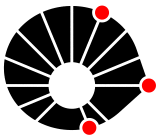
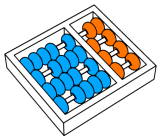




Modeling a fast ship routing optmization algorithm

L. G. Picoli *F. Usberti*

Relatório Técnico - IC-PFG-25-44
Projeto Final de Graduação
2025 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

This page intentionally left blank

Abstract

Efficient weather routing is essential for modern merchant shipping, as route planning must simultaneously account for safety, energy efficiency, and rapidly evolving metocean conditions. Traditional optimization approaches - such as isochrone methods, dynamic programming, and 2D Dijkstra formulations - typically assume fixed ship speed or coarse discretizations, often discarding potentially optimal sub-paths and requiring several minutes to produce a single route. This work presents a new graph-based, three-dimensional weather-routing algorithm designed to compute near-optimal ship trajectories in a matter of seconds. The method reinterprets the 3D weighted-graph formulation proposed by Wang *et al.* (2019) and replaces the conventional time-marching isochrone expansion with a preconstructed N-tree graph generated from spherical isodistance layers centered at the departure point. Each spatial waypoint is expanded into a set of feasible arrival times, forming a full 3D node structure. Edges between stages are built under physical and navigational constraints, and each edge is assigned a cost using a multidimensional speed-loss table provided by Amphitrite, which incorporates waves, wind, currents, heading, and vessel characteristics. Because the graph contains no cycles, the optimal route for any target ETA can be obtained efficiently through a single breadth-first search.

Performance was evaluated using operational metocean data from CMEMS MERCATOR and ECMWF IFS, and bathymetric and coastline masks for land avoidance. A systematic parameter study shows that the algorithm can produce globally optimal or near-optimal routes with execution times as low as a 2 or 3 seconds for long transatlantic voyages. The method also supports variable vessel speeds rather than fixed-power or fixed-speed assumptions, enabling more realistic operational planning. This demonstrates that the proposed 3D N-tree graph combined with BFS search is a fast, flexible, and accurate alternative to traditional weather-routing techniques, and provides a foundation for future improvements such as parallel computing (MPI/OpenMP), multi-route parameter generalization, and integration with power-level decision models.

Keywords- Weather routing; Ship navigation; 3D graph optimization; N-tree graph; Breadth-first search (BFS); Metocean data; NetCDF; Route planning; Speed-loss modeling; Marine operations; Great-circle navigation; Performance optimization; Ocean currents; Wind and wave effects; Computational marine engineering.

Acknowledgments

I would like to express my sincere gratitude to my advisor at Unicamp, Prof. Fábio Usberti, for accepting me as his PGF student and for guiding me throughout this project. I am also thankful to all members and colleagues at Amphitrite, whose collaboration and support were essential to the development of this work. Finally, thanks to my family, friends, and my girlfriend for their constant encouragement and unwavering support.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Navigating cargo ships across the world's oceans is a complex and demanding task that has shaped global trade and human development for centuries. From ancient navigation based on celestial cues to modern systems supported by GPS, satellite observations, and numerical weather prediction models, maritime navigation has undergone a profound technological transformation (1). Yet, despite these advancements, cargo vessels remain highly vulnerable to dynamic environmental conditions such as wind, waves, and ocean currents, which can significantly affect sailing speed, fuel consumption, and operational safety.

In contemporary maritime transportation, ship safety and energy efficiency stand among the most critical factors for achieving competitive and sustainable operations. The industry increasingly relies on sail-planning systems, integrating weather routing and voyage optimization to avoid hazardous conditions, reduce emissions, minimize fuel consumption, and meet operational constraints such as expected time of arrival (ETA). During voyage planning, captains use short-term weather-routing services to circumvent severe conditions and estimate arrival times, while long-term planning relies on computational algorithms that evaluate an enormous number of possible sailing paths across vast oceanic regions (2). Figure 1 illustrates an example of a possible vessel optimization where the vessel travels in favor of currents going in the same direction as its destination.

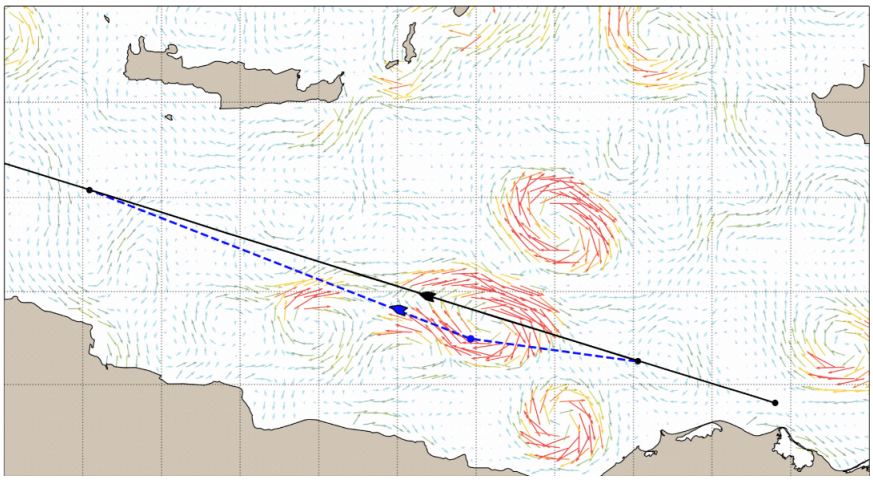


Figure 1 – Ship routing proposal example

Source: (3)

Traditionally, voyage optimization systems discretize the sailing area into a 2D grid of waypoints and search for optimal routes using algorithms such as modified isochrones, dynamic programming, or Dijkstra's algorithm. While effective, these models typically assume fixed-speed navigation and do not fully represent the ship's dynamic operational capabilities such as varying engine loads, speed–power relationships, or weather induced performance variations. This simplification often restricts the search to locally optimal solutions. More advanced 3D approaches incorporate the time dimension, dynamically generating sub-paths as the voyage progresses. However, these methods can eliminate intermediate nodes that may be part of a globally optimal solution, and they often cannot predict ETA until the full optimization completes, limiting their operational practicality.

To address these limitations, the Ocean Engineering study proposed by Wang et al. (2) that inspired this project presented a way of generating a complete 3D weighted graph, where each waypoint is associated with all feasible arrival times based on ship performance and environmental conditions. Dijkstra's algorithm is then applied to this full graph to search for the lowest-cost route that satisfies predefined objectives such as minimizing fuel consumption for a fixed ETA. By separating graph generation from path searching, their method reduces the risk of discarding potentially optimal paths and enables precise ETA targeting.

The present project is a reinterpretation and practical implementation of the method proposed by Wang et al. (2), adapted for high-speed computation and operational deployment. **The goal is to implement a reliable, robust, and extremely fast weather-routing algorithm for cargo ships, capable of generating optimized voyage plans within seconds**, substantially faster than most commercial systems that require several minutes for similar computations. Achieving this performance offers a decisive competitive advantage, enabling near-real-time route updates as environmental conditions evolve.

To achieve this level of performance and precision, three core design decisions were made. First, the C programming language was selected for its low-level control over memory and computation, its predictable execution model, and its suitability for high-performance numerical algorithms. Second, a smart graph construction strategy, inspired by an N-tree exploration structure, enables an efficient breadth-first search (BFS) across a dynamically generated state space of multiple nodes, ensuring that the most relevant routing paths are found. Finally, NetCDF (Network Common Data Form) was adopted as the primary source of environmental information, as it is a standardized, compact, and widely used format in meteorology and oceanography. It allows fast access to gridded datasets such as wind, current, wave, and pressure fields, ensuring that the routing algorithm can interact seamlessly with large, multi-dimensional environmental datasets while maintaining computational efficiency.

1.2 Amphitrite Partnership

This project was developed in close collaboration with Amphitrite, a French maritime-technology company specialized in high-resolution ocean data processing and operational decision-support tools for the shipping industry. Amphitrite played a central role in enabling the practical conception, validation, and benchmarking of the proposed routing algorithm by providing access to advanced datasets, computational models, and an existing routing framework used in real operational environments.

First, Amphitrite supplied the meteorological and oceanographic datasets required to model realistic operational conditions. These included high-quality global fields of ocean currents generated by the company's internal model, as well as wind and wave forecasts obtained from the CMEMS–Mercator Ocean circulation products and the ECMWF–IFS atmospheric prediction system. In addition, Amphitrite provided global bathymetry data, allowing the algorithm to automatically detect land masses, islands, and other navigational constraints during graph construction.

Second, Amphitrite contributed a physically informed cost function used to evaluate the feasibility and energetic impact of candidate route segments. This cost function integrates vessel geometry, environmental conditions, and power or speed settings to estimate both the fuel consumption and the resulting vessel speed between any two nodes in the routing graph. It effectively transforms the environmental inputs into actionable performance metrics, ensuring that the optimization reflects operational reality.

Finally, Amphitrite provided an already implemented operational weather-routing system based on the classical isochrone method. This existing system was used as a baseline for comparison throughout the development of the present work. By contrasting the performance of the proposed N-

tree and BFS-based algorithm with Amphitrite's established isochrone approach, the project benefits from a direct industry-relevant benchmark, enabling a rigorous evaluation of speed, robustness, and routing quality under identical environmental conditions.

Overall, Amphitrite's contributions were essential to grounding this study in real-world data and operational constraints, ensuring that the proposed algorithm is not only theoretically sound but also practical and competitive in a professional maritime context.

Section 2 introduces some important concepts and terminologies to build a common ground of knowledge that is useful for the rest of this report. Section 3 presents the functioning and the idea behind the algorithm used, the reasoning behind it, alongside with the quantitative methods implemented to obtain the results. Section 4 presents the results obtained with some discussion on sensibility tests for optimal parameters. Finally, sections 5 and 6 concludes the research with pertinent discussions for future work.

2 Common ground

2.1 Introduction

Before introducing the methodologies employed to conduct the study, it is worth revisiting and commenting on some basic concepts of navigation, geography, and terminologies that are relevant to the discussions in this report.

2.2 Latitude, longitude and azimuth angle

Latitude, longitude, and azimuth angle are fundamental concepts in cartography and navigation that help us locate points on the Earth's surface and determine directions.

Latitude is the angular measurement of the distance north or south of the Earth's equator. It is measured in degrees and ranges from 0° (at the equator) to 90° (at the North and South poles). Lines of latitude are concentric circles that encircle the Earth from east to west, parallel to the equator. Latitude is used to describe the north-south position of a specific point on the Earth's surface.

Longitude is the angular measurement of the distance east or west from the Prime Meridian, which is defined as 0°. Longitude ranges from 0° to 180° both east and west. From the Prime Meridian, longitudes extend east and west in concentric circles called meridians. Unlike lines of latitude, lines of longitude converge at the North and South poles. Longitude is used to describe the east-west position of a specific point on the Earth's surface.

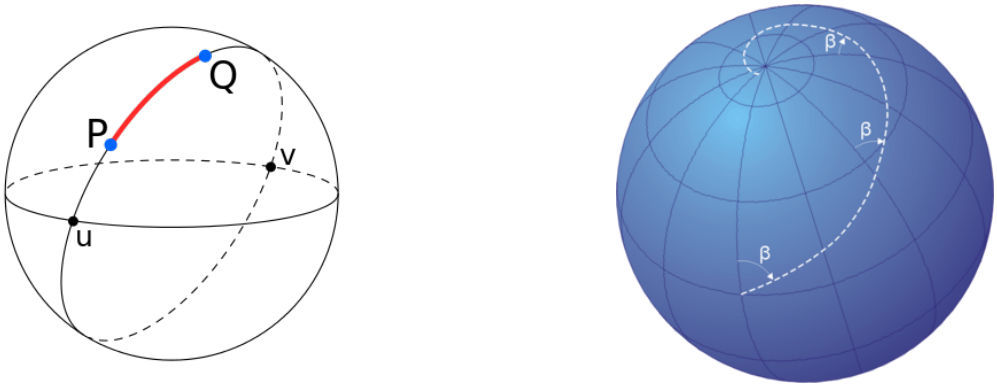
The azimuth angle is a concept used to determine the direction of a point in relation to another reference point. It is generally measured in degrees, taking the North as a reference. The azimuth angle is measured clockwise from North to the direction of the point of interest. For example, if a certain point is located 45° east of North, we say that the azimuth angle is 45°. The azimuth angle is also known as the heading, the direction a vessel is going at a particular moment.

2.3 Loxodromic and Orthodromic Routes

The Orthodromic Route (or The Great Circle) is the concept of a navigation route following a great circle on the spherical surface of the Earth. The great circle is the shortest path between two points on a sphere, such as the Earth. On a globe, a great circle route would appear as a straight line. However, when projected onto a flat map (like a world map), this route often appears curved. The great circle is often used in long-distance navigation, especially in air and sea travel, as it represents the most efficient path in terms of distance.

Loxodromic Route (or Rhumb Line or Constant Course Line): The rhumb line is a navigation route on a map or chart that follows a constant angle with all lines of longitude, forming a spiral around the globe. In other words, on a flat map, the rhumb line would appear as a straight line. This route is easier to follow using a compass and was widely used by sailors before the development of more advanced navigation technologies. However, over long distances, the rhumb line does not represent the shortest path, resulting in a longer route than the great circle.

An example of the orthodromic and loxodromic routes is shown in Figure 2. It's worth noting that, unlike the loxodromic route, in which the azimuth angle is constant along the path, the orthodromic route has a different azimuth angle along the path.



(a) Orthodromic route

Source: (4)

(b) Loxodromic route

Source: (5)

Figure 2 – Loxodromic and Orthodromic routes

2.4 Navigation Concepts

In maritime navigation, several fundamental quantities are used to describe the motion, orientation, and timing of a vessel during a voyage. These concepts are essential for understanding ship routing, performance modeling, and voyage optimization.

Speed Over Ground (SOG). Speed over ground is the vessel's actual speed relative to the Earth's surface. It represents how fast the ship moves from one geographical position to another, independent of the water it is sailing through. SOG is influenced by environmental factors such as ocean currents and is typically measured using GPS.

Speed Through Water (STW). Speed through water is the vessel's speed relative to the surrounding water mass. Unlike SOG, this speed does not account for the effect of currents. For example, a ship with an STW of 12 knots sailing into a 2-knot opposing current will have an SOG of 10 knots. STW is usually measured by Doppler log sensors or electromagnetic logs.

Ship Heading. The ship heading is the direction in which the bow of the vessel is pointed, expressed in degrees relative to true north (azimuth angle). Heading is distinct from the ship's course over ground, which is the direction of actual movement. Wind, waves, and currents can cause a difference between heading and track, requiring continual correction by the vessel's navigation systems.

Estimated Time of Arrival (ETA). The estimated time of arrival is the predicted time at which the vessel will reach its destination or a specific waypoint. ETA is calculated using current

position, expected speed, route geometry, and environmental influences. It is a key operational parameter for scheduling, port coordination, and optimization of fuel consumption.

Estimated Time of Departure (ETD). The estimated time of departure is the planned time at which the vessel is expected to leave a port or waypoint. ETD may be influenced by port operations, cargo handling schedules, weather conditions, or fleet coordination. In route optimization, ETD serves as the starting point for generating the time dimension of the routing problem.

These navigation concepts form the foundation upon which routing algorithms interpret vessel motion, evaluate feasibility, and compute optimal paths under varying environmental conditions.

2.5 Formulation of a voyage

For completeness, this subsection presents a concise restatement a voyage's formulation proposed by Wang et al. (2), which forms the conceptual foundation of the present work. It describes a ship's voyage as a sequence of waypoints $\vec{P}$, each defined by geographical coordinates and an associated passing time. At each waypoint, the ship state is represented as

$$P = [x, y, t]^T, \tag{1}$$

where x and y denote longitude and latitude, and t denotes the time at which the waypoint is reached.

We denote by D_{ij} and θ_{ij} the great-circle distance and the corresponding initial azimuth (heading) required to travel from waypoint P_i to waypoint P_j.

When the vessel sails P_i to P_j its control variables - speed and heading - can be expressed as

$$U(\vec{P}) = U_{ij}(P_i \to P_j) = [v_{ij}, \theta_{ij}]^T, v_{ij} = \frac{D_{ij}}{\Delta t_{ij}} \tag{2}$$

where v is the vessel's speed over ground and θ is the heading angle. These controls determine the ship's progression from one waypoint to the next.

Environmental inputs are expressed as a vector of weather and sea-state parameters,

$$W(\vec{P}) = [H_s,\, T_p,\, C,\, V_{wu},\, V_{wv}, \ldots]^T, \tag{3}$$

including significant wave height H_s, peak period T_p, ocean current components C, and wind velocity components V_{wu} and V_{wv}, all evaluated at the ship state $\vec{P}$.

Ship motion must satisfy a set of feasibility constraints,

$$C(\vec{P},\, U(\vec{P}),\, W(\vec{P})),$$

which encapsulate geometric limits (e.g., land avoidance), machinery limits (e.g., engine power envelope), and environmental safety limits (e.g., prohibitive sea states). The constraint function returns a Boolean value indicating whether the ship can safely and feasibly traverse from one node to another under the given conditions.

The total cost of a voyage segment is obtained by integrating an instantaneous cost function along the path where the feasibility constraints are respected:

$$C = \int_{t_0}^{t_N} f(U(\vec{P}), W(\vec{P}))\, dt. \tag{4}$$

The function $f(U(\vec{P}), W(\vec{P}))$ is the instantaneous cost function that may represent fuel consumption, expected time of arrival, fatigue accumulation, crack propagation risk, or any other performance metric relevant to the voyage optimization problem.

In their work, Wang et al. (2) highlight that conventional two-dimensional routing methods can discard potentially optimal sub-paths. To address this, they propose constructing a complete three-dimensional weighted graph where each spatial waypoint is associated with all feasible passing times. Dijkstra's algorithm is then applied to this fully generated graph to compute an optimal route for a specified objective, such as minimizing fuel consumption for a fixed ETA.

This full-graph approach avoids premature elimination of sub-paths and allows the optimization to target a precise arrival time, overcoming limitations found in dynamic programming or time-marching isochrone methods. The present work adapts, extends, and reinterprets this framework within a more computationally efficient routing architecture.

3 Methodology

3.1 Introduction

The overall objective is to find suitable waypoints $\vec{P}$ and operational control sets $U(\vec{P})$, which will lead to the minimum/maximum value of C by following the optimum route encountering sea conditions $W(\vec{P})$ at those waypoints.

3.2 The algorithm's design

In contrast to the isochrone method, the central idea of this project is to *pre-construct* a structured exploration graph $G = (\vec{P}, \vec{E})$ composed of a large set of fixed candidate waypoints and directed edges between successive stages. Rather than dynamically "creating" new positions at each time step, we first define a spatial skeleton of admissible waypoints $\vec{P}$, then connect them in a disciplined order to form multiple candidate sub-paths (edges) $\vec{E}$. Once the graph is built, the optimization problem becomes the search for a minimum-cost path through this graph. Each edge $\in \vec{E}$ is weighted by evaluating the ship controls and environmental conditions along that edge, and mapping them through the cost model. Subsections 3.2.1 to 3.2.5 present a high-level overview of the algorithmic workflow, outlining the main stages required to construct the routing methodology. These steps are intentionally described in a general manner to provide the reader with an intuitive understanding of the complete process. In the subsequent subsections (3.3 to 3.7), each of these stages is examined in detail, with rigorous explanations, mathematical formulations, and implementation considerations.

3.2.1 Stage generation - isodistance lines

After defining the departure point P_0 and destination point P_N, a great-circle reference route is computed between them. The sailing domain is then discretized into n stages characterized by *isodistance lines*: loci of points that are at the same great-circle distance from the departure. In the reference paper, these stages are obtained by creating lines perpendicular to the great-circle route at regular distance increments. In our implementation, we adopt a geometrically equivalent but spherical-consistent construction: each stage is the intersection of the Earth's surface with a *spherical circle* of radius r_i (in great-circle distance) around the departure point. This yields concentric distance fronts that naturally respect spherical geometry, avoiding distortions that can arise from purely planar perpendicular offsets. Each stage therefore represents a constant-distance shell from P_0, and is indexed by $i = 1, \ldots, n$. Figure 3 shows an example of the construction of the isodistance lines (represented in green) around the great circle (represented as the red line) of a route that begins in the Atlantic and ends in Europe.

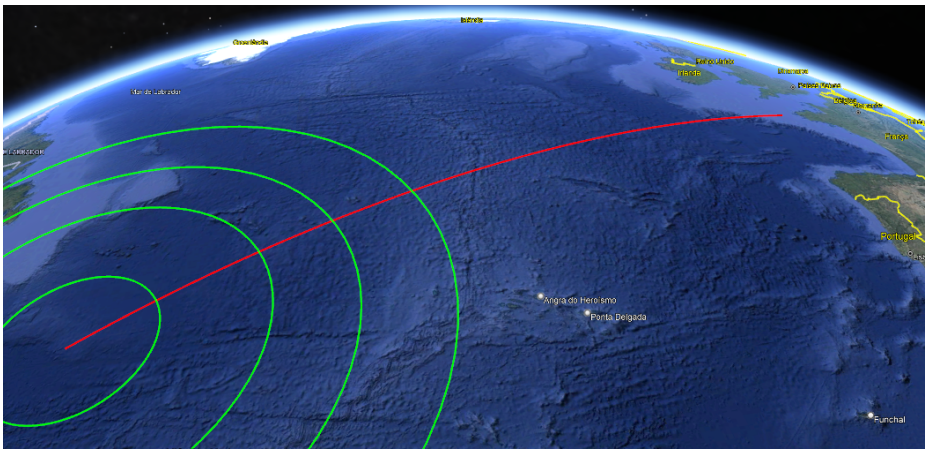


Figure 3 – Representation of 4 isodistance line expansions (green) around the great circle (red line)

3.2.2 Waypoint distribution within stages

For each stage i, we generate a finite set of candidate waypoints

$$P_{i,j} = [x_{i,j},\, y_{i,j},\, t_{i,j}]^T, \qquad j = 1, \ldots, M_i,$$

distributed around the isodistance line, where M_i is equal to the number of points in the i_{th} stage. Intuitively, these waypoints represent alternative spatial deviations from the nominal great-circle track while preserving progression toward the destination. The collection of all $P_{i,j}$ over all stages forms the node set $\vec{P}$ of the graph. Figure 4 shows the waypoints representation for the first stages.

3.2.3 Edge construction and N-tree structure

Edges are defined only between *adjacent stages*. Specifically, each node $P_{i,j}$ in stage i is connected to a subset of nodes $P_{i+1,k}$ in the next stage $i+1$, forming directed edges

$$E_{i,j,k} : P_{i,j} \to P_{i+1,k}.$$

This yields an ordered, acyclic, stage-by-stage graph that resembles an N-tree exploration structure, depicted in Figure 5. Each waypoint branches into multiple feasible successors, but never connects backwards or laterally within the same stage. By construction, any valid route is a sequence

$$P_0 \to P_{1,\cdot} \to P_{2,\cdot} \to \cdots \to P_N,$$

ensuring monotonic progress from departure to destination.

3.2.4 Edge weights from controls and weather

Each edge $E_{i,j,k}$ is assigned a cost by first determining the control required to traverse it. Let

$$U_{i,j,k} = [v_{i,j,k},\, \theta_{i,j,k}]^T$$

be the speed over ground and initial azimuth needed to sail from $P_{i,j}$ to $P_{i+1,k}$ over the chosen time step Δt_i. The associated environmental forcing is taken as the metocean state at the departure waypoint,

$$W(P_{i,j}),$$

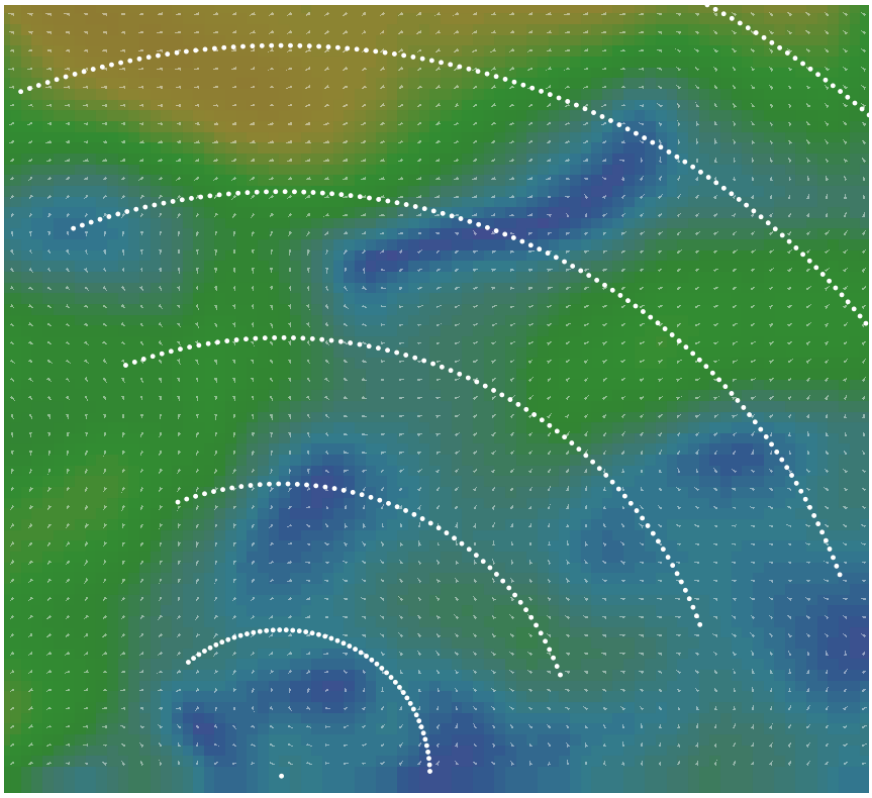


Figure 4 – Waypoints in each stage

including currents, wind, waves, and other fields. The edge weight is then computed through the performance-based instantaneous cost model $f(\cdot)$, leading to

$$C(E_{i,j,k}) \;=\; f\big(U_{i,j,k},\, W(P_{i,j})\big)\,\Delta t_i,$$

where $C(E_{i,j,k})$ may represent fuel consumption, travel time penalty, safety risk, or a multi-objective combination depending on the optimization goal. This transforms the pre-built geometric graph into a weighted operational graph.

3.2.5 Finding best path with BFS

The reference paper proposes applying Dijkstra's algorithm to the resulting 3D weighted graph in order to obtain a global minimum-cost route. However, in our specific construction there is no possibility of closed loops: edges connect strictly from stage i to stage $i+1$, producing a directed acyclic graph (DAG). In such graphs, the search does not require cycle-handling or distance relaxation over arbitrary revisits, and the optimal path can be found efficiently by a breadth-first traversal over stages while maintaining cumulative costs. Therefore, a BFS-style dynamic expansion over this ordered graph is sufficient to recover the minimum-cost route, while reducing bookkeeping overhead compared to a full Dijkstra implementation.

Overall, this graph approach preserves the exhaustive nature of other routing methods exploration, by enumerating many feasible alternatives, but achieves it through a two-step strategy: (i) build a fixed, stage-ordered N-tree graph of candidate waypoints; (ii) evaluate costs on all edges

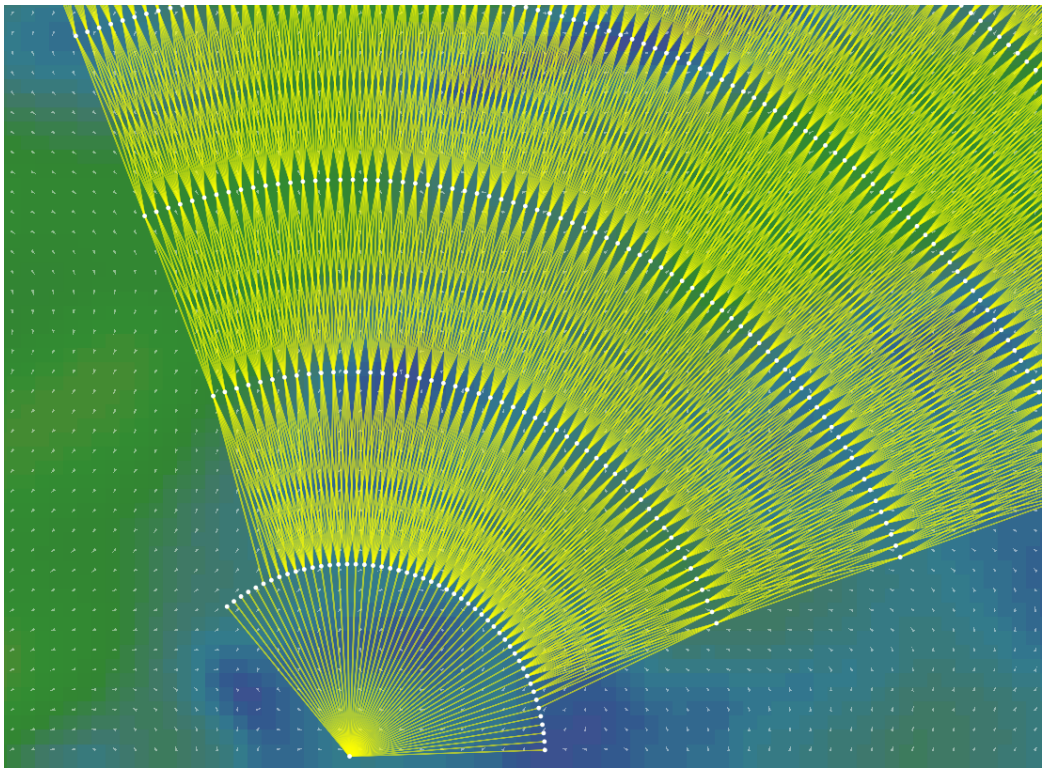


Figure 5 – Edges connection between waypoints forming an N-Tree graph

using $U_{i,j,k}$ and $W(P_{i,j})$, then extract the optimal path via BFS on the acyclic structure. This separation between geometric exploration and cost-driven search is the key architectural difference enabling high computational efficiency in the present work.

3.3 Stage generation in-depth

To discretize the sailing domain into n stages (or isodistance lines), we first compute the great-circle distance between the departure point P_0 and the destination point P_N, denoted D_{total}. This distance is then partitioned using a user-defined spacing D_L (in meters), which represents the radial separation between successive stages. The radius associated with the i-th stage is therefore

$$D_{L_i} = i\, D_L, \qquad i = 0, 1, \ldots, n,$$

with $n = \lfloor D_{\text{total}}/D_L \rfloor$. Each stage L_i corresponds to the locus of points lying at a great-circle distance D_{L_i} from the departure point.

A full spherical circle is not required for each stage. In realistic ocean routing, the optimal path tends to progress monotonically toward the destination, and only deviates laterally when necessary (e.g., to avoid land masses or unfavorable metocean conditions). Hence, exploring regions that move backward relative to the destination is computationally wasteful. For this reason, each stage is constructed as an *arc* of a spherical circle rather than as a complete circle. This arc restricts the exploration to the forward-looking corridor that remains relevant to reach P_N.

To define the exploration arc for a given stage L_i, we proceed as follows. Let Q_i be the point on the great-circle reference route such that the distance from P_0 to Q_i is D_{L_i}. At Q_i, we construct a

perpendicular segment of length D_{P_i}, centered at Q_i. The endpoints of this perpendicular segment, once projected onto the spherical circle of radius D_{L_i}, define the θ_{start} and θ_{end} azimuth angles of the arc. The arc radius is fixed to D_{L_i}, while its angular span is controlled by a user-defined function

$$D_{P_i} = f(D_{L_i}),$$

which expands the lateral exploration progressively up to mid-voyage and then contracts it as the vessel approaches the destination. This design maintains broad exploration where uncertainty is highest (typically in the mid-route oceanic region) while preventing unnecessary branching near the start and end points.

Algorithm 1 gives the pseudocode corresponding to the perpendicular-distance rule implemented in this work. The function enforces a minimum lateral exploration of 10 km near the start, increases proportionally until mid-route, and caps exploration to 100 km to avoid excessive branching.

Input: Stage radius D_{L_i}, total route distance D_{total}
Output: Perpendicular distance D_{P_i}

$$D_{P_i} = \min(d,\, D_{\text{total}} - d)$$
$$D_{P_i} = \text{clamp}\big(D_{P_i},\, 10^5,\, 10^6\big)$$

return D_{P_i};

Algorithm 1: perpendicular distance function for arc generation

Figure 6 illustrates some examples of these perpendicular lines and how their segment size grows with the evolution of the route and 7 illustrates the process of defining the initial and end azimuth (lines in light-green) of the arc of the spherical-circle (in green) using the perpendicular projection of the perpendicular line (in yellow).

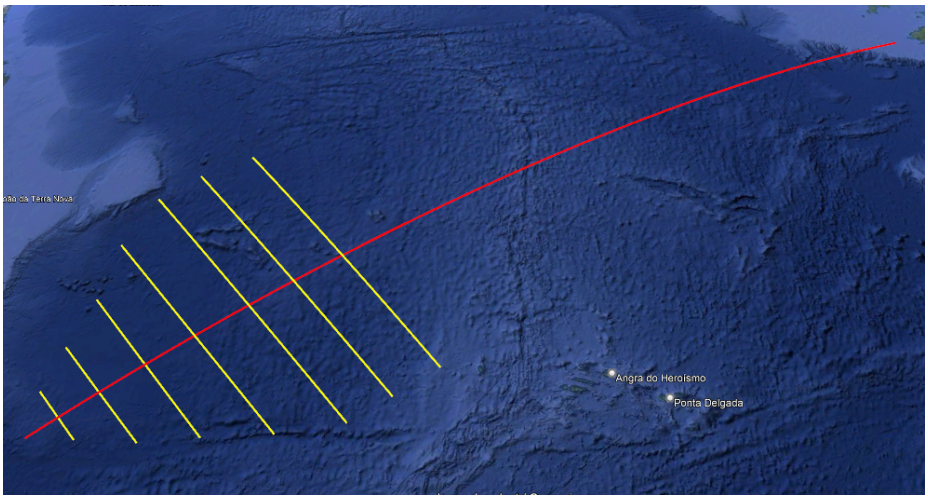


Figure 6 – Example of some perpendicular lines (yellow) varying in size along the route (red)

By doing this process repeatedly we end up building an "exploration area" defined by the set of all isodistance lines merged together along the entirety of the route, as shown in Figures 8 and 9.

Finally each stage L_i can be denoted as

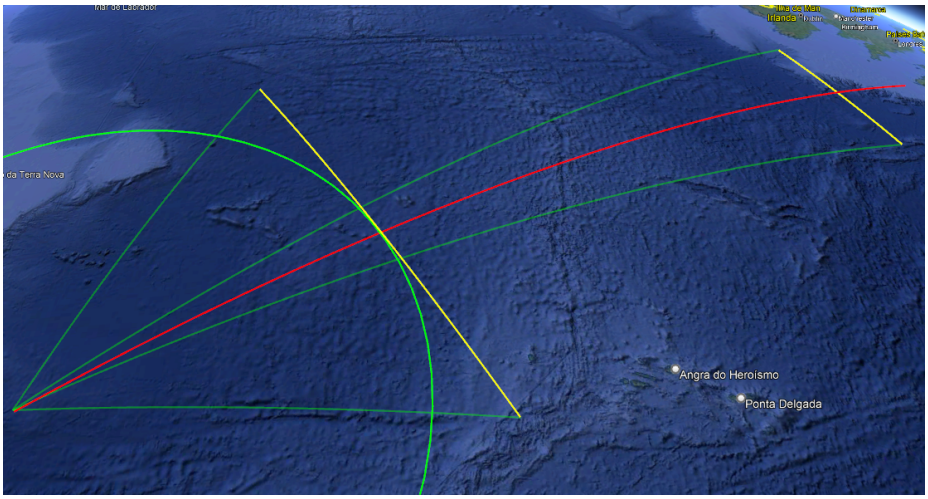


Figure 7 – Representation of the generation process of 2 different isodistance arcs (green) with azimuth angles (defined by light green lines), perpendicular lines (yellow) along the route (red)

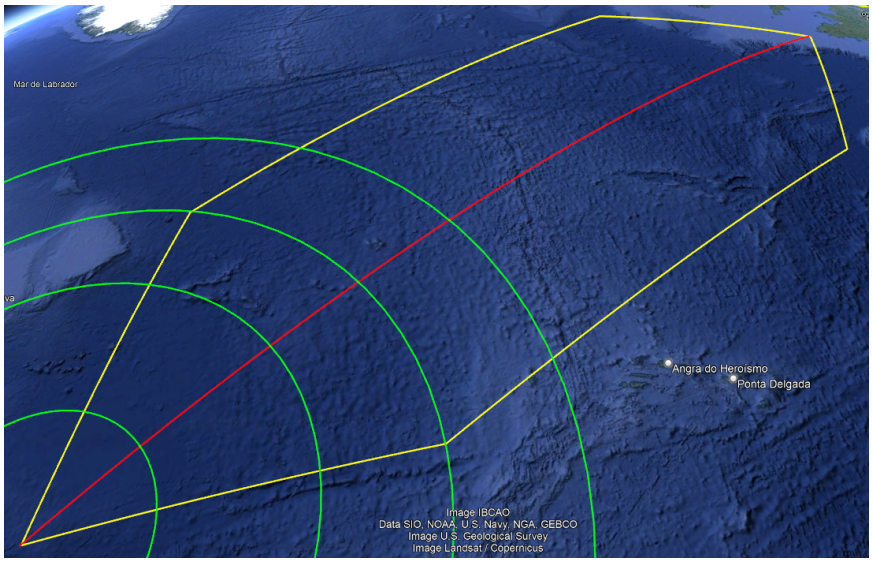


Figure 8 – Exploration area (yellow) defined by the set of all stages (green)

$$L_i = [D_{L_i}, \theta_{start}, \theta_{end}]^T,$$

3.4 Waypoints distribution for each stage in-depth

Once a stage L_i has been defined by the parameters (D_{L_i}, θ_{start}, θ_{end}), the construction of its waypoints is carried out by discretizing the corresponding spherical arc. The idea is to sweep the arc from θ_{start} to θ_{end} using an angular increment θ_{step} such that the spherical distance between two consecutive headings matches a user-defined spacing D_P, representing the desired separation

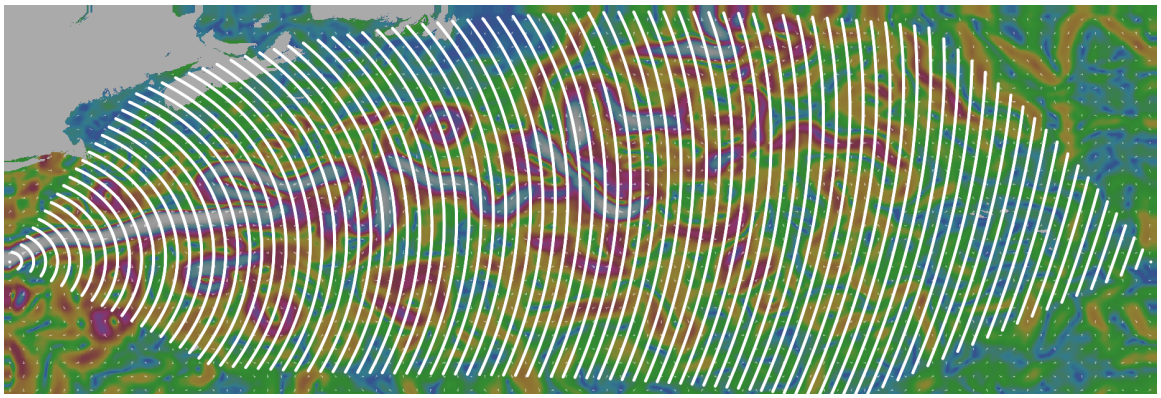


Figure 9 – All stages of a route going from $(36N, 73W)$ to $(36N, 20W)$ in the Atlantic Ocean, using $D_L = 50$ km

between neighboring waypoints on the same stage.

Since the arc length on a sphere is given by $D_{L_i}\,\Delta\theta$, where D_{L_i} is the great-circle radius of the stage, we define

$$\theta_{\text{step}} = \frac{D_P}{D_{L_i}}.$$

The total number of waypoints generated on stage L_i is therefore

$$N_i = \left\lfloor \frac{\theta_{\text{end}} - \theta_{\text{start}}}{\theta_{\text{step}}} \right\rfloor + 1,$$

and each waypoint is constructed by evaluating the geodesic at the heading

$$\theta_{i,j} = \theta_{\text{start}} + j\,\theta_{\text{step}}, \qquad j = 0, 1, \ldots, N_i - 1.$$

The figure 10 illustrates this building process. Waypoints can be seen in yellow and θ_{step} and D_P are indicated.

However, this approach constructs only the two-dimensional aspect of the graph, in which each waypoint is associated solely with a physical location on the Earth defined by its longitude and latitude (x, y). As previously discussed, purely two-dimensional routing formulations may discard potentially optimal sub-paths because they implicitly assume that the vessel sails with a fixed engine power or a constant speed. To overcome this limitation, Wang et al. (2) propose constructing a complete three-dimensional weighted graph in which each spatial waypoint is enriched with all feasible passing times. This three-dimensional formulation preserves temporal variability and prevents premature elimination of candidate paths, allowing the optimization procedure to target a precise arrival time and mitigating constraints that otherwise arise in dynamic programming or time-marching isochrone methods.

The present work adopts and extends this conceptual framework by defining each waypoint as

$$P = [x,\, y,\, t]^T.$$

Thus, for each physical location on Earth, represented by a pair (x, y), we construct multiple graph vertices corresponding to different possible arrival times at that same geographical position. This constitutes the three-dimensional aspect of the exploration graph.

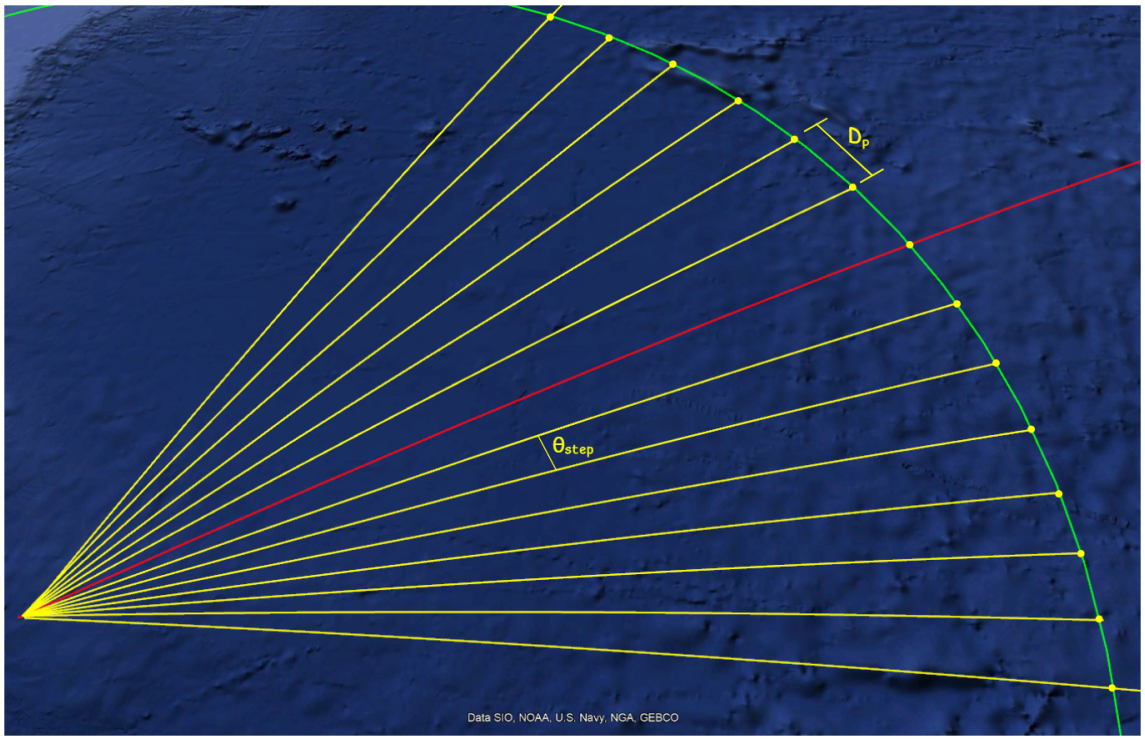


Figure 10 – Waypoints building process with θ_{step} and D_P

To generate the temporal dimension associated with each 2D waypoint, three user-defined parameters are introduced: the minimum vessel speed $v_{\min}$, the maximum vessel speed $v_{\max}$, and the ETA discretization interval ΔETA. Consider a waypoint $P_{i,j}$ located on the i-th stage, which lies at a great-circle distance D_{L_i} from the departure point. If the vessel sails at maximum speed, the earliest possible arrival time at this waypoint is

$$T_{\min} = \frac{D_{L_i}}{v_{\max}},$$

whereas at minimum speed, the latest feasible arrival time is

$$T_{\max} = \frac{D_{L_i}}{v_{\min}}.$$

The set of feasible ETAs associated with the waypoint $P_{i,j}$ is then discretized uniformly between these two limits. The temporal copies of the waypoint are defined by

$$t_{i,j}^{(k)} \;=\; T_{\min} + k\,\Delta\text{ETA}, \qquad k = 0, 1, 2, \ldots, K_{i,j},$$

where

$$K_{i,j} = \left\lfloor \frac{T_{\max} - T_{\min}}{\Delta\text{ETA}} \right\rfloor.$$

Each value $t_{i,j}^{(k)}$ corresponds to a distinct node in the graph, so that a single spatial waypoint generates a vertical "column" of temporally indexed states:

$$P_{i,j}^{(k)} = \begin{bmatrix} x_{i,j} \\ y_{i,j} \\ t_{i,j}^{(k)} \end{bmatrix}, \qquad k = 0, \ldots, K_{i,j}.$$

Figures 11 and 12 help to illustrate this process.

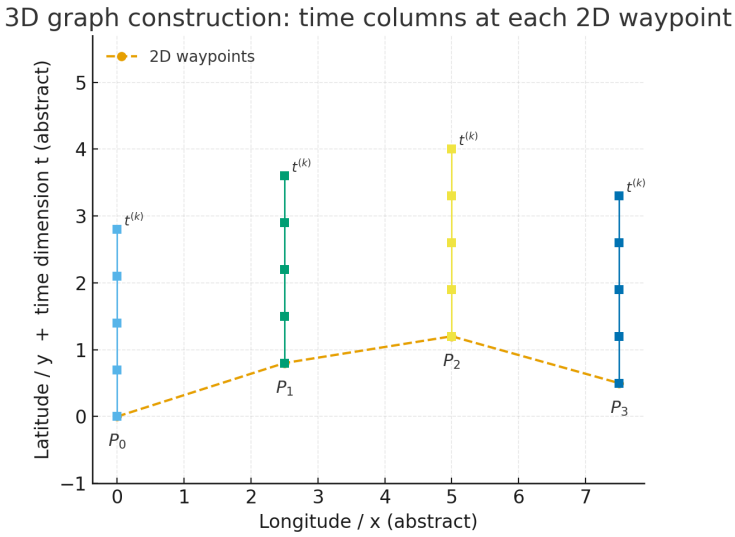


Figure 11 – Illustration of the 3D graph construction: each 2D waypoint (x, y) generates a vertical column of nodes indexed by feasible passing times $t^{(k)}$.

This construction ensures that the graph captures all feasible temporal progressions of the vessel, allowing the routing algorithm to evaluate both slow and fast trajectories, adapt to environmental conditions, and determine the globally optimal path subject to ETA and speed variability.

3.5 Edge construction in-depth

Once all waypoints have been generated, the next step is to construct the edges that connect them, forming the full exploration graph. An edge between two nodes P_1 and P_2 is added only if three feasibility constraints are satisfied:

1. **Limited change in heading.** The heading change between consecutive stages must not exceed a user-defined exploration angle β.
2. **Feasible vessel speed.** The speed required to travel from P_1 to P_2, computed as

$$v_{1\to 2} = \frac{D(P_1, P_2)}{t_2 - t_1},$$

   must lie within the user-defined admissible interval $[v_{\min}, v_{\max}]$.
3. **No land or unsafe-weather intersection.** The sailing segment must not cross land or regions characterized by prohibited metocean conditions.

To enforce the first constraint, heading smoothness between stages is ensured by the exploration angle β. When the geographic waypoints were generated using the geodesic direct problem, an

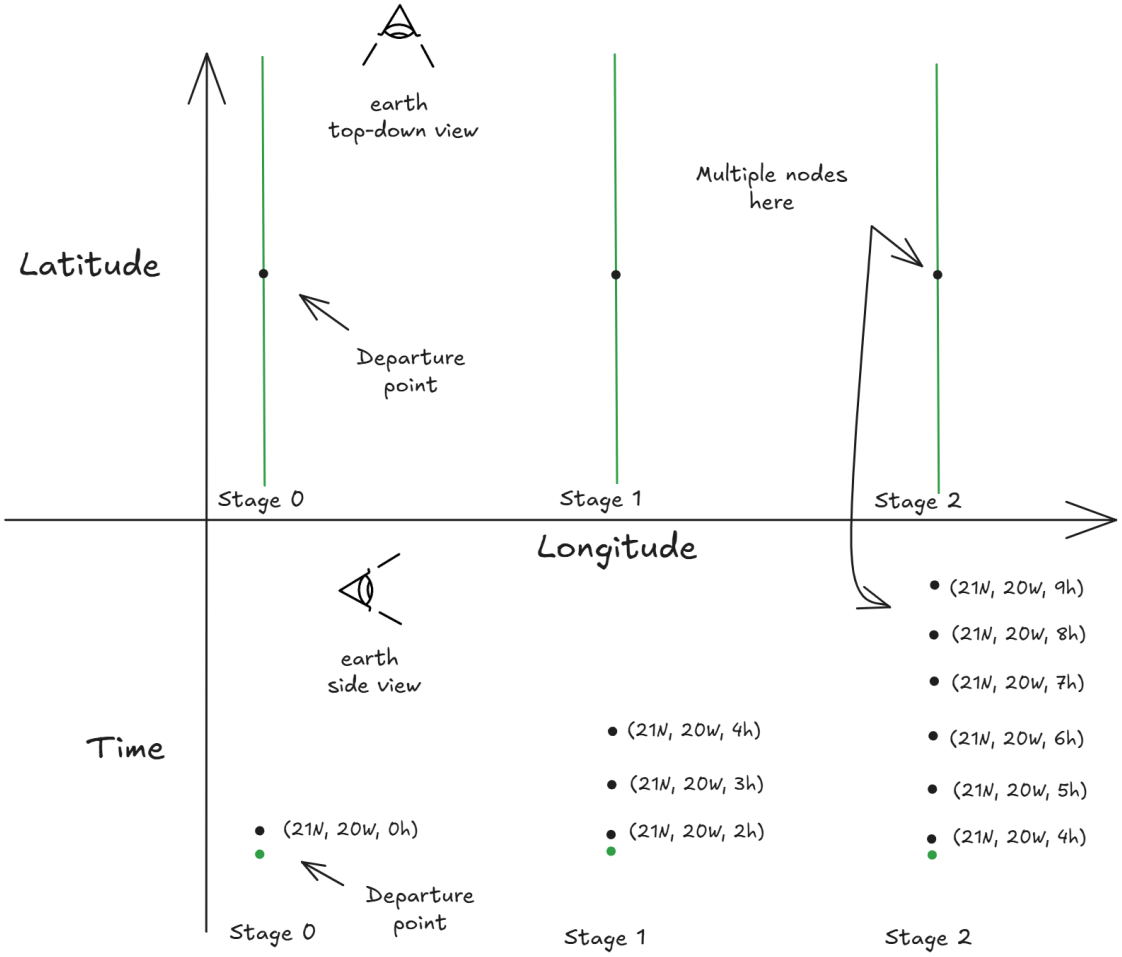


Figure 12 – Waypoints 3d visualization for a $\Delta ETA = 1h$. Green lines are the stages

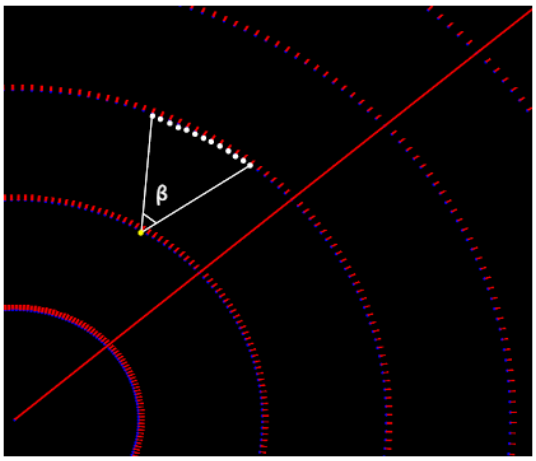
azimuth angle was computed for each point. This azimuth serves as the nominal sailing heading at that waypoint, as illustrated as small red segments in figure 13a.

Edges are therefore considered only when the heading at P_2 lies within an angular window of width β centered at the heading of P_1. Edges that require turning outside of this window are omitted to avoid unrealistic or excessively abrupt course changes (see 13b).

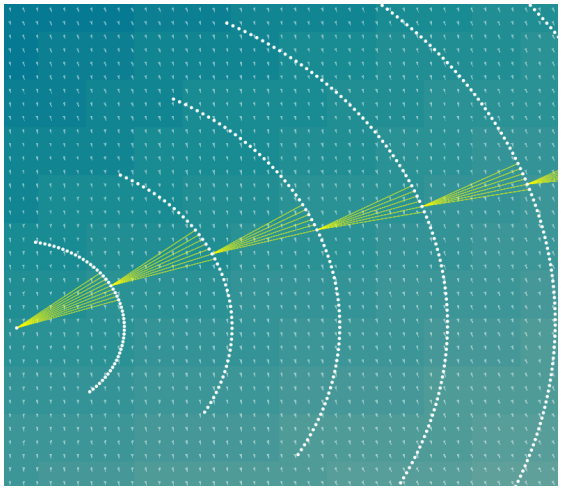
The second constraint is enforced by evaluating the implied vessel speed over the edge. If the time difference $(t_2 - t_1)$ is incompatible with the distance $D(P_1, P_2)$-that is, if the speed falls outside the permissible bounds-the edge is discarded (see figure 14)

Finally, to avoid land masses a land-mask grid provided by Amphitrite is used. Each candidate edge is discretized into a series of small intermediate points separated by a user-defined distance $D_{\text{land-step-check}}$. If any of these points intersect land cells or lie within a region flagged as unsafe according to the metocean limits, the edge is rejected (see Figure 15).

Together, these three constraints ensure that the resulting edges represent dynamically feasible, safe, and smooth sailing segments, yielding a graph that accurately reflects realistic operational



(a) Waypoints headings depicted as small red segments, with β exploration angles. The yellow point represents the current node and white points represent feasible connections.



(b) Example of edge connections for a subset of waypoints.

Figure 13 – Visualization of heading constraints and edge feasibility between waypoints.

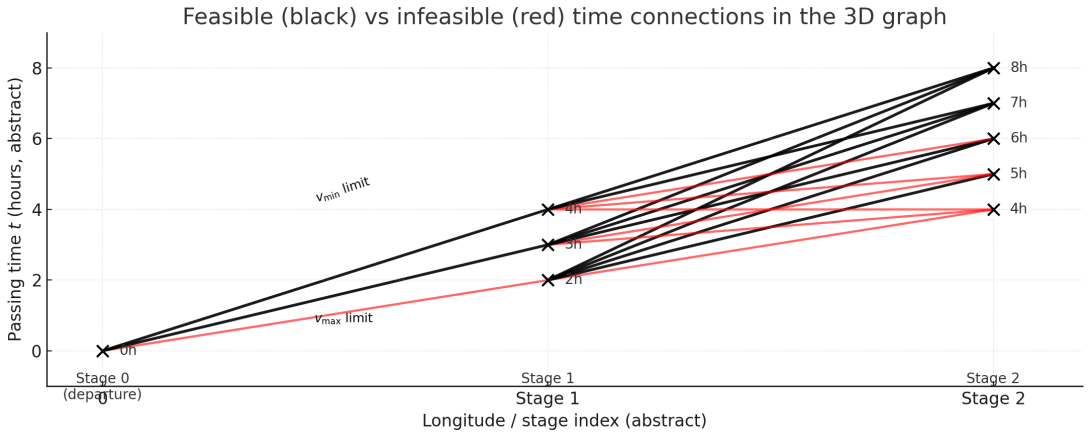


Figure 14 – Feasible (black) and infeasible (red) temporal connections between stages. Each spatial waypoint generates multiple time-indexed nodes; edges are admissible only if the implied speed remains within $[v_{\min}, v_{\max}]$.

limitations.

3.6 Edge weights and cost function in-depth

Once the full 3D exploration graph has been constructed, each directed edge $E_{i,j,k}$ connecting two nodes $P_{i,j}$ and $P_{i+1,k}$ must be assigned a scalar cost (or benefit) that quantifies the operational impact of sailing that segment. Let

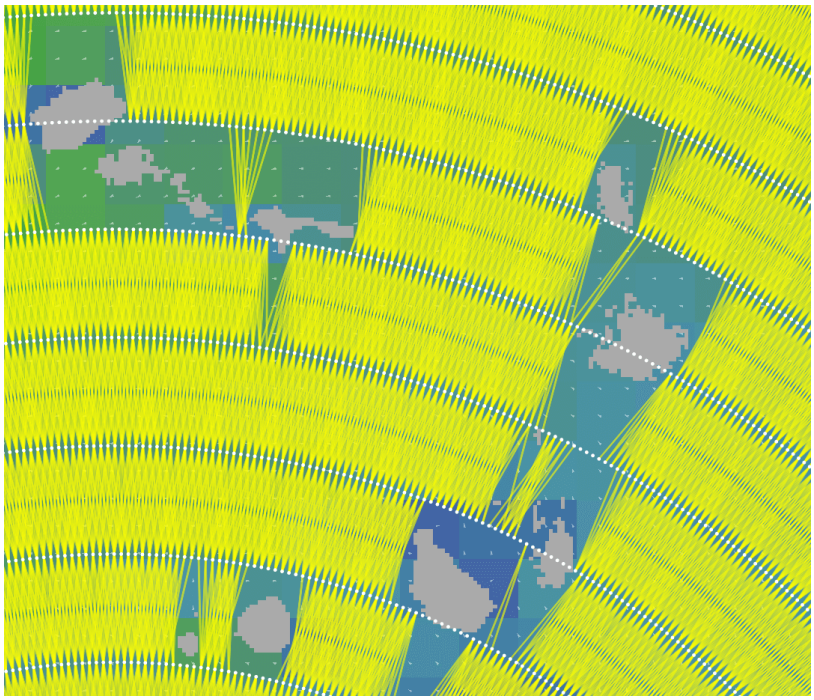


Figure 15 – Example of edge connections for all points while avoiding land masses

$$P_{i,j} = P_1 = [x_1, y_1, t_1]^T, \qquad P_{i+1,k} = P_2 = [x_2, y_2, t_2]^T.$$

The weight assigned to edge $E_{i,j,k}$ depends on:

1. **The required control inputs.** The control vector needed to sail from P_1 to P_2 is

$$U_{1\to2} = \begin{bmatrix} v_{1\to2} \\ \theta_{1\to2} \end{bmatrix},$$

   where $v_{1\to2} = D(P_1, P_2)/(t_2 - t_1)$ is the speed over ground and $\theta_{1\to2}$ is the initial heading.

2. **Environmental forcing along the edge.** The metocean state influencing the vessel is denoted $W(P_1 \to P_2)$. In this work, the relevant environmental variables are:

$$W = [H_s,\ T_p,\ V_{w,x},\ V_{w,y},\ \vec{v}_c]^T,$$

   where H_s is the significant wave height, T_p the wave period, $(V_{w,x}, V_{w,y})$ the wind velocity components, and $\vec{v}_c$ the ocean current vector.

3. **Performance-based cost model.** A performance function $f(\cdot)$ maps (U, W) into a physical or operational cost.

The continuous cost of an edge is:

$$C(E_{i,j,k}) = \int_{t_1}^{t_2} f\big(U_{1\to2}(t),\ W(P(t))\big)\, dt.$$

Discrete approximation. For small D_L, metocean fields vary weakly over the edge. Thus:

$$C(E_{i,j,k}) \approx f\big(U_{1\to2},\, W(P_1)\big)\,(t_2 - t_1).$$

If D_L is large, the edge is discretized into points $\{P^{(q)}\}$, giving:

$$C(E_{i,j,k}) \approx \sum_{q=0}^{Q-1} f\big(U^{(q)},\, W(P^{(q)})\big)\,\Delta t^{(q)}.$$

On the difficulty of defining physics-based cost functions. In classical naval architecture, predicting ship performance at sea requires evaluating several semi-empirical components:

— calm-water resistance (ITTC formulations),
— added wave resistance (STAWAVE, Holtrop–Mennen, or strip theory),
— wind drag models,
— shallow-water corrections,
— trim and sinkage effects,
— temperature and density corrections,
— nonlinear wave–hull interaction models.

A fully analytical model combining all these effects is extremely complex, often computationally expensive, and depends on numerous vessel parameters that must be tuned using model tests or sea trial data. Consequently, using such physics-based formulations for this university project is generally impractical.

Amphitrite's multidimensional speed-loss table. To avoid these challenges, Amphitrite provides a precomputed, high-dimensional lookup table (built with machine learning and high-quality data of past voyages for multiple vessels) that directly returns the *speed loss* experienced by the vessel for any combination of:

$$(v_s,\ \theta,\ H_s,\ T_p,\ \vec{v}_c,\ V_{w,x},\ V_{w,y},\ \text{draft}, \ldots).$$

Here, the vessel input speed v_s already incorporates both magnitude and heading direction, i.e.,

$$v_s = v_{1\to2}, \qquad \theta = \theta_{1\to2}.$$

For each waypoint, the local metocean conditions are extracted via index-based search in the NetCDF grids. Once all inputs are gathered, the multidimensional speed-loss table is queried using a nearest-neighbour search over all dimensions. This approach yields the speed through water and resulting total speed loss immediately, without requiring any expensive resistance or seakeeping computations.

Cost function used in this work: accumulated speed gain. A vessel's speed over ground satisfies:

$$\vec{v}_{\text{sog}} = \vec{v}_{\text{stw}} + \vec{v}_c.$$

The speed through water is given by the lookup table:

$$\vec{v}_{\text{stw}} = g(v_s,\, \theta,\, H_s,\, T_p,\, V_{w,x},\, V_{w,y})\,.$$

Thus,

$$\vec{v}_{\text{sog}} = g(v_s, \theta, H_s, T_p, V_{w,x}, V_{w,y}) + \vec{v}_c.$$

The speed gain relative to calm-water service speed is defined as:

$$\Delta v = v_{\text{sog}} - v_s.$$

The cost assigned to edge $E_{i,j,k}$ is therefore:

$$C(E_{i,j,k}) = \Delta v_{i,j,k},$$

and the routing problem becomes one of finding the path that *maximizes* accumulated speed gain across all stages. As a result, each terminal node at the final stage contains multiple ETAs, each associated with the maximum achievable cumulative speed gain under the vessel's operational constraints and encountered metocean conditions. A cost function to minimize fuel could be used as well but the Amphitrite's lookup table doesn't have yet implemented a fuel consumption dimension to be used as output.

The NetCDF data The metocean information required for evaluating each graph edge is stored in a collection of NetCDF files. These files are loaded into memory using the official NetCDF C API. To minimize memory usage and improve I/O performance, the system does not load the entire global dataset; instead, only the subset of variables that fall inside the bounding box of the exploration region (as defined in Subsection 3.3) are pre-loaded into RAM. This ensures that the graph generation and cost-evaluation steps access only the relevant spatial domain.

A practical complication arises from the fact that different operational datasets adopt different longitude conventions: some span $[-180°, 180°]$ while others use $[0°, 360°)$. When the exploration bounding box crosses these coordinate boundaries, wrapping must be handled explicitly. A dedicated longitude-normalization routine was implemented to ensure that: (i) requests outside the native NetCDF longitude domain are correctly mapped via modular arithmetic, and (ii) the extracted data form a continuous spatial field consistent with the route-search domain. This careful preprocessing step guarantees that the metocean variables remain coherent with the geometry of the generated graph.

Although alternative strategies were tested, such as lazy-loading and on-demand disk access through the NetCDF API, the performance degraded significantly, primarily due to high random-access latency and the large number of data lookups required during cost evaluation. Empirically, pre-loading the relevant spatial subset into RAM proved to be the most efficient and stable solution, drastically reducing I/O overhead and enabling real-time or near real-time edge-cost computation.

3.7 Applying BFS in-depth

After the full 3D exploration graph is constructed-with spatial layers indexed by stage i, weights assined, and temporal copies of each waypoint $P_{i,j}^{(k)}$-the next step is to determine the optimal path(s) that maximize the accumulated speed gain until reaching the destination layer. Unlike classical routing systems that rely on Dijkstra's algorithm or dynamic programming approaches, the graph produced in this work has a special structure: it is a *directed acyclic layered graph* (DAG) in which edges only connect nodes in stage i to nodes in stage $i + 1$. Because no backward edges or cycles exist, the shortest/longest path problem can be solved efficiently using a simple Breadth–First Search (BFS) traversal across layers.

Figure 16 illustrates this process. Each node in stage $i + 1$ receives edge contributions from all feasible nodes in stage i. For each incoming edge, its cost contribution $C_{p,q}$-representing the accumulated speed gain-is added to the cumulative cost of the predecessor node. Thus, the cost

of reaching a node is computed recursively. For example, in the left part of the diagram, the cost of reaching node P_6 is the minimum (or maximum, depending on the objective) among all possible incoming cumulative costs:

$$CP_6 = min(C_{1,6},\ C_{2,6},\ C_{3,6},\ C_{4,6}).$$

Similarly, when expanding the next stage, all predecessor cumulative costs are propagated forward. In the middle diagram, the cost of reaching node P_5 is

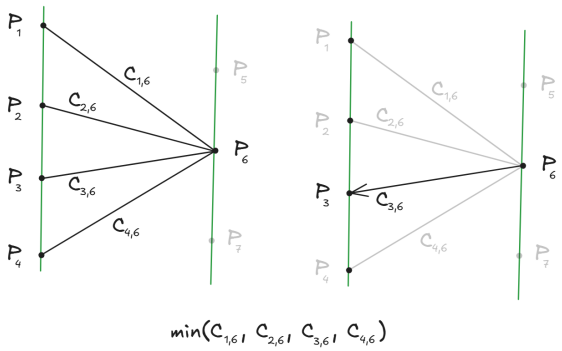
$$CP_5 = C_{1,5} + CP_1,$$

while the cost of reaching P_7 or P_6 follows analogously.

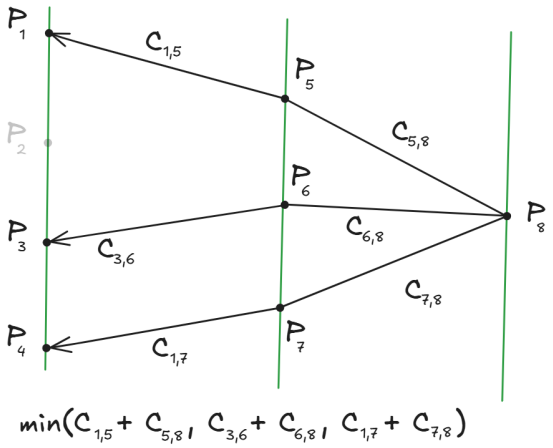
Finally, at the destination layer (rightmost diagram), the cumulative costs of all candidate paths reaching the terminal waypoint P_8 are compared:

$$CP_8 = min(C_{1,5} + C_{5,8},\ C_{3,6} + C_{6,8},\ C_{1,7} + C_{7,8}).$$

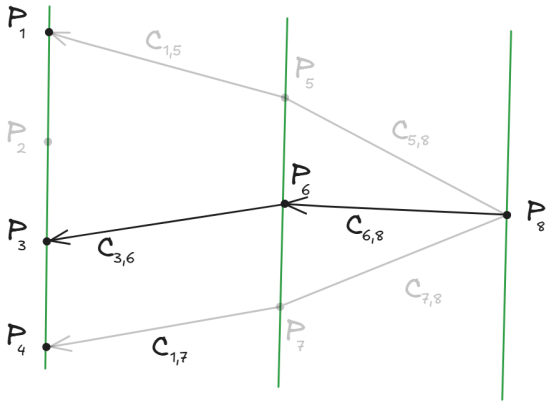
This layered propagation mechanism is exactly the same recursive structure described in the original paper by Wang et al. (2) (Eq. (9)–(11)), but implemented through BFS instead of Dijkstra's algorithm. Because every node in stage i connects exclusively to stage $i + 1$, all outgoing edges from stage i are visited before moving to stage $i + 2$. This ensures that BFS naturally follows the temporal-spatial structure of the graph.



(a) Step 1



(b) Step 2



(c) Step 3

Figure 16 – Illustration of the cumulative-cost propagation across layers of the 3D exploration graph. In each stage, all incoming feasible edges contribute potential cumulative costs, and the optimal value is selected. This procedure is equivalent to dynamic programming but is executed via BFS because of the acyclic layered graph structure.

Why BFS is Sufficient for This Graph Structure

Several important properties justify the use of BFS:

— **Acyclicity.** Edges always go from stage i to stage $i+1$. There are no backward transitions or cycles, making the graph a DAG.

— **Uniform layer expansion.** All nodes in stage i are processed together, exactly matching the natural structure of BFS.

— **Fixed horizon.** The number of stages is predetermined by the discretization of the voyage, so BFS completes in exactly n iterations.

— **Equivalent to dynamic programming.** BFS combined with cumulative cost propagation produces the exact recurrence described in Eq. (10) of Wang et al. (2):

$$CP_{i,j} = \min_k \left(CP_{i-1,k} + \Delta c_k\right),$$

but without requiring priority queues or relaxation steps.

Because BFS processes the graph level by level, it is computationally lighter than Dijkstra's algorithm: no heap operations, no key decreases, and no search across intra-layer edges are required. This is especially important for large maritime grids containing tens of thousands of nodes.

Extracting the Optimal Path

During the BFS traversal, each node stores the index of the predecessor that yielded the optimal cumulative cost:

$$\text{parent}(P_{i,j}) = P_{i-1,k}.$$

Once the destination layer is reached, the optimal route is recovered by backtracking from the best terminal node:

$$P_N^{(k)} \to P_{N-1}^{(k)} \to \cdots \to P_0.$$

This yields the full spatio-temporal optimal ship trajectory-both the waypoints and their associated ETAs.

Overall, BFS is not only simpler but also more efficient than methods typically used for continuous-time shortest-path problems. Its layer-by-layer expansion perfectly matches the discretized structure of the routing graph and allows optimal paths to be found in milliseconds.

3.8 Extracting waypoints

Once all optimal waypoints have been extracted by backtracking the BFS parent links,

$$P_N^{(k)} \to P_{N-1}^{(k)} \to \cdots \to P_0,$$

the resulting trajectory consists of one waypoint per stage. Depending on the discretization resolution-especially the choice of stage spacing D_L and the total voyage distance-the complete optimal route may contain more than one hundred waypoints. Although this dense waypoint representation is useful for the internal optimization procedures, it is not desirable for operational use. In practice, ship captains typically work with a much smaller set of navigational waypoints: approximately 15–20 waypoints for transoceanic voyages and 5–10 for shorter coastal routes.

To reduce the number of waypoints while preserving the overall geometry of the optimal path, the Ramer–Douglas–Peucker (RDP) algorithm is applied. RDP is a classical polyline simplification technique that iteratively removes points that contribute minimally to the shape of the path, while retaining those that define significant deviations or curvature. The algorithm is controlled by a user-defined parameter $\epsilon_{\max}$, which specifies the maximum allowable perpendicular distance between the

simplified path and the original polyline. This threshold determines the fidelity of the final route representation.

How the RDP Algorithm Works. Given an ordered set of waypoints $\{P_0, P_1, \ldots, P_N\}$, the RDP algorithm proceeds as follows:

1. Connect the first and last points P_0 and P_N with a straight line.
2. Compute, for every intermediate point P_i, the perpendicular distance d_i to this line.
3. Identify the point P_{i^*} with the maximum deviation:

$$i^* = \arg\max_i d_i.$$

4. If $d_{i^*} > \epsilon_{\max}$, then P_{i^*} is an essential point and must be retained. The polyline is recursively subdivided into two segments:

$$\{P_0, \ldots, P_{i^*}\}, \qquad \{P_{i^*}, \ldots, P_N\},$$

   and the algorithm is applied to each segment.
5. If $d_{i^*} \leq \epsilon_{\max}$, all intermediate points between P_0 and P_N are discarded; the straight line segment is a sufficiently accurate approximation.

The recursion terminates when all segments satisfy the $\epsilon_{\max}$ tolerance criterion. The result is a significantly reduced set of waypoints that preserves the essential geometric features of the optimal route (see images 17 and 18).

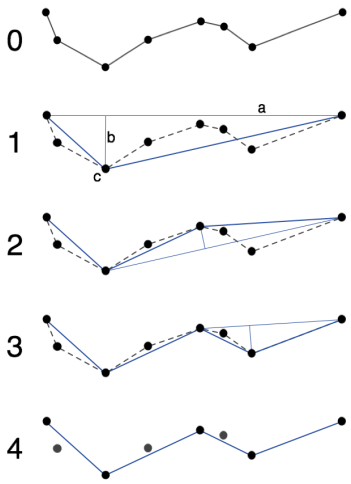


Figure 17 – Example of Ramer-Douglas-Peucker Algorithm

Practical Impact. By adjusting $\epsilon_{\max}$, the user can balance route smoothness and navigational precision: larger values produce fewer waypoints and a more streamlined route, while smaller values preserve detailed turns and avoidances. This allows the method to produce operationally realistic voyage plans from a mathematically optimal but overly dense waypoint sequence.

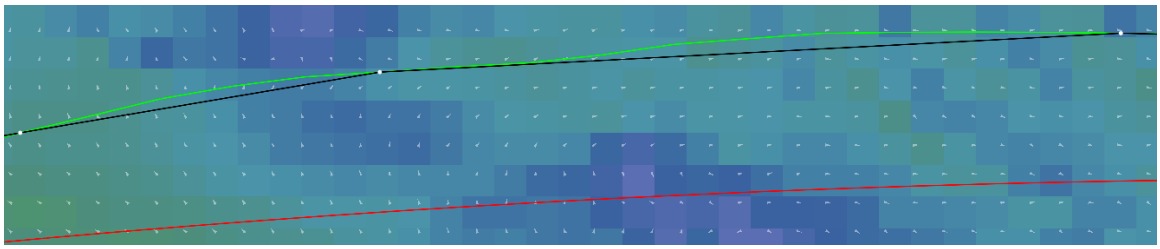


Figure 18 – Example of a particular route after application of RDP algorithm. The red line represents the great-circle route, the green line represents the optimal route by connecting all nodes, and the black line represents the route after application of RDP algorithm by unifying the waypoints highlighted in white. The chosen $\epsilon_{max} = 10$km for this example.

3.9 Construction of the Display

To support the analysis, validation, and debugging of the routing algorithm, a dedicated visualization tool was developed using OpenGL. This display window enables real-time inspection of all relevant NetCDF fields, including ocean currents, wind vectors, and wave parameters, across the entire exploration domain. A customizable heatmap renders scalar variables (e.g., significant wave height, wind speed, current magnitude) using a perceptually uniform color scale, allowing rapid identification of spatial gradients and meteorological structures.

In addition to scalar fields, the system overlays directional information. Small triangles are rendered at each grid point to indicate the orientation of currents, wind, or wave propagation, providing an intuitive vector-field representation superimposed on the underlying heatmap.

The visualization environment includes interactive controls: mouse-driven panning to navigate across the domain, smooth zooming for inspecting local details, and keyboard shortcuts for switching between datasets (currents, wind, waves, bathymetry, or land masks). A NetCDF-based landmask is also loaded and displayed, allowing islands and coastlines to be visually distinguished. This is particularly useful when diagnosing incorrect graph edges (e.g., edges crossing land or shallow-water regions).

Overall, the OpenGL display acts as an essential debugging interface. When the graph generation, edge construction, or cost-evaluation components produce unexpected behavior, the visual tool allows rapid identification of inconsistencies between the algorithm's output and the underlying meteorological or bathymetric data.

Figures 19 and 20 illustrate examples of the visualization tool in use.

3.10 Parameters definition

3.10.1 The problem

Because the proposed methodology relies on an explicitly constructed graph, the quality of the final solution depends directly on the density and structure of that graph. In principle, the denser the graph and the greater the number of feasible edges, the higher the likelihood of discovering a truly global optimal solution rather than a locally optimal route constrained by sparse connectivity. However, this comes at a cost: increasing the number of nodes and edges also increases the computational load. Thus, an intrinsic trade-off exists between *precision* and *execution time*, and it is reasonable to assume an inverse correlation between these two quantities.

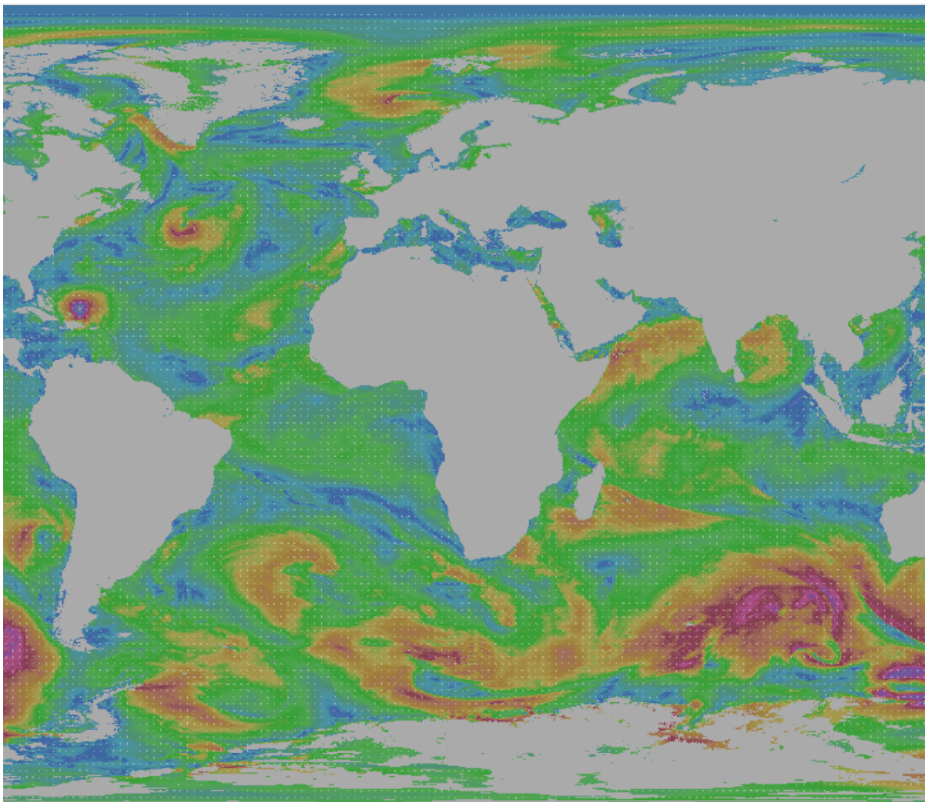


Figure 19 – Display window showing wind + land data

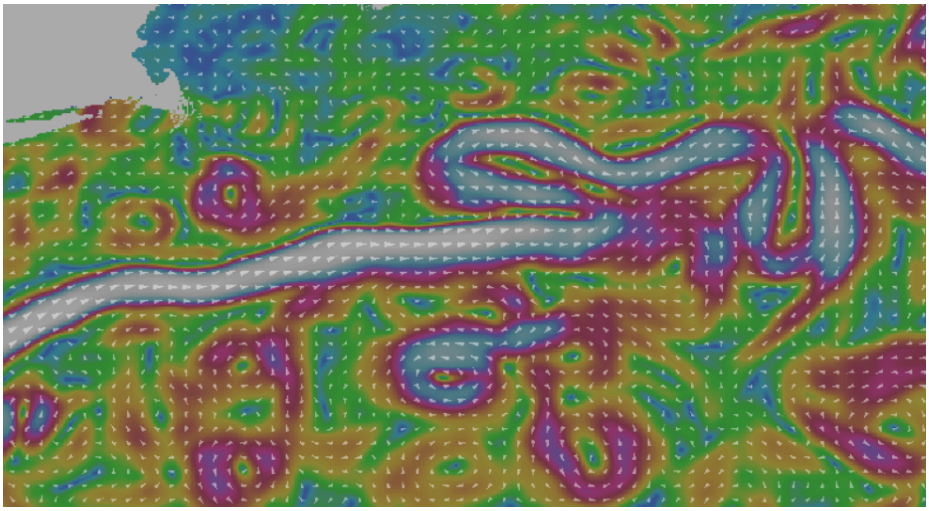


Figure 20 – Display window showing currents data near the gulf stream

The parameters that control the density of the spatial and temporal graph are:

$$D_L, \qquad D_P, \qquad \beta, \qquad \Delta ETA$$

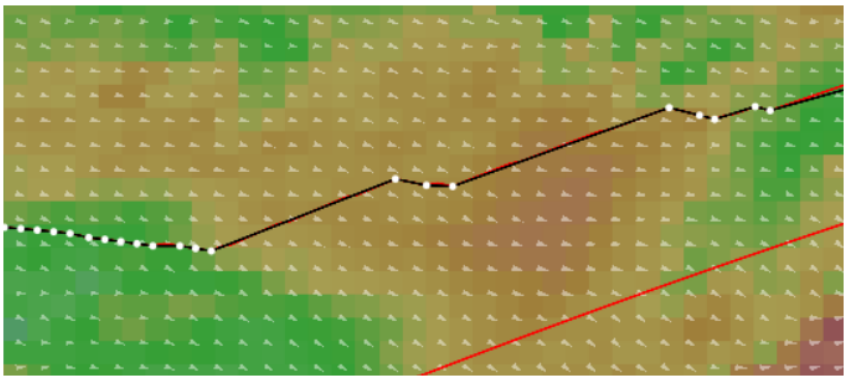


Figure 21 – Example of a "staircase effect" in a route where $D_P \approx D_L$

representing respectively the stage spacing, lateral waypoint spacing, heading exploration angle, and the time difference between all possible ETA's. Smaller values of D_L and D_P produce more stages and more waypoints per stage, while a larger value of β increases the angular spread of connections between successive stages. A larger speed range and a lower ETA difference will impact the amount of temporal connections that can be made between one stage and the other. Together, these parameters determine the resolution of the search space.

The question that arises is: **what are the best values of those parameters that will ensure a fast execution (few seconds) and an optimal solution equal to (or at least the closest possible to) the global optimal?**

3.10.2 Premise nº 1

Before answering this question it's worth exploring a little bit how the data is stored. If we take a closer look in our NetCDF files, we'll see the following resolutions:

- **Currents resolution**: $\frac{1}{30}$ deg $= 2'$, meaning that from one grid cell to the other we have roughly 2 nautical miles ($R_c = 2nm = 3704m$) spacing, and 1 day of temporal spacing (currents only change significantly from one day to another).
- **Wind and Waves resolution**: $\frac{1}{4}$ deg $= 15'$, meaning that from one grid cell to the other we have roughly 15 nautical miles ($R_w = 15nm = 27780m$) spacing, and 3 hour of temporal spacing (winds and wave change more often).

This means that graphs generated from distance values D_L and D_P that are lower than $min(R_c, R_w) = R_c = 3704m$ will eventually generate two points that lie in the same cell leading to redundancy. To avoid redundancy (2 points in the same cell), we need to set

$$D_L >= D_P >= R_c$$

and it would be reasonable to assume that those conditions are sufficient, given the spatial resolution of the data, to find a global optimal path. The problem by setting $D_L = D_P$ or even $D_L \approx D_P$ in the above equation is that the spacing between a stage and the lateral waypoints from the next stage is equal, meaning that the heading change of the vessel from one stage to another usually is quite big in some cases (approx 45 deg), and that often results in unrealistic, "staircase effect" routes that doesn't represent the real motion and heading change capacity of a vessel (example showed in figure 21).

Therefore, the premise number 1 considered to find the global optimal path is that

$$D_L >> D_P >= R_c$$

3.10.3 Experiments

To understand the trade-offs between accuracy and speed, a systematic parameter exploration was performed. First the parameters for global optimal path $\vec{P_G}$ were defined, then several combinations of (D_L, D_P, β) were tested to observe how graph density affects: (i) the smoothness and optimality of the resulting route, (ii) how much it deviates from the defined $\vec{P_G}$, and (iii) the algorithm's execution time.

For simplicity, and to avoid an overly long and repetitive results section, the analysis presented here focuses on the estimation of the ETA on a single representative transoceanic corridor in the North Atlantic with a fixed speed vessel ($v_{min} = v_{max}, \quad \Delta ETA = 0$), to simulate fixed engined power. The evaluated route spans from coordinates (30.0°N, 77.0°W) to (43.0°N, 30.0°W). This choice provides a reasonable testbed: long enough to expose sensitivity to graph density, yet well contained within regions with strong currents and variable weather patterns.

For all experiments, the land-intersection discretization parameter was fixed at

$$D_{\text{step-check-land}} = 1 \text{ nmi},$$

and the waypoint simplification tolerance was set to

$$\epsilon_{\max} = 3 \text{ nmi}.$$

These values were selected to guarantee consistent obstacle detection and stable path simplification for all tested configurations, regardless of graph density.

Empirical tests indicated that the parameter combination

$$D_L = 20 \text{ nmi}, \qquad D_P = 2 \text{ nmi}, \qquad \beta = 120°$$

produces highly reliable solutions. This configuration was used to generate the reference route $\vec{P_G}$, which is assumed to be the closest approximation to the global optimum. All other experiments are evaluated with respect to this baseline.

To investigate the sensitivity of the algorithm to graph density, the following variations were tested:

$$\begin{aligned} D_L &= (20, 25, 30, 35, 40, 45, 50), \\ D_P &= (2, 2.5, 3.0, 3.5, 4, 4.5, 5.0), \\ \beta &= (120, 110, 100, 90, 80, 70, 60), \end{aligned}$$

yielding a full factorial set of $7^3 = 343$ parameter combinations. Each experiment was labeled from $L_0P_0B_0$ to $L_6P_6B_6$, and the results were stored in individual JSON files containing the computed ETA, total sailed distance, execution time, number of generated waypoints, and performance metrics used by the routing engine. Since $\vec{P_G}$ is expected to be the most optimal route, its ETA should be minimal; any increase in ETA among the other test cases indicates either loss of precision or insufficient exploration caused by coarser parameter choices.

4 Results

The first set of comparisons analyzes the impact of reducing the angular exploration window β and increasing waypoint spacing D_P, while keeping the stage spacing fixed at $D_L = 20$ nmi.

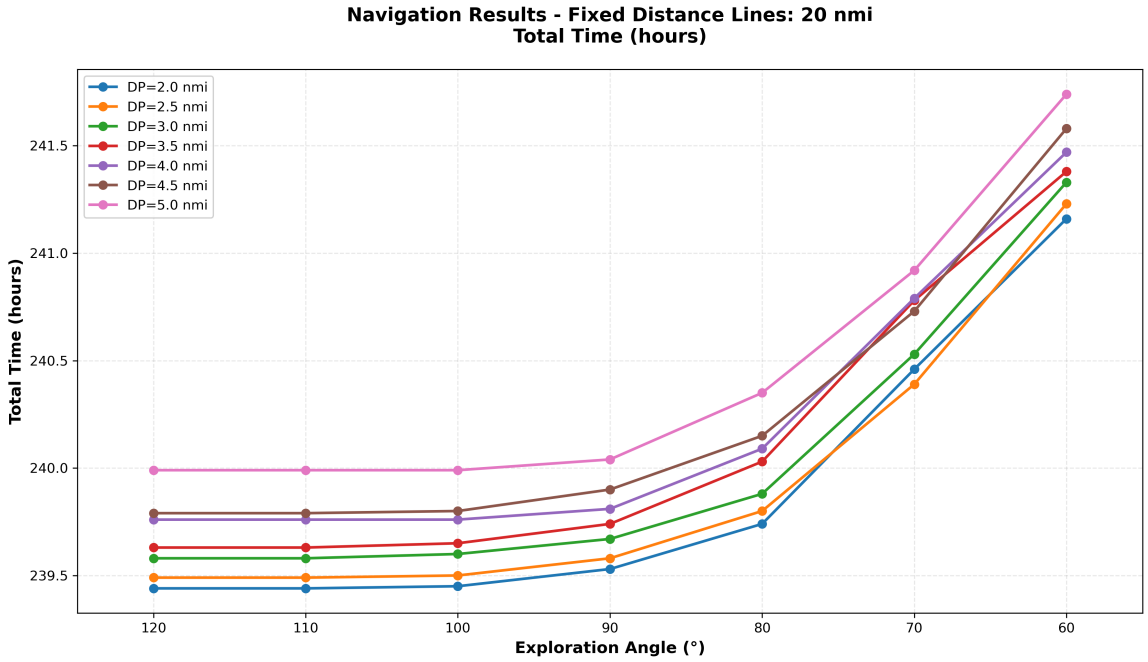


Figure 22 – Navigation results showing total travel time (ETA, in hours) for a fixed stage spacing $D_L = 20$ nmi, across varying exploration angles β and lateral waypoint spacings D_P.

Figure 22 compares the ETAs of the $7^2 = 49$ simulations generated by varying β and D_P under the fixed stage spacing $D_L = 20$ nmi. Each colored curve corresponds to a different value of D_P. For instance, the pink curve ($D_P = 5$ nmi) shows how the ETA changes as the exploration angle is reduced from 120° to 60°. The ETA begins at approximately 240 hours and increases to about 241.5 hours as β decreases. This trend is expected: a smaller exploration angle reduces the number of feasible headings, limiting the algorithm's ability to discover highly optimal local variations in the route.

The same qualitative behavior arises for all other values of D_P: optimality remains stable for large exploration angles, but declines progressively as β becomes narrower. However, a finer inspection reveals an important pattern: across all curves, the ETA remains essentially unchanged for the first three exploration angles $(120°, 110°, 100°)$. Only when β drops to 90° or below does a noticeable deterioration in precision emerge. Interestingly, this stability region persists across all tests of different waypoint spacings D_P and stage spacing D_L.

This suggests that, for this particular transatlantic corridor, an exploration band of $\beta \geq 100°$ is sufficient to recover a globally optimal or near-optimal solution.

Now, if we examine the corresponding execution-time plot for the same set of simulations (Figure 23), again with fixed stage spacing $D_L = 20$ nmi, but now displaying the execution time on the vertical axis, a very clear pattern emerges. As both D_P and β decrease, the execution time consistently decreases as well. This behavior is expected: smaller waypoint spacing reduces the number of lateral nodes per stage, while smaller exploration angles reduce the number of feasible edges that must be evaluated. The combined effect is a substantial reduction in graph density, leading to fewer edge-cost computations and a shorter BFS traversal.

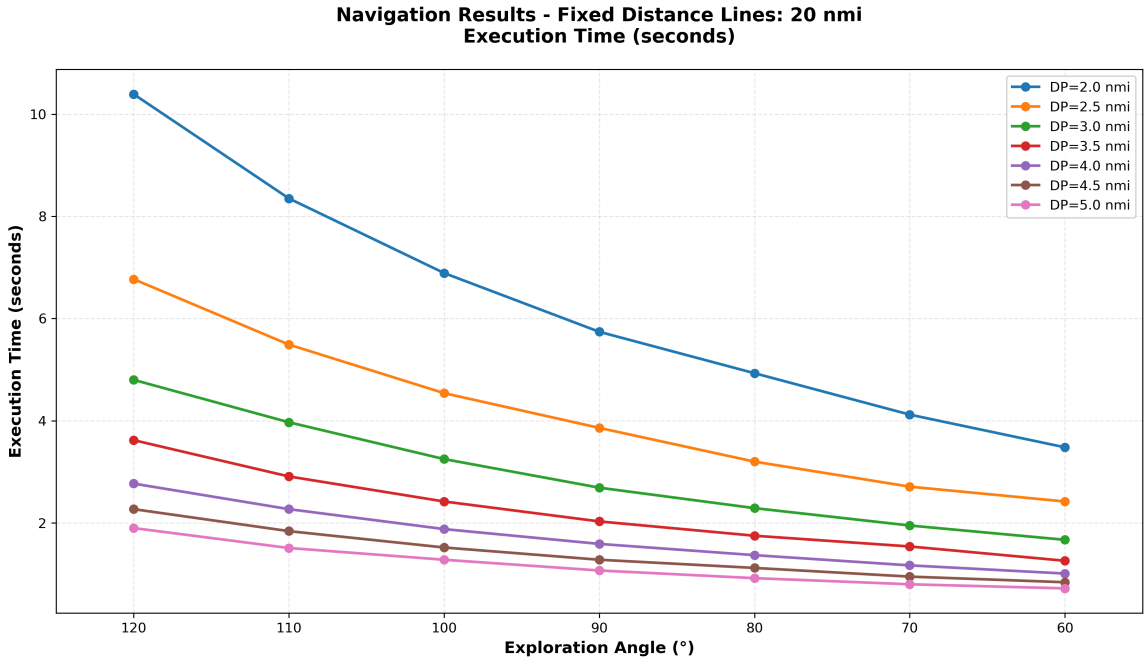


Figure 23 – Execution time (in seconds) for fixed stage spacing $D_L = 20$ nmi, across varying values of lateral spacing D_P and exploration angle β.

When the ETA results (Figure 22) and the execution times (Figure 23) are analyzed together, a strong trade-off becomes evident. Although execution time improves monotonically as β decreases, optimality does *not*. The ETA curves show that route quality remains stable for exploration angles of 120°, 110°, and 100°, but begins to degrade significantly below 100°. Therefore, the exploration angle $\beta = 100°$ represents the optimal compromise:

— values of $\beta < 100°$ reduce execution time but also degrade optimality by restricting the search space too severely;

— values of $\beta > 100°$ increase execution time without improving ETA or route quality.

Thus, for this transatlantic corridor, $\beta = 100°$ emerges as the best-performing configuration, achieving the minimal computational effort necessary to preserve the global optimal route while avoiding the inefficiencies associated with overly broad exploration windows.

With a fixed exploration angle of $\beta = 100°$, we now examine the full performance landscape obtained by evaluating all $7 \times 7 = 49$ combinations of (D_L, D_P). Each configuration is represented in Figure 24 by a labeled marker of the form $\texttt{L}i\texttt{P}j$, indicating the corresponding values of $D_L = D_{L_i}$ and $D_P = D_{P_j}$. The horizontal axis shows the execution time, while the vertical axis shows the resulting ETA (optimality). Colors range from green (best) to red (worst), allowing an immediate visual identification of high-quality configurations.

Several important patterns emerge from this distribution:

— **Cluster of optimal ETAs.** A tight cluster of green and light-green points lies around ETA $\approx$ 239.55 h, corresponding to the most optimal or near-optimal solutions. These configurations use relatively small values of D_L and moderate values of D_P, which create a sufficiently dense graph to capture subtle improvements in the route without excessive computational cost.

— **Execution-time gradient.** As expected, execution time increases toward the right side

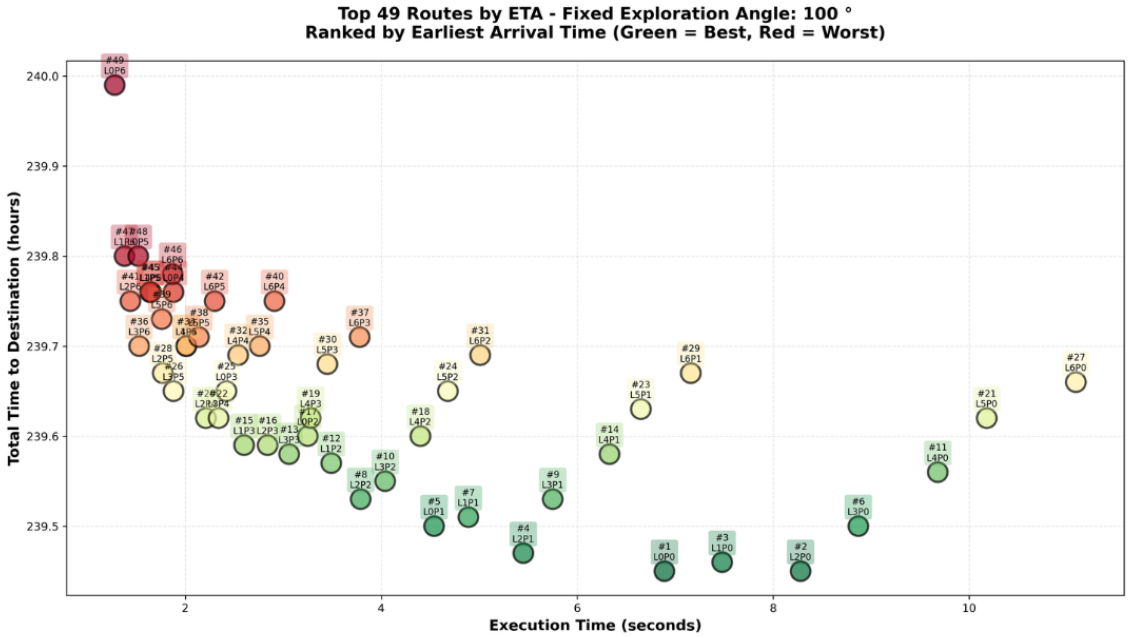


Figure 24 – Scatter plot of all 49 configurations for fixed exploration angle $\beta = 100°$. Each point shows ETA versus execution time, colored from green (best ETA) to red (worst ETA). Labels indicate the parameter codename LiPj.

of the graph. Larger values of D_L (denser stage spacing) and smaller values of D_P (denser lateral spacing) both cause the graph to grow, increasing run time. This explains why the slowest cases (`L0P0`, `L1P0`, etc.) appear toward the far right.

— **Pareto-optimal band.** The green and light-green points near the bottom-left represent the true Pareto frontier: these configurations achieve both low ETA and relatively low execution time. They include, for example, `L2P1`, `L1P2`, `L2P2`, and `L3P2`. These are of particular interest because they preserve optimality while improving efficiency compared to the baseline configuration `L0P0`.

— **Degraded optimality for coarse parameters.** Points in orange and red, generally those with large values of both D_L and D_P, show visibly higher ETAs ($\approx 239.78 - 240.00$ h). In these configurations the graph becomes too coarse, and the algorithm fails to capture the subtle influence of currents and waves on the optimal route.

— **Clear relationship between ETA and speedup.** The top-21 configurations listed in Table 1 confirm what is seen in the scatter plot: many routes offer substantial speedups (up to 3.66×) with only negligible losses in ETA (0.1–0.2 h), illustrating that the algorithm's precision can be preserved without the computational cost associated with the densest graph.

Overall, the scatter plot demonstrates that $\beta = 100°$ provides a large plateau of stable optimality across many configurations of (D_L, D_P). This reinforces the conclusion from the previous analysis: exploration angles above 100° only slow the algorithm without improving precision, while angles below 100° begin to restrict the graph and produce worse ETAs. The table of top-performing routes further shows that carefully chosen combinations of D_L and D_P can yield dramatic gains in execution time while maintaining optimal route quality.

Table 1 – Top 21 routes for fixed exploration angle $\beta = 100°$ with speedup greater than 1.00.

Rank	Codename	D_L (nmi)	D_P (nmi)	ETA (h)	Exec. Time (s)	Speedup
1	L0P0	20.0	2.0	239.45	6.89	1.00
2	L2P1	30.0	2.5	239.47	5.45	1.26
3	L0P1	20.0	2.5	239.50	4.54	1.52
4	L1P1	25.0	2.5	239.51	4.89	1.41
5	L2P2	30.0	3.0	239.53	3.79	1.82
6	L3P1	35.0	2.5	239.53	5.75	1.20
7	L3P2	35.0	3.0	239.55	4.04	1.71
8	L1P2	25.0	3.0	239.57	3.49	1.97
9	L3P3	35.0	3.5	239.58	3.06	2.25
10	L4P1	40.0	2.5	239.58	6.33	1.09
11	L1P3	25.0	3.5	239.59	2.60	2.65
12	L2P3	30.0	3.5	239.59	2.84	2.43
13	L0P2	20.0	3.0	239.60	3.25	2.12
14	L4P2	40.0	3.0	239.60	4.40	1.57
15	L4P3	40.0	3.5	239.62	3.28	2.10
16	L2P4	30.0	4.0	239.62	2.21	3.12
17	L3P4	35.0	4.0	239.62	2.34	2.94
18	L5P1	45.0	2.5	239.63	6.65	1.04
19	L5P2	45.0	3.0	239.65	4.68	1.47
20	L0P3	20.0	3.5	239.65	2.42	2.85
21	L3P5	35.0	4.5	239.65	1.88	3.66

5 Next Steps

The present work demonstrates the feasibility and performance of a fast, three-dimensional weather-routing algorithm based on an N-tree exploration graph and a BFS-style propagation procedure. However, several improvements and extensions can significantly enhance the capabilities, robustness, and operational usability of the system:

1. **Parallelization and high-performance computing.** The current implementation executes graph generation and edge-cost evaluation using a single thread. Since each edge and each stage of the graph can be processed independently, the algorithm is naturally suited for parallel computation. A combined use of MPI (for stage distribution across nodes) and OpenMP (for intra-stage parallelism) could result in substantial performance gains.
2. **Generalized parameter selection across multiple routes.** The parameter analysis presented in this work was conducted using a single transatlantic trajectory located near the Gulf Stream, a region characterized by strong and highly nonlinear current patterns. Although this provides a meaningful stress test for the algorithm, the resulting parameter recommendations are inherently route-specific. A broader study-evaluating multiple departure–arrival pairs across different ocean basins, current regimes, and voyage lengths-is necessary to determine a robust, globally applicable set of parameters $(D_L, D_P, \Delta\text{ETA}, \beta)$. Such an analysis would allow us to identify parameter configurations that consistently yield high-quality routes and computational efficiency across a wide range of environmental and operational scenarios.

3. **Improving graph skeleton topology.** The current N-tree exploration strategy assumes that a reasonably unobstructed ocean corridor exists between the departure and arrival points. When large landmasses or island chains lie between the two, the algorithm may fail to find a valid route because lateral exploration is not included in the base graph structure.

   To address this limitation, a future extension involves computing a preliminary *visibility graph* around land boundaries using algorithms such as A* or Theta*. This visibility graph would supply a set of anchor points circumventing landmasses, after which the N-tree route-search can proceed normally along the oceanic corridors between them. This hybrid approach would combine global land-aware navigation with fast local weather-optimized routing.

4. **Integration of power levels** Although the current system allows the vessel to vary its speed continuously within the bounds $[v_{\min}, v_{\max}]$, real ships do not adjust speed in an analog manner. Instead, engine output is typically controlled through a discrete set of *power levels* or *RPM settings*, each corresponding to a specific service-speed regime. Transitions between these levels are neither instantaneous nor arbitrarily precise, and operators generally avoid frequent or small-scale adjustments due to fuel consumption constraints and engine wear.

   A natural extension of this work is therefore to integrate a discrete power-state model into the routing engine. In such a framework, rather than treating speed as a continuous variable, the vessel would choose among a finite set of admissible power levels, each associated with:
   — a nominal calm-water service speed,
   — a characteristic startup or ramp-up time,
   — an expected fuel-consumption curve,
   — a speed-loss response to waves, wind, and currents.

   Within the 3D graph, each node would carry not only a spatial and temporal dimension, but also a *power-state dimension*. The resulting expansion enables the algorithm to evaluate realistic operational decisions such as reducing power to conserve fuel in heavy seas, or increasing power to meet a narrow ETA window. Although this increases the graph's dimensionality, the layered structure remains compatible with BFS parallelization, especially under a hybrid MPI–OpenMP execution environment.

6 Conclusion

This work introduces a novel weather-routing methodology by constructing a full three-dimensional N-tree exploration graph and propagating costs through a BFS-based layered traversal. The resulting algorithm is not only conceptually simpler but also significantly faster: for small and medium-length voyages, optimal routes can often be computed in tenths of a second, enabling near real-time voyage evaluation.

A key advantage of the proposed framework is its ability to incorporate variable vessel speeds. Traditional routing models frequently assume a fixed engine power or a constant service speed; in contrast, our approach evaluates a range of feasible speeds at every stage, capturing the full operational flexibility of modern vessels and enabling optimization based on speed loss, speed gain, or fuel efficiency.

Overall, the methodology presented here demonstrates that high-fidelity, time-aware routing can be performed with great computational efficiency when the graph structure is designed appropriately and much faster than most of the current commercial solutions. The approach forms a solid foundation for future developments in large-scale parallel routing, adaptive resolution, visibility-graph integration, and operational decision-support systems for commercial maritime navigation.

7 References

[1] History of navigation https://en.wikipedia.org/wiki/History_of_navigation

[2] Helong Wang, Wengang Mao, Leif Eriksson, *A Three-Dimensional Dijkstra's algorithm for multi-objective ship voyage optimization*, Ocean Engineering, Volume 186, 2019, 106131. doi:10.1016/j.oceaneng.2019.106131.

[3] Amphitrite SA - Ocean Data Ingelligence https://www.amphitrite.fr/

[4] The great circle distance. In Wikipedia. https://en.wikipedia.org/wiki/Great-circle_distance

[5] Loxodromic navigation. In Wikipedia. https://en.wikipedia.org/wiki/Loxodromic_navigation