* route, plotted alongside the IcePathFinder post-processed route. For comparison, in part (b) we run Polar Route using the same mesh and uniform speed.

The A* route constructed by IcePathFinder is 6171 km long, giving a travel time of 232.87 hours. The post-processing stage shortens the length by 273.8 km, giving a length of 5897.2 km and a travel time of 222.53 hours. The mesh-optimal path generated by Polar Route is 5821.9 km long, with a travel time of 219.69 hours. The smoothed route is 5651.3 km long with a travel time of 213.26 hours. Thus, Polar Route finds a path that is 9.27 hours faster and 245.9 km shorter than the IcePathFinder path. Furthermore, IcePathFinder does not correctly visit the start and end waypoints, instead visiting the bottom-left corners of the cells containing them. In this example, IcePathFinder over-extends at one end of the path, and falls short at the other. The larger contribution to the additional distance accumulated by IcePathFinder is that, in both the A* and post-processed paths, IcePathFinder can only change direction at the corners of cells, while Polar

11. A knot is 1.852 km/h.

Route changes direction on boundaries. It can be seen in Figure 30 (b) that Polar Route exploits this opportunity and closely approximates a great circle arc albeit at the expense of many changes in bearing due to the high resolution of the uniform mesh. It can be noted that, in this example, the Polar Route path runs along the top of the blocked region, while IcePathFinder observes a separation forcing it to travel 0.25 degrees north of the top boundary.

Polar Route can find smoothed paths requiring significantly fewer bearing changes when a non-uniform mesh is used. Figure 24 shows how Polar Route performs on the above artificial example in a non-uniform mesh in which open water cells are not subdivided. The largest cells in this mesh are 5° latitude by 10° longitude, and the smallest cells are exactly the same size as the cells in the uniform mesh shown in Figure 30. IcePathFinder cannot use the non-uniform mesh as IcePathFinder relies on uniform cell sizes. This is a significant limitation for long-distance navigation where land and other features are present.

It can be observed that the mesh-optimal path produced using the non-uniform mesh is 8.6 km longer than the mesh-optimal path produced in the uniform mesh, and the smoothed path in the non-uniform mesh is 4 km longer than the smoothed path in the uniform mesh. Thus the non-uniformity of the mesh slightly increases the lengths and travel times of the paths constructed by Polar Route. This is because travelling through larger cells leads to a coarser discretisation of the path. However, as we show in the CPU time comparisons, the non-uniform mesh leads to a highly efficient compaction of the environmental model allowing very fast path construction across large, heterogeneous environments, at the cost of only a small degradation in path quality. Furthermore, the number of bearing changes in the smoothed path is substantially reduced (from 94 in the uniform mesh to 21 in the non-uniform mesh) which is beneficial for the navigator.

To conduct the Baltic Sea experiment, we constructed a high resolution uniform mesh for IcePathFinder using a resolution of approximately 1×1 nautical miles which is $1/60° = 0.0167°$at the equator. Lehtola *et al.* use this resolution, with a uniform mesh of 0.0167° latitude by 0.0167° longitude. For Polar Route, we generated a non-uniform mesh with maximum cell size of 0.267° latitude by 0.267° longitude and minimum cell size equal to those in the uniform mesh used by IcePathFinder. Both meshes are constructed using the elevation (global GEBCO) and sea ice concentration (HELMI) datasets used in the IcePathFinder codebase as reported by Lehtola *et al.* (2019).

The start and destination waypoints are placed at lat-long coordinates (58.241,18.219) and (65.577,24.328) respectively, spanning an area of more than 8×6 degrees (hence a region of size in the order of 500,000 square kilometers). Figure 31 shows the differences between the corrected path of IcePathFinder, found in the uniform mesh, and the smoothed path found by PolarRoute in the non-uniform mesh. While the paths are similar, the path generated by PolarRoute transits through lower ice concentrations and therefore has a shorter overall transit time. Furthermore, the geodesics used by IcePathFinder to estimate the length of the path, sometimes cross inaccessible cells, as we show in the two artificial examples given in Figures 32 and 33. These examples demonstrate that geodesics can underestimate the path lengths and travel times of paths. We therefore calculate the lengths of both the IcePathFinder path and the Polar Route path as the sum of the lengths of the rhumb lines comprising the paths. Although it takes Polar Route three times longer to compute its path, Polar Route saves 10.8 hours of travel time, and generates a path that is 5.3 km shorter than that found by IcePathFinder. Although the saving in distance is very small, the saving in travel time is important. The expanded red box in Figure 31 shows how Polar Route carefully navigates the lower ice concentrations, while IcePathFinder ploughs through the adjacent area of high sea ice

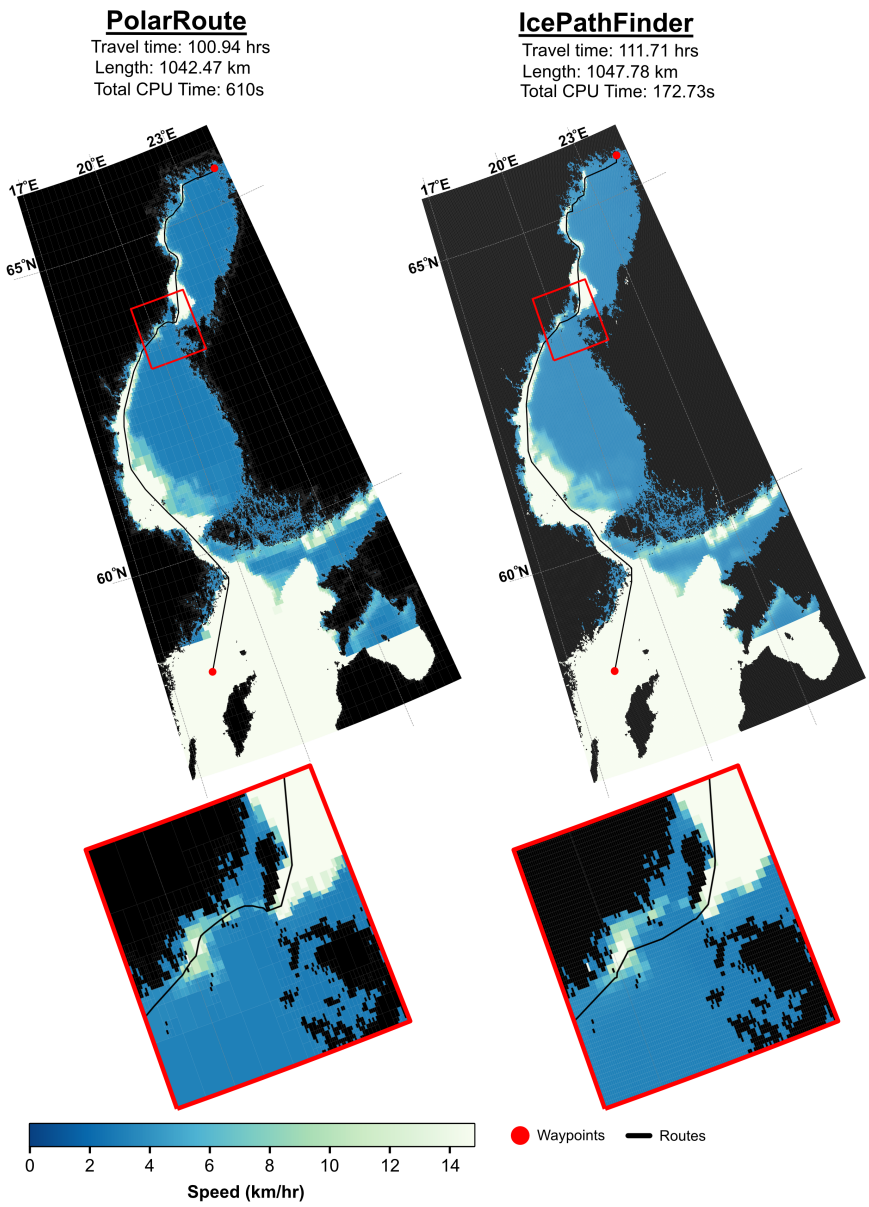

Figure 31: Comparison between the final paths of PolarRoute (left) and IcePathFinder (right) in the Baltic Sea region shown in Lehtola (2019).

concentration. This significantly slows the ship (and significantly impacts on fuel use), contributing to the longer travel time of the IcePathFinder path.

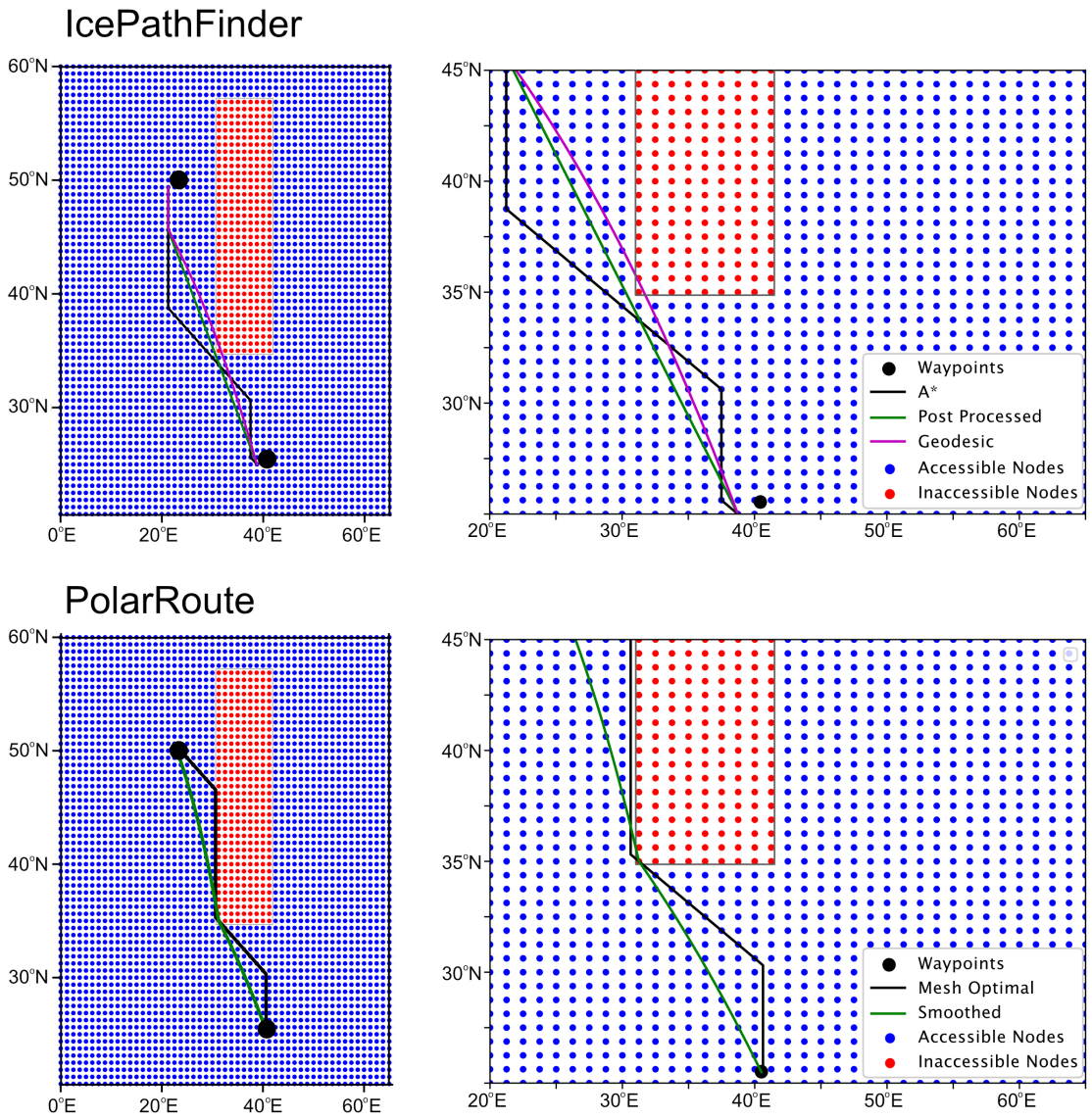

Figure 32: A vertical case in which IcePathFinder uses geodesics that sometimes enter inaccessible areas, while Polar Route never visits cells it has identified as inaccessible.

9. Conclusions

We present an automated method for path-planning in a variant of Zermelo's navigation problem (1931), in which a vessel is navigating on a spherical surface with obstacles, subject to vector and scalar field effects presented as tiled regions of constant effect. The method is called Polar Route, and is suitable for marine navigation of a vessel in complex marine environments in which there are surface currents, seasonal sea ice and variable ocean depths, among other factors.

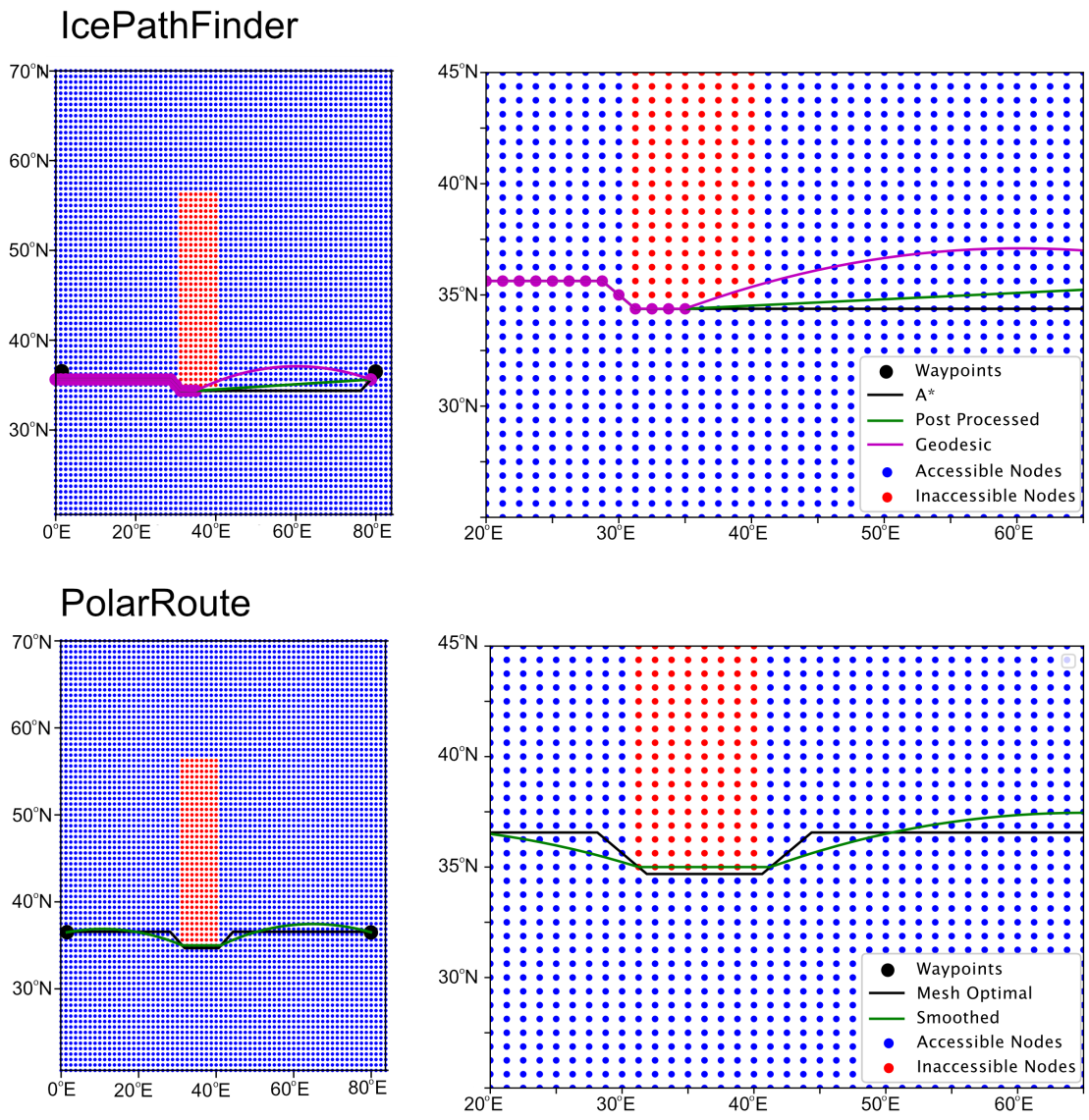

Figure 33: A horizontal case in which IcePathFinder uses geodesics that sometimes enter inaccessible areas, while Polar Route never visits cells it has identified as inaccessible.

The problem solved by Polar Route is similar to the problem of finding shortest paths in triangular meshes, described by Mitchell and Papadimitriou (1991). Polar Route does not use a triangulation, but a non-uniform grid-based decomposition of the environment, and our surface is spherical rather than planar. In addition, and in contrast to the majority of work in this area of research, we include both scalar (with weighting effects similar to those used in some of the related work discussed in Section 5) and vector fields. Vector fields significantly complicate the problem, because the direction of travel across a region becomes critical in determining the shortest path.

Polar Route works in two phases: first, a baseline path is constructed using Dijkstra's algorithm on a coarse approximation of the spherical surface. The grid-based Dijkstra method is combined with an optimisation method to determine where to cross the boundaries between latitude-scaled cells in a combined scalar and vector field. These equations are solved using Newton's method. Second, the resulting mesh-optimal paths are smoothed, using a mathematically rigorous smoothing method. We have demonstrated the utility of our two-stage method in a variety of different conditions (with and without exclusion zones, vector fields and scalar fields, in a range of differently cluttered areas, and over long and short distances). Given an environmental mesh as input, the path-planning method described here has been shown to generate conveniently navigable paths that minimise travel time to within an accuracy that depends on the mesh size.

Continuous methods have been proposed which rely on the environment being characterised by analytic and differentiable functions. Polar Route relies on a mesh constructed using GIS data which is sampled at points and hence discontinuous. In general, there is no natural analytic form for the data fields, and often insufficient data to allow smooth interpolations.

Comparison between Polar Route and IcePathFinder, a long-distance marine path-planner solving a similar problem to ours, shows that Polar Route produces better quality paths than those of IcePathFinder and never produces paths that collide with exclusion zones or violate the scalar and vector constraints of the environment. By contrast, because the IcePathFinder smoothing method works by removing crossing points rather than by recalculating them, it sometimes find paths that contain collisions with exclusion zones. Furthermore, IcePathFinder only works with a uniform mesh, which limits the scale and complexity of the environments it can plan for. The use in IcePathFinder of geodesics to estimate path lengths and travel times sometimes violates the modelled environmental constraints.

Comparision with a Probabilistic Roadmap method showed that – while PRMs in theory guarantee optimal paths in binary environments – these can take considerable CPU time to find and cannot, in practice, achieve solutions as close to optimal as those generated by the two-stage method of Polar Route. Of course the performance of the PRM method depends on the sampling strategy used, and importance sampling is much better than uniform sampling for selecting useful points. However, in order to define a suitable importance sampling strategy, the general trajectory of the path itself must be known, and this is not known until at least one phase of path-planning is completed.

Much of the work in grid-based path planning relies on high resolution meshes in order to capture a suitable level of environmental detail for the path-planning application. Polar Route uses a non-uniform mesh in which areas featuring unvarying data profiles are modelled by large cells containing constant scalar and vector fields. Polar Route is required to plan long distance routes as well as shorter ones (still typically hundreds of kilometers long) in complex environments. Smoothed paths generated by Polar Route comprise sequences of rhumb lines with changes in bearing at the waypoints between them. Navigation over the distances between the waypoints can be considered with respect to uniform and constant vector and scalar fields, since the control problem of following a rhumb line is confronted in real time by the vessel's navigator or helmsman. Thus, as shown by our results, a mesh-based approximation of the environment, following the description in Section 2.1, is sufficient for generating high quality routes in these situations.

The costly parts of Polar Route are in the computation of edge weights during graph construction and in the iterative path-smoothing process. The construction of mesh-optimal paths, given the accessibility graph, is not usually a bottleneck because of the non-uniformity of the mesh. This keeps

the number of cells small relative to the area of the globe being covered. We use a lazy strategy for the calculation of edge costs in the graph, which also mitigates the effect of large graph sizes. We are using a standard implementation of Dijkstra's algorithm, so the performance of the mesh-optimal path construction – should we discover scaling problems – could perhaps be improved by switching to A* search, or by exploiting an optimised data structure such as an untidy priority queue (Yatziv et al., 2006). The smoothing process requires that every crossing point be re-evaluated, on every smoothing iteration, using the refined model of curvature. Since thousands of iterations may be required to achieve path convergence, the process is expensive, and complex paths can take a few minutes to construct. Nonetheless our results show that the waiting time is acceptable and the approach achieves smooth paths with good properties: they never cross obstacles and they spend as little time as possible in areas that impede the progress of the vessel.

As a topic for future work, it would be of interest to enable Polar Route to optimise metrics other than travel time. In particular, navigators are often interested in reducing fuel consumption along a route while still reaching a destination before a set deadline. In Polar Route, Dijkstra's algorithm can be applied using an objective function in which the weights are modified via a further function such as determination of fuel cost given the conditions present in the cells travelled through. In such a case, fuel cost would be optimised *subject to* individual edges being selected for optimal travel time. Optimising fuel directly would require three degrees of freedom to be considered: the speed in each cell and the crossing point. Proper implementation of this objective would involve a substantial change to the crossing point calculation. Similarly other objectives, such as minimising discomfort to those on board, would lead a navigator to avoid heading into difficult weather conditions, or sailing parallel to high waves, but these cannot yet be properly captured as objective functions in Polar Route.

Polar Route has been designed to work within a plan-replan execution framework, where the approach is to plan in snapshots of the environment and then replan when conditions change. An alternative approach would be to model the temporal dynamics of the environment and plan for expected changes in the ice condition and weather.

Finally, the proper treatment of winds and waves requires a shift away from the model of the vessel as a particle. Winds and waves can act on the side of the vessel, depending on the bearing of the vessel, and therefore require a more complex specification of the vessel and additional modelling complexity. Furthermore, wind, in particular, can lead to rapidly changing conditions and the model of the environment further motivating the inclusion of a temporal dimension. It might not be possible to capture all of the necessary relationships that could affect navigation by means of a single combined scalar field and a single combined vector field. These questions raise considerable extra complexity and will be addressed in future work.

Acknowledgements

We would like to acknowledge UKRI-NERC for funding the work reported in this paper. We thank the anonymous reviewers of an earlier draft of this paper for their detailed and helpful comments.

Appendix A. Software Resources

The methods and plots in this paper are provided in the route planning software Package PolarRoute, version 0.5, found at:

`https://github.com/antarctica/PolarRoute/releases/tag/v0.5` . The IcePathFinder paths were generated using the code provided by Ville Lehtola at:
`https://github.com/vlehtola/icepathfinder`. A much earlier, unpublished, version of the work described in this paper can be found on ArXiv `https://arxiv.org/pdf/2209.02389`.

Appendix B. Auxilliary Functions Supporting the Smoothing Algorithm

The smoothing algorithm described in Section 6.3 depends on a collection of basic operations as follows (error cases are not considered in these descriptions):

- **insert(x,L1,L2)**: Inserts the list, $L1$, into the list, $L2$, at position index x (the first element of $L1$ will appear at position x after the insertion). The first element of a list is at index position 0.
- **remove(x,L)**: Removes the element at position x from list L.
- **length(L)**: Returns the length of list L.

The following functions operate with the basic data structures of the geophysical representation. It is assumed that the space is represented with a mesh of *cells*, that *points* are represented by a latitude and longitude on the surface of the sphere, that *edges*, *ap*, are characterised by two adjacent cells (sharing a boundary or single common corner), *ap.start* and *ap.end*, and that an edge has a crossing point, *ap.crossing*, on the shared boundary between the adjacent cells (which could be a common corner). Furthermore, it is assumed that each edge has a direction, which is diagonal if the shared boundary is a single point, and is vertical (north or south) if the shared boundary is oriented east-west, and horizontal otherwise. Three constants are parameters used in the algorithm: *MERGESEP*, which determines how close points become before we merge them into the same point, *CONVERGESEP* which determines the minimum change in the position of a point that is considered to have been a significant change for the smoothing of the path, and *WORSE* which is a threshold on how much worse the conditions (scalar field or vector field) can be in a cell, C, compared with an alternative, A, for it to be considered possible to consider passing through C instead of A.

- **dist(x,y)**: Returns the distance between the points x and y, which is the length of the shorter great circle arc that connects them.
- **findEdge(cellA,cellB)**: Returns the edge that connects the two cells, $cellA$ and $cellB$. If there is no edge connecting the cells (or its weight is infinite), it returns **null**.
- **isDiagonal(ap)**: Returns true if the edge *ap* is diagonal (connects cells that share only a common corner).
- **inside(x,cell)**: Returns true if the point x lies inside the cell *cell* (including its boundary).
- **clipTo(cell, x)**: Returns the point inside *cell* that is closest to the point x. Note that if x lies on a line that is an extension of one of the sides of *cell*, then the point that is returned will be a corner of *cell*. Also, if x is inside *cell*, then the function will return x.

- **nearestNeighbour(cellA,cellB,x)**: Returns the cell in the mesh that shares a boundary with *cellA* and has an edge on the line that extends the common boundary of *cellA* and *cellB* (and on which the point x lies) in the direction of x. If x lies inside *cellA* or there is no cell that satisfies these requirements, it returns **null**.
- **selectSide(firstpoint,midpoint,lastpoint,ap)**: Assuming that *midpoint* is the common corner of the two cells in the diagonal edge *ap*, returns the cell that shares a boundary with both *ap.start* and *ap.end* on the same side of *midpoint* as the shorter great circle arc passing between *firstpoint* and *lastpoint*. In the case that *midpoint* is within *CONVERGESEP* of the arc, then it returns **null**.
- **blocked(targetA,cellA,cellB)**: Returns true if *targetA* is **null** (meaning it is not accessible) or if the conditions in the cell are relatively worse than those in either *cellA* or *cellB* by the relevant threshold, *WORSE*. If the maximum speeds in the target cell and cells A and B are S_T, S_A and S_B, respectively, the relative impact of entering the new cell is determined by checking whether

$$\frac{S_A - S_T}{S_T} > WORSE$$

or

$$\frac{S_B - S_T}{S_T} > WORSE.$$

Finally, **NewtonSmooth(firstpoint,midpoint,lastpoint,ap)** is a method that returns the new crossing point on the boundary between *ap.start* and *ap.end*, when crossing from the first point, fp, on the boundary of *ap.start* to the last point, lp, on the boundary of *ap.end*. The mid point, mp, which is the current crossing point *ap.crossing*, is used as the initial value for Newton's method. The orientation of *ap* determines whether this will invoke horizontal or vertical smoothing. In the horizontal case, the selection of the crossing point implements the derivation given in Appendix C. In the vertical case, the selection implements the derivation given in Appendix D.

Appendix C. Derivation of the Minimal Smoothed Path Travel Time in the Horizontal Case

In the method described in Section 6.1.1, each cell is treated as having a constant width which is scaled by the latitude of its centre point. During smoothing we introduce a latitude correction so that the choice of y respects the curvature of the sphere. We therefore replace x with $x\,cos\,\theta$, where $\theta = \frac{y}{2R} + \lambda$. Here, θ is the latitude of the crossing point, and λ is the latitude of the entry point into the cell pair. Using $z_l = x\,cos\,\theta$, the speed of the vessel in the entry cell is:

$$s_l^2 t_1^2 = (z_l - t_1 u_1)^2 + (y - t_1 v_1)^2 \tag{35}$$

As in Section 6.1.1, b (the horizontal travel in the exit cell) is equal to $Y - y$. We can form a corresponding equation for the exit cell, using $\psi = \frac{-b}{2R} + \tau$, the latitude of the crossing point (where τ is the latitude of the exit point) and $z_r = a\,cos\,\psi$, the horizontal distance travelled in the exit cell.

We define:

$$d_1^2 = z_l^2 + y^2 \tag{36}$$

$$d_2^2 = z_r^2 + (Y - y)^2 \tag{37}$$

$$D_1 = z_l u_1 + y v_1 \tag{38}$$

$$D_2 = z_r u_2 + (Y - y) v_2 \tag{39}$$

The terms C_1, C_2, X_1, X_2, t_1 and t_2 are as defined in Section 6.1.1, with the D terms in those equations replaced by Equations 38 and 39.

Quadratic equations for travel in the entry and exit cells can then be constructed and solved as in Section 6.1.1.

Differentiating 35, we obtain:

$$s^2 t_1 \partial t_1 = (z_l - t_1 u_1)(\partial z_l - u_1 \partial t_1) + (y - t_1 v_1)(1 - v_1 \partial t_1) \tag{40}$$

After cancelling and factorising:

$$\partial t_1 (s^2 t_1 + (z_l - t_1 u_1) u_1 + v_1 (y - t_1 v_1)) = y - t_1 v_1 + \partial z_l (z_l - t_1 u_1) \tag{41}$$

and after rearranging terms:

$$\partial t_1 ((s^2 - u_1^2 - v_1^2) t_1 + z_l u_1 + y v_1) = y - t_1 v_1 + \partial z_l (z_l - t_1 u_1) \tag{42}$$

Equation 42 can be rewritten using Equations 11, 13 and 19 as follows:

$$\partial t_1 (C_1 t_1 + D_1) = y - t_1 v_1 + \partial z_l (z_l - t_1 u_1) \tag{43}$$

Using Equation 17, we have:

$$\partial t_1 X_1 = y - t_1 v_1 + \partial z_l (z_l - t_1 u_1) \tag{44}$$

$$\partial t_1 = \frac{y - t_1 v_1 + \partial z_l (z_l - t_1 u_1)}{X_1} \tag{45}$$

An expression for ∂t_2 can be derived through similar steps.

$$\partial z_l = \frac{-x \, sin \, \theta}{2R} \tag{46}$$

$$\partial z_r = \frac{-a \, sin \, \psi}{2R} \tag{47}$$

$$\partial D_1 = \partial z_l u_1 + v_1 \tag{48}$$

$$\partial D_2 = \partial z_r u_2 - v_2 \tag{49}$$

Generalising Equation 35 to the exit cell, we can write:

$$t_2^2 s^2 = (z_r - t_2 u_2)^2 + (Y - y - t_2 v_2)^2 \tag{50}$$

Differentiating Equation 50, we obtain:

$$s^2 t_2 \partial t_2 = (z_r - t_2 u_2)(\partial z_r - u_2 \partial t_2) + (Y - y - t_2 v_2)(-1 - v_2 \partial t_2) \tag{51}$$

which can be written:

$$\partial t_2(s^2 - u_2^2 - v_2^2)t_2 + D_2 = Y - y + t_2 v_2 + \partial z_r(z_r - t_2 u_2) \tag{52}$$

and therefore:

$$\partial t_2 = \frac{Y - y + t_2 v_2 + \partial z_r(z_r - t_2 u_2)}{X_2} \tag{53}$$

As in Section 6.1.1, we require that:

$$\frac{y - t_1 v_1 + \partial z_l(z_l - t_1 u_1)}{X_1} + \frac{Y - y + t_2 v_2 + \partial z_r(z_r - t_2 u_2)}{X_2} = 0 \tag{54}$$

By summing and rearranging terms, we obtain:

$$(X_1 + X_2)y - t_1 X_2 v_1 + t_2 X_1 v_2 - Y X_1 + \partial z_r(z_r - t_2 u_2)X_1 + \partial z_l(z_l - t_1 u_1)X_2 = 0 \tag{55}$$

so $F(y)$, the function we wish to minimise, is:

$$F(y) = (X_1 + X_2)y - t_1 X_2 v_1 + t_2 X_1 v_2 - Y X_1 + \partial z_r(z_r - t_2 u_2)X_1 + \partial z_l(z_l - t_1 u_1)X_2 \tag{56}$$

Equation 56 can be rewritten, using Equations 19 and 20, as:

$$\begin{aligned} F(y) = (X_1 + X_2)y - \tfrac{(X_1 - D_1)X_2 v_1}{C1} + \tfrac{(X_2 - D_2)X_1 v_2}{C_2} - Y X_1 + \\ \partial z_r(z_r - \tfrac{(X_2 - D_2)u_2}{C_2})X_1 + \partial z_l(z_l - \tfrac{(X_1 - D_1)u_1}{C_1})X_2 \end{aligned} \tag{57}$$

To form the derivative of Equation 57 we require the derivatives of ∂z_l and ∂z_r. Since $\partial\theta = \partial(\frac{y}{R} + \lambda) = 1/R$ and $\partial\psi = \partial(-\frac{Y-y}{R} + \tau) = \frac{1}{R}$, we have:

$$\partial\partial z_l = -\frac{x cos\theta}{R}\partial\theta = -\frac{z_l}{R^2} \tag{58}$$

$$\partial\partial z_r = -\frac{a sin\psi}{R}\partial\psi = -\frac{z_r}{R^2} \tag{59}$$

The derivative of Equation 57 is then:

$$\begin{aligned} \partial F(y) = (X_1 + X_2) + y(\partial X_1 + \partial X_2) \\ - \tfrac{v_1}{C1}(\partial X_2(X_1 - D_1) + X_2(\partial X_1 - \partial D_1)) \\ + \tfrac{v_2}{C_2}(\partial X_1(X_2 - D_2) + X_1(\partial X_2 - \partial D_2)) - Y\partial X_1 \\ - \tfrac{z_r}{4R^2}(z_r - \tfrac{(X_2 - D_2)u_2}{C_2})X_1 - \tfrac{z_l}{4R^2}(z_l - \tfrac{(X_1 - D_1)u_1}{C_1})X_2 \\ + \partial z_r(\partial z_r - \tfrac{u_2}{C_2}(\partial X_2 - \partial D_2))X_1 \\ + \partial z_l(\partial z_l - \tfrac{u_1}{C_1}(\partial X_1 - \partial D_1))X_2 \\ + \partial z_r(z_r - \tfrac{(X_2 - D_2)u_2}{C_2})\partial X_1 + \partial z_l(z_l - \tfrac{(X_1 - D_1)u_1}{C_1})\partial X_2 \end{aligned} \tag{60}$$

where ∂X_1 and ∂X_2 are defined as follows.

$$\begin{aligned} \partial X_1 &= \tfrac{D_1\partial D_1 + C_1(z_l\partial z_l + y)}{X_1} \\ &= \tfrac{(\partial z_l u_1 + v_1)D_1 + C_1(y + z_l\partial z_l)}{X_1} \\ &= \tfrac{D_1 v_1 + C_1 y + \partial z_l(D_1 u_1 + C_1 z_l)}{X_1} \end{aligned} \tag{61}$$

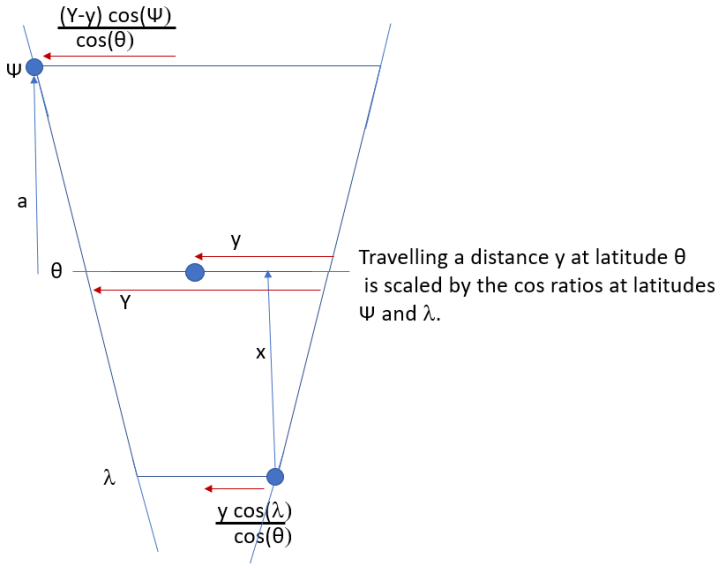

Figure 34: Using cosine ratios to approximate horizontal distance in a vertical alignment.

$$\begin{aligned} \partial X_2 &= \frac{D_2(\partial z_r u_2 - v_2) + C_2(z_r \partial z_r - Y + y)}{X_2} \\ &= \frac{-v_2 D_2 - C_2(Y-y) + \partial z_r(D_2 u_2 + C_2 z_r)}{X_2} \end{aligned} \tag{62}$$

Using Equations 56 and 60, we can use Newton's method to arrive at a value of y that makes $F(y) = 0$. Due to the new latitude correction, this value of y might visit latitudes outside the horizontal boundaries of the two cells. This results in the addition of a suitable horseshoe pattern, which will result in new cell pairs to be smoothed on later iterations.

During smoothing, it can happen that the travel time is zero, or infinitesimal, in one or other cell. If the two points concerned are *fp* and *mp*, no action is taken on this iteration and we simply advance to the next triplet of points. On the next iteration of smoothing along the path, this pair of close points will be *mp* and *lp*, and will be addressed in the following way. Figure 17 shows the situation in which the close points are *mp* and *lp*. The situation is resolved by removing mesh-optimal path edges *ap* and ap', and replacing them with a single edge ap''. This new edge, which was not previously part of the mesh-optimal path, now connects the end points of *ap* and ap'. Since all three cells in the figure are accessible, ap'' must exist. The points *mp* and *lp* no longer lie on the path, as they are replaced by the single crossing point for the edge ap''. This crossing point becomes the new *mp*, and *lp* is identified as the exit point for the target cell at the end of the edge ap''. The point *fp* remains unchanged.

Appendix D. Derivation of the Minimal Smoothed Path Travel Time in the Vertical Case

When smoothing from an entry point to an exit point in a vertical alignment, the path passes through a crossing point at a latitude, θ, between the entry and exit point latitudes, λ and ψ respectively. Suppose that the vertical distance from the entry point to the crossing point is x that the vertical distance from the crossing point to the exit point is a. When travelling from the entry point to the crossing point, at a latitude θ, the horizontal distance travelled, d, depends on both the longitudinal separation of the entry point and the crossing point, and θ. We approximate d using cos ratios as shown in Figure 34. The figure shows two longitudinal lines and three latitudes, in a vertical adjacent cell pair arrangement. The entry point is at latitude λ, the exit point is at latitude ψ, and the crossing point is at latitude θ. As shown, a distance travelled at the crossing point latitude must be scaled to give the corresponding distances at the other two latitudes. Two ratios are required, one to approximate the horizontal distance travelled at the entry latitude, and the other to approximate the horizontal distance travelled at the exit latitude. We call these ratios r_1 and r_2 respectively, defined in Equations 63 and 64.

$$r_1 = \frac{cos\lambda}{cos\theta} \tag{63}$$

$$r_2 = \frac{cos\psi}{cos\theta} \tag{64}$$

$$d_1^2 = x^2 + (r_1 y)^2 \tag{65}$$

$$d_2^2 = a^2 + (r_2(Y - y))^2 \tag{66}$$

$$D_1 = xu_1 + r_1 v_1 y \tag{67}$$

$$D_2 = au_2 + r_2 v_2 (Y - y) \tag{68}$$

$$\partial D_1 = r_1 v_1 \tag{69}$$

$$\partial D_2 = -r_2 v_2 \tag{70}$$

The travel times in the start and end cells, t_1 and t_2, are given as:

$$s_l^2 t_1^2 = (x - t_1 u_1)^2 + (r_1 y - t_1 v_1)^2 \tag{71}$$

$$s_r^2 t_2^2 = (a - t_2 u_2)^2 + (r_2(Y - y) - t_2 v_2)^2 \tag{72}$$

Differentiating Equations 71 and 72, we obtain:

$$s_l^2 t_1 \partial t_1 = -(x - t_1 u_1) u_1 \partial t_1 + (r_1 y - t_1 v_1)(r_1 - v_1 \partial t_1) \tag{73}$$

$$\partial t_1(s_l^2 t_1 + (x - t_1 u_1)u_1 + v_1(r_1 y - t_1 v_1)) = r_1(r_1 y - t_1 v_1) \tag{74}$$

$$\partial t_1((s_l^2 - u_1^2 - v_1^2)t_1 + x u_1 + r_1 v_1 y) = r_1(r_1 y - t_1 v_1) \tag{75}$$

$$\partial t_1(C_1 t_1 + D_1) = r_1(r_1 y - t_1 v_1) \tag{76}$$

$$\partial t_1 X_1 = r_1(r_1 y - t_1 v_1) \tag{77}$$

$$\partial t_1 = \frac{r_1(r_1 y - t_1 v_1)}{X_1} \tag{78}$$

and, by similar reasoning:

$$\partial t_2 = -\frac{r_2(r_2(Y - y) - t_2 v_2)}{X_2} \tag{79}$$

As before, we require $\partial t_1 + \partial t_2 = 0$.

We differentiate X_1 and X_2 to obtain:

$$\partial X_1 = \frac{r_1(D_1 v_1 + r_1 C_1 y)}{X_1} \tag{80}$$

and

$$\partial X_2 = \frac{-r_2(D_2 v_2 + r_2 C_2(Y - y))}{X_2} \tag{81}$$

We can now write:

$$\partial t_1 + \partial t_2 = r_1(\frac{r_1 y - t_1 v_1}{X_1}) + r_2(\frac{r_2(y - Y) + t_2 v_2}{X_2}) = 0 \tag{82}$$

This can be rewritten as follows using Equations 19 and 20:

$$F(y) = (r_2^2 X_1 + r_1^2 X_2)y - \frac{r_1(X_1 - D_1)X_2 v_1}{C_1} + \frac{r_2(X_2 - D_2)X_1 v_2}{C_2} - r_2^2 Y X_1 \tag{83}$$

which, differentiated, gives:

$$\begin{aligned} \partial F(y) = {} & (r_2^2 X_1 + r_1^2 X_2) + y(r_2^2 \partial X_1 + r_1^2 \partial X_2) \\ & - \tfrac{r_1 v_1}{C_1}((\partial X_1 - \partial D_1)X_2 + (X_1 - D_1)\partial X_2) \\ & + \tfrac{r_2 v_2}{C_2}((\partial X_2 - \partial D_2)X_1 + \partial X_1(X_2 - D_2)) \\ & - r_2^2 Y \partial X_1 \end{aligned} \tag{84}$$

Using the definitions of $F(y)$ and $\partial F(y)$, Equations 83 and 84, we use Newton's method to arrive at a value of y that makes $F(y) = 0$.

Appendix E. Dijkstra's Algorithm

Algorithm 8 is the standard version of Dijkstra's algorithm as used in Polar Route.

input : Graph $G = (V, E, w)$ with edge costs $w : E \to \mathbb{R}^+$; initial vertex $v_I \in V$ and destination vertex, $v_G \in V$
output: Dijkstra path, P, which is a list of vertices starting at v_I and ending at v_G, with successive pairs connected by an edge in E

```
Begin
    Q:= empty priority queue of vertices sorted by increasing dist;
    for v ∈ V do
        dist[v] = ∞;
        prev[v] = undefined;
        add v to Q;
    end
    dist[v_I] = 0;
    while Q not empty
        u:= vertex in Q with minimum dist;
        remove u from Q;
        if u == v_G then
            P:= empty path;
            u = v_G;
            prepend u to P;
            while not u == v_I
                u:=prev[u];
                prepend u to P;
            return P;
        end
        for x ∈ V such that (u, x) ∈ E do
            newdist:=dist[u] + w(u, x);
            if newdist < dist[x] then
                dist[x]:=newdist;
                prev[x]:=u;
            end
        end
    return No path exists;
```

Algorithm 8: Dijkstra's Algorithm for finding shortest path between a given start and end location in a graph.

References

Aldea, N., & Kopacz, P. (2020). Time-optimal navigation in arbitrary winds. *Annual Reviews in Control*, *49*, 164–172.

Bast, H., Delling, D., Goldberg, A. V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., & Werneck, R. F. (2016). Route Planning in Transportation Networks. In Kliemann, L., & Sanders, P. (Eds.), *Algorithm Engineering - Selected Results and Surveys*, Vol. 9220 of *Lecture Notes in Computer Science*, pp. 19–80. Springer.

Bijlsma, S. J. (1975). *On Minimal-Time Ship Routing*. Ph.D. thesis, University of Delft.

Bijlsma, S. J. (2001). A Computational Method for the Solution of Optimal Control Problems in Ship Routing. *NAVIGATION*, *48*(3), 144–154.

Bonnard, B., Cots, O., & Wembe, B. (2021). Zermelo Navigation Problems on Surfaces of Revolution and Hamiltonian Dynamics. Tech. rep., hal-03209491v2f.

Crane, K., Livesu, M., Puppo, E., & Qin, Y. (2020). A Survey of Algorithms for Geodesic Paths and Distances. *arXiv e-prints*, *2007.10430*, arXiv:2007.10430.

Daniel, K., Nash, A., Koenig, S., & Felner, A. (2014). Theta*: Any-Angle Path Planning on Grids. *J. Artif. Intell. Res. (JAIR)*, *39*, 533–579.

Dellin, C., & Srinivasa, S. (2016). A Unifying Formalism for Shortest Path Problems with ExpensiveEdge Evaluations via Lazy Best-First Search over Paths with Edge Selectors. In *Proc. of the Internaitonal Conference on Automated Planning and Scheduling*.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, *1*(1), 269–271.

Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, *23*.

Finkel, R. A., & Bentley, J. L. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, *4*(1), 1—9.

Garrido, S., Álvarez, D., & Moreno, L. (2016). Path Planning for Mars Rovers Using the Fast Marching Method. In Reis, L. P., Moreira, A. P., Lima, P. U., Montano, L., & Muñoz-Martinez, V. (Eds.), *Robot 2015: Second Iberian Robotics Conference*, pp. 93–105, Cham. Springer International Publishing.

Garrido, S., Alvarez, D., & Moreno, L. E. (2020). Marine Applications of the Fast Marching Method. *Frontiers in Robotics and AI*, *7*.

Harabor, D., & Grastien, A. (2013). An Optimal Any-Angle Pathfinding Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 308–311.

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Kavraki, L. E., & Latombe, J.-C. (1998). Probabilistic roadmaps for robot path planning. In *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pp. 33–53. John Wiley.

Kavraki, L. E., & LaValle, S. M. (2016). Motion Planning. In *Springer handbook of robotics*, pp. 139–162. Springer.

Koenig, S., & Likhachev, M. (2002). D*Lite. In *Proceedings of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, pp. 476–483. AAAI Press / The MIT Press.

Koenig, S., Likhachev, M., & Furcy, D. (2004). "lifelong planning a^*". *Artificial Intelligence*, *155*(1), 93–146.

Kotovirta, V., Jalonen, R., Axell, L., Riska, K., & Berglund, R. (2009). A System for Route Optimization in Ice-Covered Waters. *Cold Regions Science and Technology*, *55*(1), 52–62.

Lanthier, M. (1999). *Shortest path problems on polyhedral surfaces*. Ph.D. thesis, Carleton University, School of Computer Science.

Lanthier, M., Maheshwari, A., & Sack, J.-R. (1997). Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*.

LaValle, S. M., Branicky, M. S., & Lindemann, S. R. (2004). On the Relationship between Classical Grid Search and Probabilistic Roadmaps. *The International Journal of Robotics Research*, *23*(7-8), 673–692.

Lehtola, V., Montewka, J., Goerlandt, F., Guinness, R., & Lensu, M. (2019). Finding Safe and Efficient Shipping Routes in Ice-Covered Waters: A Framework and a Model. *Cold Regions Science and Technology*, *165*.

Li, Z., Ringsberg, J. W., & Rita, F. (2020). A Voyage Planning Tool for Ships Sailing between Europe and Asia via the Arctic. *Ships and Offshore Structures*, *15*(sup1), S10–S19.

Liu, B., Chen, S., Xin, S.-Q., He, Y., Liu, Z., & Zhao, J. (2017). An optimization-driven approach for computing geodesic paths on triangle meshes. *Computer-Aided Design*, *90*, 105–112.

Mata, C. S., & Mitchell, J. S. B. (1997). A New Algorithm for Computing Shortest Paths in Weighted Planar Subdivisions. In *Proceedings Symposium on Computational Geometry*.

Mishra, P., Alok, S., Rajak, D. R., Beg, J. M., Bahuguna, I. M., & Talati, I. (2021). Investigating Optimum Ship Route in the Antarctic in Presence of sea ice and Wind Resistances – A Case Study between Bharati and Maitri. *Polar Science*, *30*.

Mitchell, J. S. B., Mount, D. M., & Papadimitriou, C. H. (1987). The Discrete Geodesic Problem. *SIAM Journal on Computing*, *16*(4), 647–668.

Mitchell, J. S. B., & Papadimitriou, C. H. (1991). The Weighted Region Problem: Finding Shortest Paths through a Weighted Planar Subdivision. *J. ACM*, *38*(1), 18—73.

Pêtrès, C., Pailhas, Y., Patrón, P., Petillot, Y., Evans, J., & Lane, D. (2007). Path Planning for Autonomous Underwater Vehicles. *IEEE Transactions on Robotics*, *23*(2), 331–341.

Powell, M. D. (1964). An Efficient Method for finding the Minimum of a Function of Several Variables without Calculating Derivatives. *The Computer Journal*, *7*, 155–162.

Rivera, N., Hernández, C., Hormazábal, N., & Baier, Jorge, A. (2020). The 2^k Neighborhoods for Grid Path Planning. *J. Artif. Intell. Res.*, *67*, 81–113.

Rospotniuk, V., & Small, R. (2022). Optimal Any-Angle Pathfinding on a Sphere. *J. Artif. Int. Res.*, *72*, 475—-505.

Sen, D., & Padhy, C. P. (2015). An Approach for Development of a Ship Routing Algorithm for Application in the North Indian Ocean Region. *Applied Ocean Research*, *50*, 173–191.

Sethian, J. (1996). A Fast Marching Level Set Method for Monotonically Advancing Fronts. In *Proc. National Acadacemy of Science*, Vol. 93, pp. 1591–1595.

Sprunk, C. (2008). Planning Motion Trajectories for Mobile Robots using Splines. Tech. rep., Albert-Ludwigs Universität Freiburg.

Topaj, A. G., Tarovik, O. V., Bakharev, A. A., & Kondratenko, A. A. (2019). Optimal Ice Routing of a Ship with Icebreaker Assistance. *Applied Ocean Research*, *86*(November 2018), 177–187.

Veness, C. (2002). Calculate Distance, Bearing and More Between Latitude/Longitude Points. `https://www.movable-type.co.uk/scripts/latlong.html`. Accessed: 2022-01-04.

Vezie, K. (2016). Mercators Projection: A Comparative Analysis of Rhumb Lines and Great Circles. Tech. rep., Whitman College.

Walther, L., Rizvanolli, A., Wendebourg, M., & Jahn, C. (2016). Modeling and Optimization Algorithms in Ship Weather Routing. *International Journal of e-Navigation and Maritime Economy*, *4*, 31–45.

Yatziv, L., Bartesaghi, A., & Sapiro, G. (2006). O(N) implementation of the fast marching algorithm. *Journal of Computational Physics*, *212*(2), 393–399.

Zermelo, E. (1931). Über das Navigations problem bei ruhender oder veränderlicher Windverteilung. *Zeitschrift für Angewandte Mathematik und Mechanik*, *11*(2), 114–124.