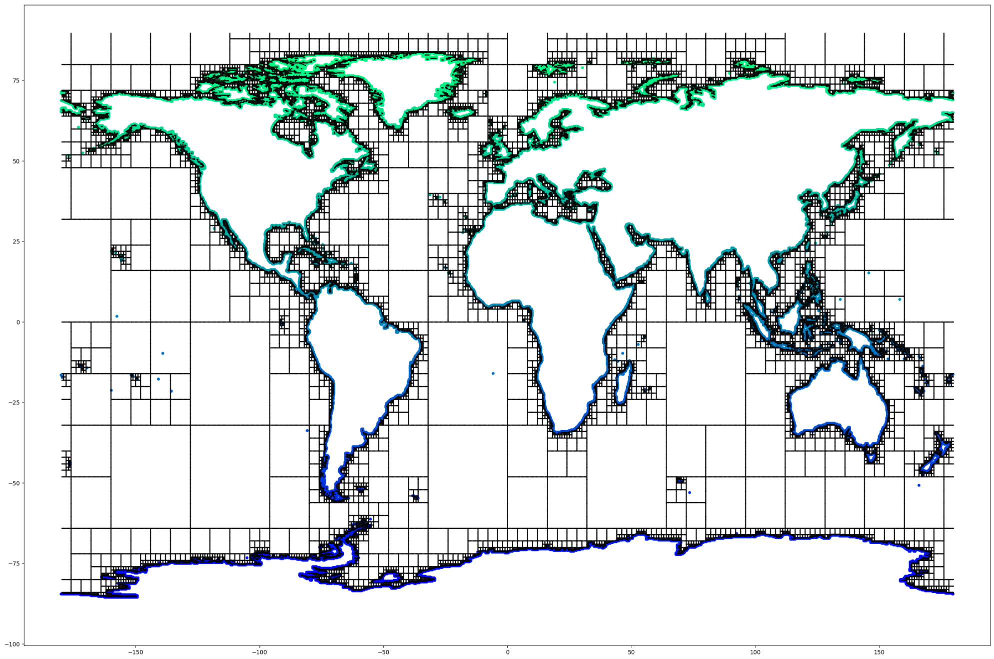
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Algorithms for Wind-Powered Cargo Ship Routing

A Framed Quadtree Approach with Graph Weight Interpolation

Master's thesis in Computer science and engineering

Björn Leskinen, Gabriel Käll

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Algorithms for Wind-Powered Cargo Ship Routing

A Framed Quadtree Approach with Graph Weight Interpolation

Björn Leskinen, Gabriel Käll

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Gothenburg, Sweden 2024

Algorithms for Wind-Powered Cargo Ship Routing
A Framed Quadtree Approach with Graph Weight Interpolation
Björn Leskinen, Gabriel Käll

Supervisor: Peter Damaschke, Computer Science and Engineering
Advisor: Amelie Binder, CargoKite GmbH
Examiner: Carl-Johan Seger, Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Tell 'em to bring out the whole ocean!

Typeset in LaTeX
Gothenburg, Sweden 2024

Algorithms for Wind-Powered Cargo Ship Routing
A Framed Quadtree Approach with Graph Weight Interpolation
Björn Leskinen, Gabriel Käll
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis explores a novel approach to long-distance ship weather routing by employing a quadtree data structure to represent the ocean surface, combined with an approach to interpolating graph weights used by a modified Dijkstra's Algorithm. This approach allows access to a wider range of relative wind angles compared to using a uniform grid.

This study evaluates the performance of the interpolation technique, specifically examining how quadtree subdivision levels (or bounding box size) relate to interpolation accuracy. The findings indicate that the interpolation method performs similarly over a varying range of bounding box sizes if one assumes a relatively high engine-driven calm water speed. The auxillary electric engine is used when wind speeds can't propel the vessel above the calm water speed. In some cases, larger bounding boxes yield better results, however it is possible this was due to the weather forecast deviating from the actual historical data, or approximation errors in the graph weight interpolation. The study highlights the importance of selecting an appropriate default calm-water speed, as it influences the accuracy of the interpolation method.

The interpolation algorithm introduces runtime overhead to the path planning algorithm, especially when the size of the bounding box is large. Memory savings are significant, even though the quadtree was only subdivided in the local area around the path. The best trade-off between memory and runtime savings is achieved with a bounding box size of 222-111 km.

Future work should focus on refining default speed selection, incorporating additional weather data, and further optimizing the quadtree framework to improve efficiency and robustness in real-world maritime applications.

Keywords: Ship Weather Routing Problem, SWRP, Quadtree, CargoKite, Autonomous Sailing Vessel, USV, Multi Resolution Path Planning.

Acknowledgements

We would like to thank Amelie and the people working at CargoKite for providing us with the opportunity to be involved in this project. Special thanks to Amelie Binder, Peter Damaschke, Filip Nordmark, Edvard Heinmetz, Robin Sandblom and Ziyu Fang for the support through this projects ups and downs.

Björn Leskinen, Gabriel Käll, Gothenburg, 2024-07-13

Contents

List of Figures

1

Introduction

The maritime shipping industry is a significant contributor to global carbon emissions, accounting for approximately 915 million metric tons of CO annually [1]. It is also estimated that around 90% of manufactured goods are transported by ship. As concerns mount over port capacity constraints and the increasing size of container ships [2], researchers are exploring the potential of smaller, possibly autonomous wind-powered vessels as a viable alternative [3]. These smaller vessels could alleviate pressure on port infrastructure, mitigate negative impacts on aquatic ecosystems, and potentially reduce global economic costs [4].

In an attempt to decarbonize shipping and alleviate the downsides of these larger ships, CargoKite GmbH has proposed a concept for an autonomous cargo vessel propelled by a kite with an auxiliary electric motor [5]. The batteries that power this electric motor can also be charged while utilizing wind propulsion, potentially making the ship suitable for long distance travel. The proposed ship will utilize an Airborne Wind Energy System (AWES) - a system that includes a kite flying at 300 meters. Therefore, the ship is designed to harness high-altitude winds as its primary means of propulsion.

Weather routing – the practice of leveraging wind and other weather conditions to achieve higher speeds and/or save fuel, is a known challenge in both competitive sailing and commercial shipping. However, the routing problem posed by CargoKite's ship concept differs from traditional approaches. Unlike conventional vessels where fuel consumption is often the primary optimization parameter, with travel time sometimes compromised for lower fuel usage, CargoKite's wind-powered vessel presents a different set of optimization challenges.

For this ship concept, fuel consumption is not relevant, and the length of the path becomes less critical. Instead, the key parameters to optimize are maximum speed and minimum travel time. This shift in optimization priorities necessitates a different approach to route planning, one that is tailored to the characteristics of wind-powered vessels.

In the dynamic and time-dependent environment of oceanic navigation, the efficiency of autonomous sailing vessels partly depends on the reliability of routing algorithms. The ship weather routing problem (SWRP) aims to find the shortest path for a vessel, considering predicted weather conditions at each point along the route. Existing approaches often favor graph algorithms like Dijkstra's and A* to compute optimal paths, representing the ocean surface using a uniform grid [6, 7, 8, 9].

Solving the ship weather routing problem on a uniform grid presents several challenges. Firstly, the direction of travel is limited, restricting the vessels ability to adjust its orientation relative to the wind. Secondly, planning a long-distance route can lead to significant memory consumption, especially considering that time-dependent wind data must be maintained for every grid cell. Additionally, increasing the resolution yields a larger search space, requiring more computational resources, thus increasing runtime.

An alternative data structure to a uniform grid representation is a quadtree, suitable for sparse data. Quadtrees recursively subdivide data into four quadrants until some terminating condition is met. An example of such a condition is when only a few data points remain within a quadrant, indicating that further subdivision is unnecessary, as the complexity of the data has been sufficiently outlined. Sparse or homogeneous regions would be represented by larger quadrants, while regions with complex data would be divided into smaller quadrants. Utilizing these characteristics to store information about the ocean, such as the locations of large landmasses or homogeneous wind patterns can reduce memory consumption and runtime for long-distance ship weather routing.

Proposed Approach

In this paper, we utilize a quadtree to represent the ocean and apply it to plan long-distance ship routes using a modified version of Dijkstra's Algorithm. Only partial weather data is stored in the quadtree, specifically on the borders of each quadrant. Therefore, interpolation is employed to estimate travel times within a quadrant. This approach yields access to a wider range of relative wind angles when planning a path. Different levels of subdivision, representing different quadtree resolutions, are compared to determine the effectiveness of the interpolation method and its performance on time-dependent wind data.

1.1 Related Work

Previous research conducted regarding the application of quadtrees to an ocean environment consider a purely engine-driven vessel. None, to our knowledge, applied this approach to a mainly wind-powered vessel. Shah et al. proposes an approach where landmasses were stored in a high-resolution quadtree and used to create a visibility graph that is employed to navigate land-dense regions more intelligently [9]. Specifically, the visibility graph was used to design a heuristic that weighted nodes based on the shape and size of the landmasses that obscured the straight-line path from a node to the goal. However, wind is not accounted for in this paper, and more focus is placed on tide since this impacts the shapes of the landmasses.

Lee et al. also propose a visibility graph approach to path planning, using a Polygon Map Random (PMR) quadtree data structure to represent the coastline and islands in a marine environment [10]. The visibility graph is generated using a predetermined set of path waypoints and subsequently used to calculate the shortest path from start to goal. The authors use an empirical formula to calculate the speed reduction of an engine-driven vessel due to wind and waves, compared to its calm water cruising speed. They also test the algorithm on a high-resolution map of the coastline. However, this map was limited to less than one degree latitude and longitude squared.

Stentz and Goldberg propose "Field D*", an algorithm that addresses heading angle limitations when planning paths for mobile robots [11]. Instead of using a uniform grid-based approach where heading angles are limited to multiples of $\pi/4$, they introduce an interpolation-based algorithm to estimate traversing across a cell. Given a cell with four edges situated on the border, the algorithm uses a linear combination of the costs of traversing between two neighboring nodes to estimate the cost of traversing across the cell, with an endpoint anywhere on the target cell's border. This allows the algorithm to plan with a continuous heading angle.

The same authors also developed a Multi-Resolution version of Field D*, which operates on quadtrees [12]. The algorithm allows representing the environment with a coarse low-resolution prior map, which is updated with high-resolution sensor data in the robot's field of view during traversal. This approach enables a less restrictive heading angle and reduces time and space complexity compared to a uniform grid by focusing the level of detail on the most important parts of the problem. Therefore, the algorithm is well-suited for handling larger maps of terrain. It estimates the cost of traversing through the quadtree cells using interpolation, similar to the original Field D* algorithm, but now applied to cells of varying sizes. Note that the environment in the paper represented static terrain of varying resolution, and no testing was done in time-dependent environments.

Path Planning in Road Networks

The A* landmarks and triangle inequality (ALT) method uses landmarks, which are selected nodes with precalculated distances to all other nodes [13]. These nodes are selected during a preprocessing stage, and their distances are used to obtain a heuristic that is a tighter lower bound on the distance to the goal. The quality of the lower bound highly depends on the selection of landmarks, and optimal landmark selection is an NP-hard problem.

ALT algorithms are suitable for road networks since landmarks rarely need to be recalculated and provide correct paths as long as edge costs don't drop below an initial value [14]. However, in ship weather routing, the wind patterns represent a road network where the layout may change significantly during a short period, making ALT algorithms less suitable.

Calculating the cost from the landmarks to all nodes is an expensive task. Combining the large search spaces present in a long-distance SWRP and the fact that optimal landmark selection is an NP-hard problem, approximate solutions may need to be considered. Additionally, the dynamic and stochastic nature of edge weights means that the calculated bounds might not hold in the future. If edge weights do not decrease below an initial value, then the calculated bounds will still hold, as assumed in [14]. This assumption works in a road network since the shortest travel time will be achieved when there is no traffic. However, it is not clear what the equivalent scenario is in a dynamic ocean environment. One idea could be to assume "optimal" weather conditions, calculating the distances with the boat's top speed in mind.

Time-Dependent Dijkstra's Algorithm and Bidirectional Search

In some domains, the cost to travel between nodes in a network can vary depending on the current time. This time-dependent nature of edge costs poses challenges for traditional shortest path algorithms like Dijkstra's. To address this, Ding et al. introduced a time-dependent version of Dijkstra's Algorithm that can handle edge costs that are functions of time and also search for an appropriate start time to minimize path cost [15]. There have been various implementations of time-dependent algorithms that use heuristic functions [16, 17]. Nannicini et al. apply a bidirectional search, where the backward search is used for bounding the set of nodes that will be explored by the forward search. Since the arrival time is not known in advance, the backward search needed to be time-independent [16].

2

Background

In this section we provide the theoretical foundations and concepts that the work builds upon. It begins by delving into the general path planning problem, which involves finding an optimal sequence of states transitioning from a start to a goal state on a graph, while minimizing the total transition cost. It covers Dijkstra's Algorithm as a fundamental approach to this problem.

The section then explores quadtrees, a data structure used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. Quadtrees are particularly useful in storing sparse data, which can significantly reduce memory consumption while path planning in sparse but complex environments.

Finally, the section addresses path planning in a time-dependent space, where transition costs may vary over time.

2.1 Path Planning

The single-source-single-destination path planning problem is defined on a finite graph with a finite set of states S within the graph. Let $\mathrm{succ}(s)$ denote the set of successor states of state $s \in S$, and $\mathrm{pred}(s)$ denotes the set of predecessor states of s. For any pair of states s and $s' \in \mathrm{succ}(s)$ the cost of transitioning from s to s' is given by $0 < c(s, s') \leq \infty$. Given $s_{start} = s_0$ and $s_{goal} = s_k$ a path planning algorithm finds a sequence of states:

$$\pi := s_0, s_1, ..., s_k \text{ where for all } s_i \in \pi, s_i \in \mathrm{succ}(s_{i-1})$$

The cost of the path is given by $\sum_{i=1}^{k} c(s_{i-1}, s_i)$. Let $c^*(s, s')$ denote the cost of the cheapest path from s to s'.

The goal of a shortest path algorithm is to find the path π^* that starts at s_{start} and ends at s_{goal}, for which $\sum_{i=1}^{k} c(\pi^*_{i-1}, \pi^*_i) = c^*(s_{start}, s_{goal})$. Let $c^*(s_{start}, s) = g^*(s)$. If $g^*(s)$ is known for all states $s \in S$, then such a path can be found by starting at the goal state, and at every state s_i picking a state $s_{i-1} = \mathrm{argmin}_{s' \in \mathrm{pred}(s_i)} g^*(s')$ until $s_{i-1} = s_{start}$ [18].

Dijkstra's Algorithm

Dijkstra's Algorithm is a procedure that solves the single source, single destination shortest path problem [19]. It does this by having a representation of the cost of the best path from start to the current state s, $g(s)$ (cost-to-come). Initially, all states are assigned an infinite g-value, except for the start state s_{start}, which has $g(s_{start}) = 0$. These g-values are stored in a priority queue called OPEN.

During each iteration, the algorithm extracts the state s with the smallest g-value from the OPEN list and expands it. When expanding a state s, the algorithm checks if passing through s to reach any of its successor states $s' \in succ(s)$ yields a lower g-value than the current $g(s')$. If a shorter path is found, the g-value of s' is updated to the cost of the new path, making s' a candidate for future expansion. This cost update propagates to the successors of s'.

The algorithm terminates when there are no more reachable states that haven't been expanded, i.e., when $\forall s \in S : g(s) = g^*(s)$, where $g^*(s)$ is the cost of the shortest path from s_{start} to s. In other words, it terminates when either the goal state has been found, or no more g-values can be improved, meaning the OPEN list is empty, and no state remain for future expansion.

Once terminated, a shortest path can be extracted greedily by following the optimal predecessor state from the goal state back to the start state, as described in Section 2.1.

2.2 Quadtrees

A quadtree is a hierarchical representation of a 2-dimensional space in a tree data structure. Based on a specific condition or criteria, the root node is subdivided into four equal quadrants or bounding boxes. This subdivision process continues recursively for each child node until either a maximum resolution level is reached or the condition for further subdivision is no longer satisfied, these nodes are called leaf nodes [20].

In the case of a point-based Point Region (PR) quadtree, the subdivision condition is typically based on the number of points contained within a node. If the number of points in a node exceeds a predefined splitting threshold, the node is subdivided into four child nodes, and the points are distributed among these child nodes based on their spatial locations [21].

In the standard quadtree representation, paths generated between two points are constrained to segments connecting the centers of the bounding boxes (Figure 2.1). This limitation can lead to suboptimal paths with unnatural changes in direction, especially in sparse environments with large empty areas [22].

Framed quadtrees are a variation of the quadtree data structure where a layer of highest resolution border cells, called “frames", is added around the perimeter of each bounding box (Figure 2.2). Planning a path on frames allows for more directions of travel, enabling paths to closer approximate optimal paths. While framed quadtrees improve path quality, they require more memory than standard quadtrees, due to the overhead of maintaining the additional border cells [23]. Assuming the length of the side of the bounding box for leaf nodes is n, a standard quadtree requires one node containing data per bounding box. In contrast, a framed quadtree would require

$4n - 4$ nodes per bounding box, whereas a uniform grid at the highest resolution would require n^2 nodes per bounding box.

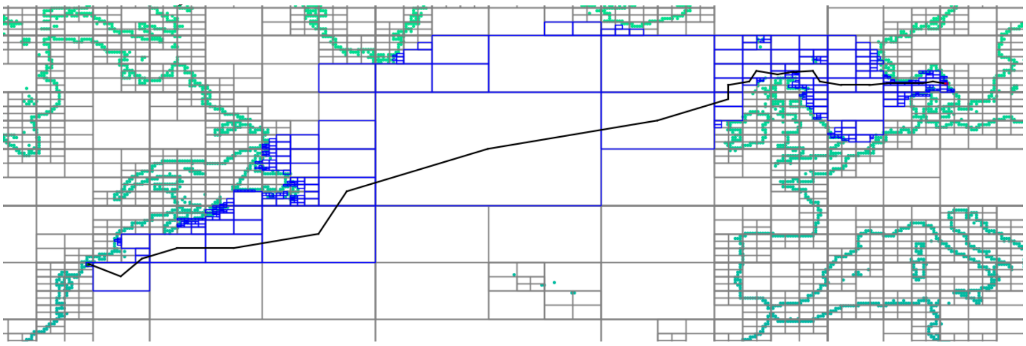


Figure 2.1: A path planned using midpoints of bounding boxes.

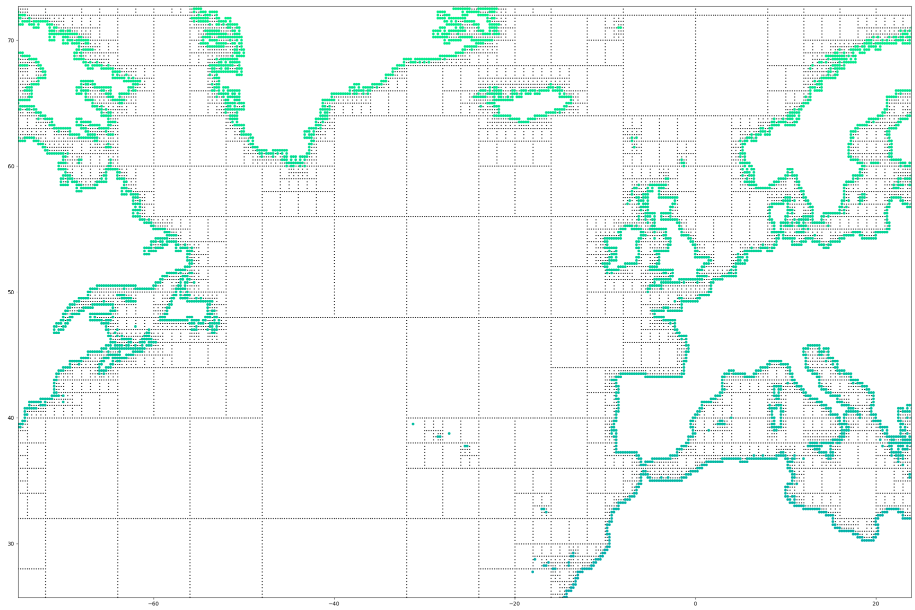


Figure 2.2: An example of an initial framed quadtree subdivided based on border data.

2.3 Time Dependency

If the cost of transitioning in a state space is dependent on at what time that transitioning occurs, the state space is said to be time-dependent. The non time-dependent representation described in Section 2.1 can be extended to fit in a time-dependent scenario in the following way:

Let T be the time window for the algorithm, that is the set of time steps between the start time and end time. $[a_s, b_s]$ be the time window within it is possible to visit state s. Then for states $s, s' \in S : s' \in succ(s)$ the cost of transitioning from s to s' at time step $t \in T$ is given by $c_{s,s'}(t)$.

Therefore, every transition will have a list of costs one for each possible time step in the discrete case. When the algorithm traverses the state space it needs to keep track of at what time it makes the transition and use the appropriate cost. Therefore, for a start time t_0 the Time-Dependent Shortest Path (TDSP) problem is to find a path between s_{source} and s_{goal} that leaves at time t_0 and minimizes the arrival time while satisfying all of the constraints of the time window at every state.

3

Method

This section describes the approach taken to represent the ocean surface with a framed quadtree data structure, and how path planning is applied to this representation. It details the process of constructing a quadtree to model the ocean surface, and how wind data is incorporated. An overview of the interpolation method used to approximate travel times when crossing a bounding box is then provided, along with how a node is expanded in the modified time-dependent Djikstra's Algorithm. Additionally, the section outlines the methodology employed to post-process an interpolated path in order to gather relevant evaluation metrics.

3.1 Constructing the Quadtree

The wind dataset that is used during the construction of the quadtree is provided by the National Oceanic and Atmospheric Administration (NOAA) and is a 279-hour forecast sampled every 3 hours [24]. This forecast contains the wind speed and direction for cells of approximately 111x111 kilometers in size, or 1 degree of latitude/longitude. Linear interpolation is applied to this dataset to increase the resolution to 0.25x0.25 cells of latitude/longitude. Thereafter, a quadtree is created representing the graph in the following manner:

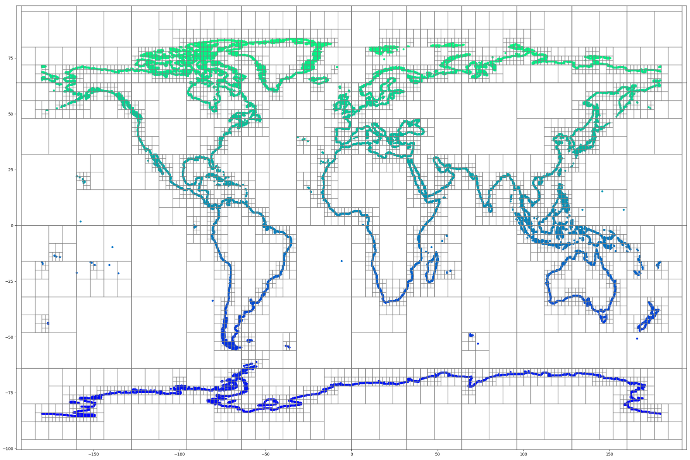
1. The whole ocean area is first flattened into a dataset of coordinates using the latitude and longitude coordinates provided in the interpolated wind data. A mask is then created to identify the ocean data points versus the land data points.

2. A border convolution is applied on the mask to identify the border regions between the ocean and land. This is done by applying a 3x3 kernel where the border is marked with ones and the center with -8. After applying this convolution, the new mask is filtered where the convolution result is greater than 0.

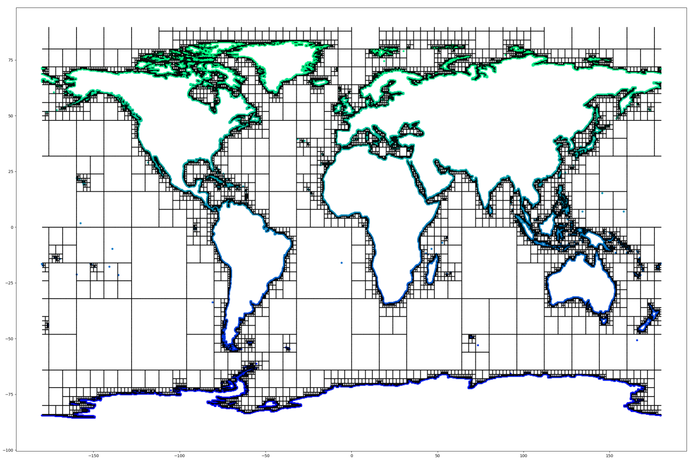
3. A quadtree is then created using the border coordinates as the initial nodes. The quadtree is recursively subdivided until each leaf node contains a maximum number of 1 border coordinate. This promotes the quadtree to stop subdividing in landmasses and ocean surfaces, only subdividing close to borders. The result of this operation is also visible in Figure 3.1a.

4. Frame nodes of size .25x.25 are assigned to each bounding box by recursively

filtering the dataset of coordinates and assigning the coordinates that align with the border of each box. Adjacent bounding boxes share frame nodes along their common borders. The result of this operation is shown in 3.1b. A dictionary $frame\_dict : frame\_node \rightarrow list(quadtree\_node)$ is maintained, providing $O(1)$ lookups between a frame node and its parents.



(a) Quadtree representing the ocean surface, subdivided based on border coordinates.



(b) Quadtree representing the ocean surface, subdivided based on border coordinates.

Figure 3.1

3.2 Calculating Boat Speeds

Given two coordinates a and b that are .25 degrees apart, and the bearing angle between them θ (geodesic angle where $0°$ points north), a boat speed is calculated using the following method:

1. Acquire the wind speed s and direction ϕ for node b.
2. Calculate the signed angle difference between ϕ and θ. The result is the relative wind angle in node b.
3. Given the speed and relative wind angle, calculate the boat speed assuming the sail will be adjusted optimally. This estimation is performed by applying linear interpolation to a vehicle dynamics model, creating the function max_boat_speed($speed, relative\_wind\_dir$). The vehicle dynamics model was provided by CargoKite.

3.3 Algorithm Description

In this section we provide an example to describe how the modified Dijkstra's Algorithm expands a node and calculates graph weights. We start by showing a small part of a framed quadtree search space in Figure 3.2. The figure provides a zoomed-in look at how the borders of the bounding boxes appear and the spacing between them.

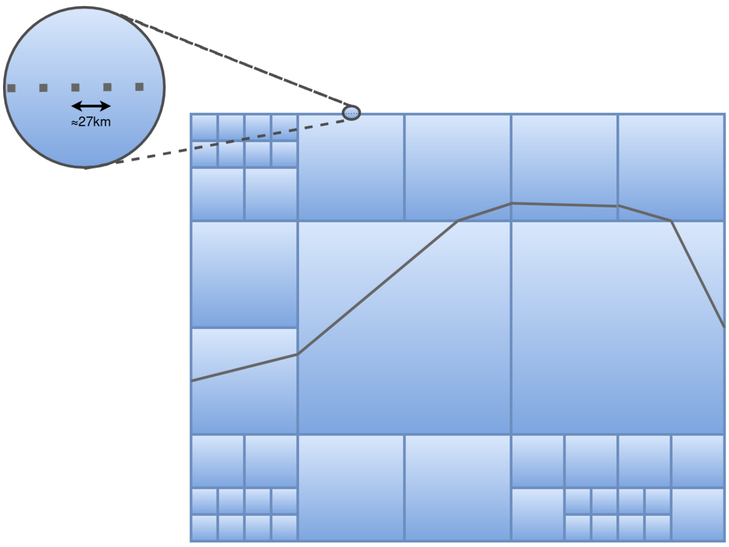


Figure 3.2: Example search space and path on the framed quadtree.

We then move on to expanding a node. Assume that Dijkstra's Algorithm has already picked an appropriate entry node in the highlighted bounding box and arrived there at time t. From this point, we want to estimate the travel time from the entry point to all adjacent frame nodes and add them to the OPEN list (Figure 3.3).

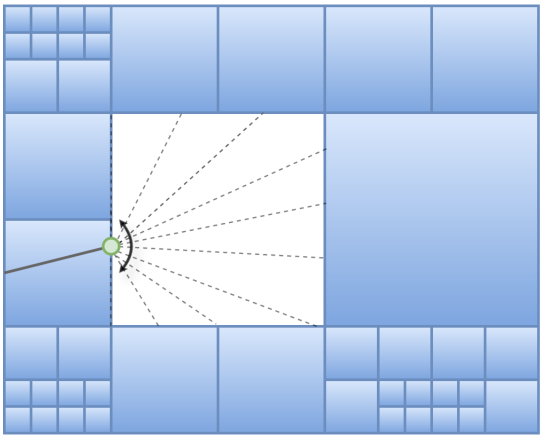
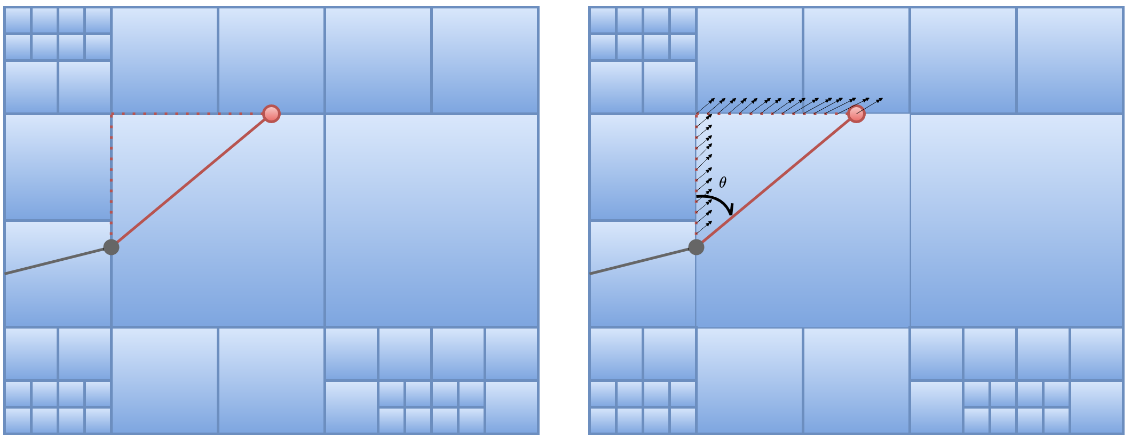


Figure 3.3: Picking an appropriate exit point given an entry point (green dot).

To add these nodes to the OPEN list, we need to estimate the travel time from the entry to all adjacent framed nodes. For this, we employ the Interpolation Method, which will be described in more detail in Section 3.4. This method uses the wind data on the bounding box border and steps through the perimeter of the box, calculating a cumulative travel-time from entry to exit point.

Before calculating the travel-time we first need to index the wind data according to the bearing angle θ between the entry and exit point (3.4a). This step can be seen in Figure 3.4b. Note that the wind direction is not plotted here, and the bearing angle will be used in conjunction with the wind direction to calculate the maximum boat speed according to Section 3.2.



(a) Entry and exit point on a bounding box in a framed quadtree.

(b) Indexing the wind data with the bearing angle between entry and exit point.

Figure 3.4: Example showing entry and exit points on a bounding box and how wind data is indexed according to the bearing angle between them.

Subsequently, we step through the perimeter of the bounding box, calculating two cumulative travel-times, one in north-south direction (a) and one in east-west direction (b). The cost-to-come value of the exit point node is then set to $t + \sqrt{a^2 + b^2}$, and added to the OPEN list. This represents the travel time from the start vertex of the search to a given node, and is used for the Dijkstra's Algorithm node expansion order (explained further in Section 2.1). This step is shown in Figure 3.5.

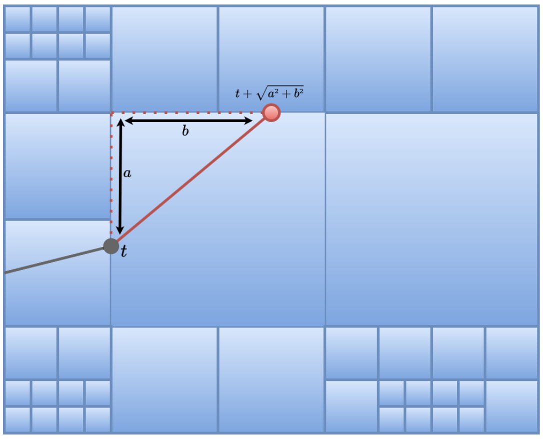


Figure 3.5: Indexing the wind data with the bearing angle between entry and exit point.

After calculating the graph weights for all entry-exit point pairs and adding them to the OPEN, we can continue running Djikstra's Algorithm, repeating all aforementioned steps, and aquiring a path. Assume that the rightmost dot shown in Figure 3.6 is the goal node.

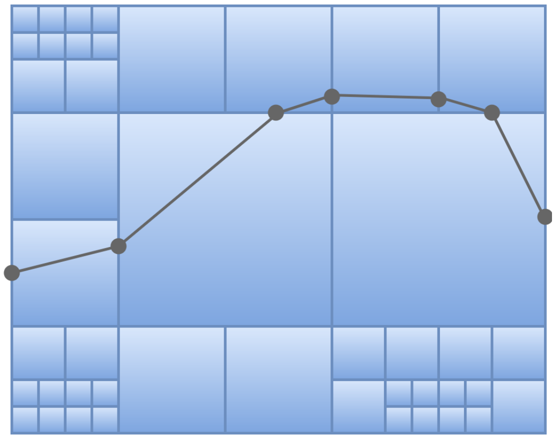


Figure 3.6: Example of a path calculated on a framed quadtree.

3.4 Interpolation for Estimating Graph Weights

In this section, we present how we used interpolation to estimate graph weights while crossing a bounding box. The interpolation function is divided into two methods: `calculate_tupled_path_cost` and `interpolate_node`.

The `calculate_tupled_path_cost` method computes the cumulative cost and time to traverse a given path segment along the frame of a bounding box based on wind speed and direction data. It proceeds as follows:

1. Loop through each pair of points in the path segment.
2. For each pair, fetch wind data and distance between nodes for the current time step.
3. Compute the boat speed and the travel cost.
4. Accumulate the total cost along the path, updating the time.
5. When the path has been completed, return the accumulated weight and time.

The `interpolate_node` method interpolates the cost between an entry point and an exit point on a frame. It starts by generating a path along the frame of the box in north-south and east-west directions. The method follows these steps:

1. Slice the frame to acquire a path segment that traverses the frame from entry to exit point.
2. Tuple the path segment.
3. Calculate the bearing angle θ. This angle is used to calculate the maximum boat speed for every pair of points along the segment. Use the bearing angle to index the wind data (Figure 3.4b).
4. Use `calculate_tupled_path_cost` to compute the cost for each path segment (Figure 3.5).
5. Aggregate the costs of the path segments, handling **invalid paths** by returning a high cost. A path is considered invalid if the weight between two nodes is ∞, caused by a boat speed being 0.
6. Return the final interpolated cost.

To avoid invalid paths where some boat speed along the line segment is 0, one can assume a **default boat speed** that represents the calm-water speed of using an auxiliary electric engine.

3.5 The Actual Duration of a Path

The path generated by the algorithm relies on estimating the arrival time at each point along the route. However, inaccuracies in these estimates can lead to the vessel encountering different wind conditions than initially anticipated during planning, potentially resulting in deviations from the expected timeline and less accurate routing.

Moreover, paths generated by interpolating across multiple resolutions may not be spaced evenly at a .25x.25 resolution. To evaluate a path consisting of line segments, we need to post-process it, stepping through each line segment at the highest resolution possible and using the intermediate wind data on each line segment (Figure 3.7).

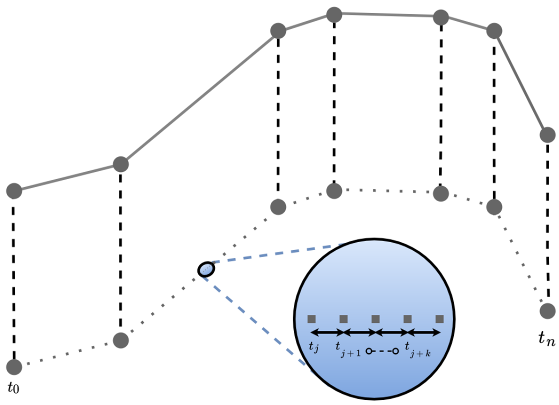


Figure 3.7: Post-processing a path calculated on a framed quadtree in order to access intermediate data on each line segment. The path time is calculated by stepping though the points, starting at t_0.

For both of these reasons, many evaluation metrics need to be reacquired during an evaluation phase that involves systematically stepping through the data after the path is generated.

The evaluation phase consists of the following steps:

1. Iterate through the pairs of consecutive coordinates along the calculated path.
2. For each pair of coordinates:
   (a) Determine the line segment connecting the coordinates.
   (b) Iterate through the coordinates along the line segment.
   (c) For each pair of consecutive coordinates a, b along the line segment:
      i. Calculate the distance and bearing angle θ between the coordinates.
      ii. Retrieve the wind speed and direction for coordinate b from the wind data for the current time.
      iii. Calculate the boat speed based on the relative wind angle between the wind direction and the bearing angle, as described in Section 3.2.

  iv. If the boat speed is below the calm-water engine speed, set the speed to the calm-water speed for electric assistance and update the time and distance spent with electric assistance.
  v. Calculate the time required to travel the distance between the coordinates based on the boat speed.
  vi. Update the current time and append it to the list of path times.

3. Return the distance traversed with electric assistance, the list of path times, the total distance traveled, and the distance traveled with electric assistance.

This procedure accounts for wind conditions at each point along the calculated path, adjusting the boat's speed accordingly. It assumes electric assistance when the boat's speed falls below the assumed calm-water speed. The output of the procedure includes the actual time required to complete the path and the percentage of distance traversed with electric assistance.

4 Experimental Results

In this section, we present the outcomes of planning three routes with varying default speeds and different levels of subdivision. Each path was initially planned on the quadtree subdivided on border points (see Figure 3.1b). The quadtree was then re-initialized with different maximum allowed bounding box sizes in each partition, for example, a maximum side length of 444km. This means all bounding boxes are subdivided until they have a side length which is less than 444km. The highest-resolution quadtree had a bounding box size of 27km.

The first route stretches from Gothenburg to New York City, the second from Gothenburg to Portugal, referred to as the "UK" path due to its passage through the British Channel, and the third from Luanda to Tramandai, which we call the "Brazil" path.

The paths were all planned using the same weather forecast, spanning 0 to 243 hours. The actual recorded data spans 0 to 278 hours, and is used in a post-processing stage to calculate the **actual duration**. This duration is acquired by stepping through the forecast as described in Section 3.5. This is necessary because both interpolation and forecast innacuracies may cause the expected path duration to be inaccurate. The percentage of assisted kilometers (percentage of kilometers traveled using the auxillary electric engine) was also calculated in the post-processing stage, using the historical data.

Each path was planned using four default speeds that are multiples of 15 knots ($\approx$7.72m/s). The 15 knot speed was chosen because CargoKite estimated that the maximum calm water speed of their prototype vessel would be similar. All figures that show the paths also depict a heatmap illustrating wind direction variance within the bounding boxes in the partition. Colors closer to the bright-yellow end of the spectrum correlate with higher variance, while darker purple colors correlate to low variance. The variance is plotted for all timesteps in the dataset, and is not related to the path planning algorithms entry or exit time in the bounding box.

All results were obtained on a computer with 48GB of RAM allocated to Windows Sub-System for Linux (WSL), running on an Intel Core i7-13700K. WSL seemed to limit CPU usage substantially, as it was never above 20%.

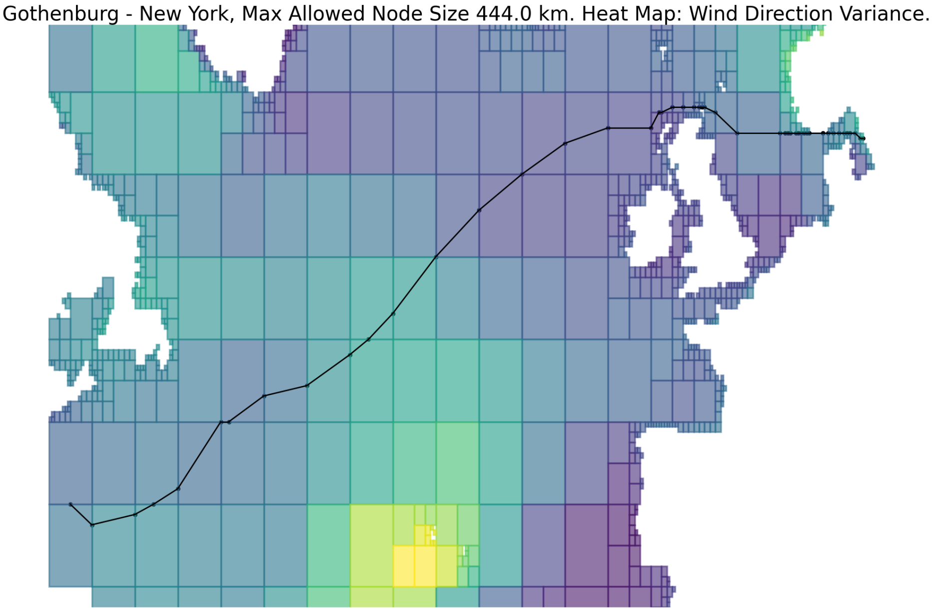


Figure 4.1: Gothenburg to New York, default speed 15 knots.

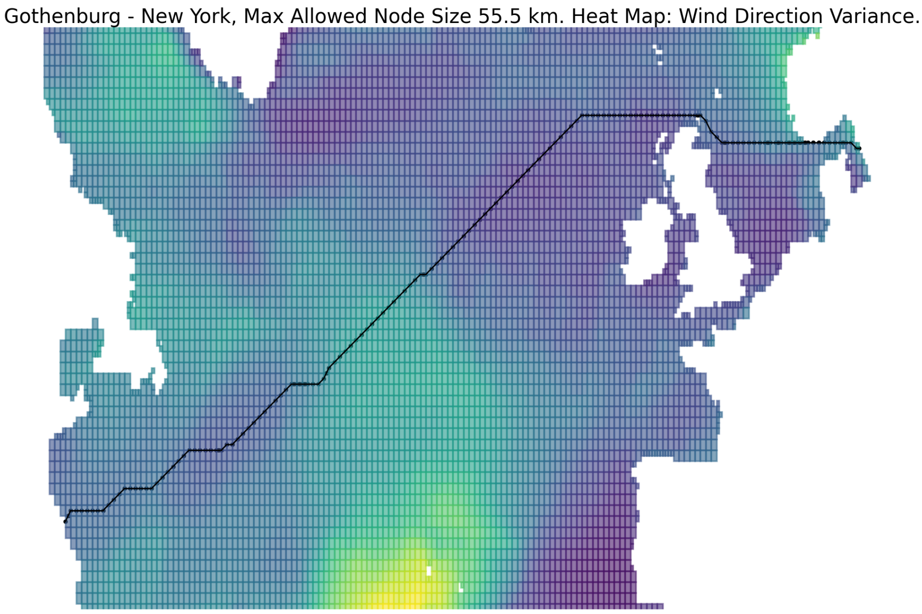


Figure 4.2: Gothenburg to New York, default speed 15 knots.

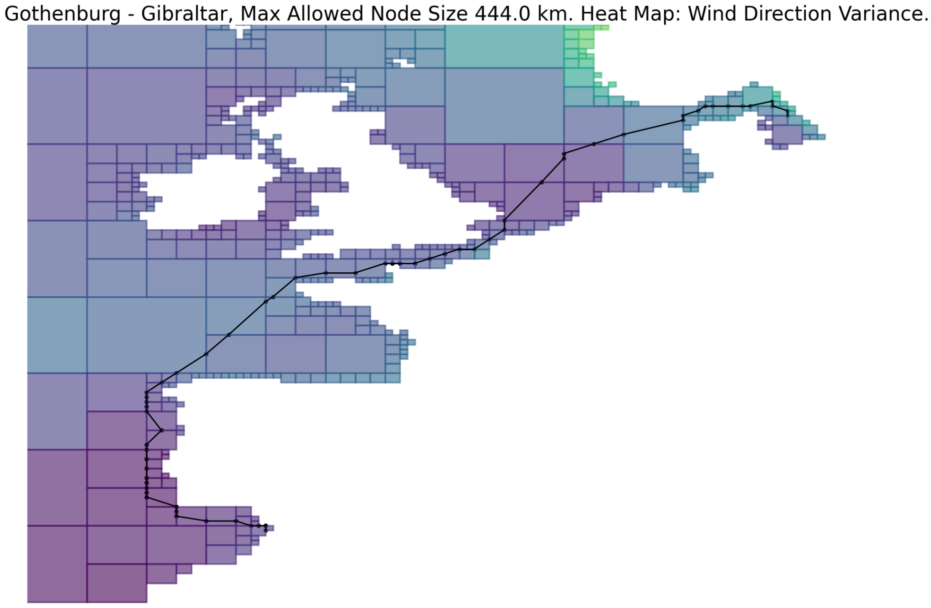


Figure 4.3: Gothenburg - Gibraltar, default speed 15 knots.

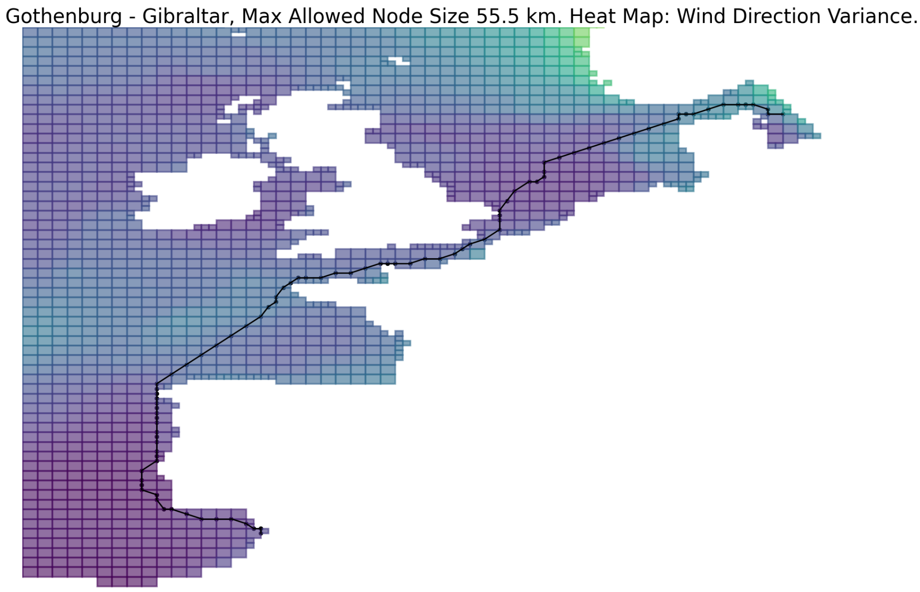


Figure 4.4: Gothenburg - Gibraltar, default speed 15 knots.

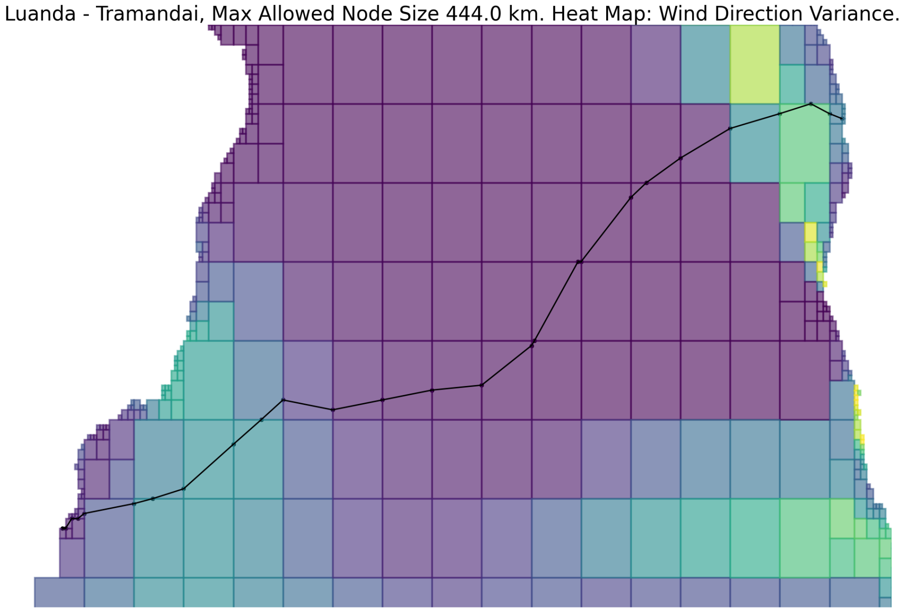


Figure 4.5: Luanda - Tramandai, default speed 15 knots.

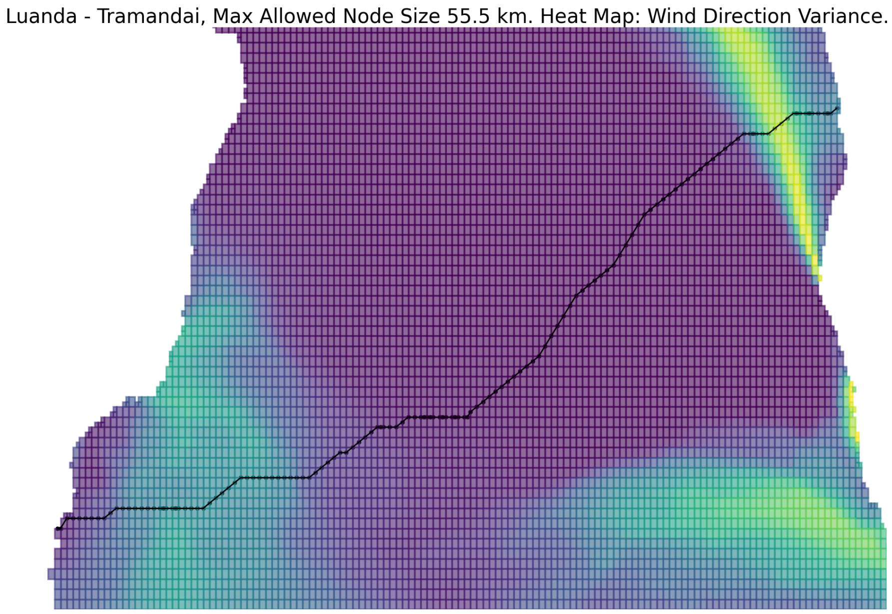


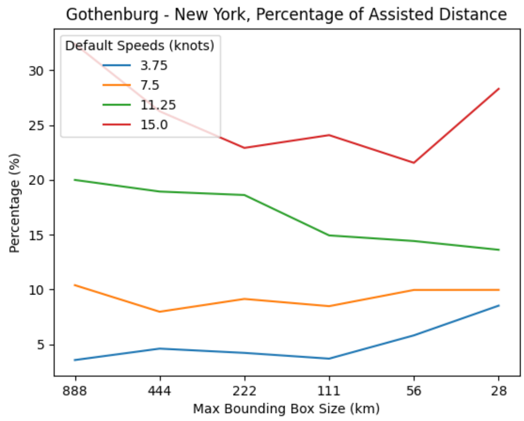
Figure 4.6: Luanda - Tramandai, default speed 15 knots.

4.1 Percentage of Assisted Distance

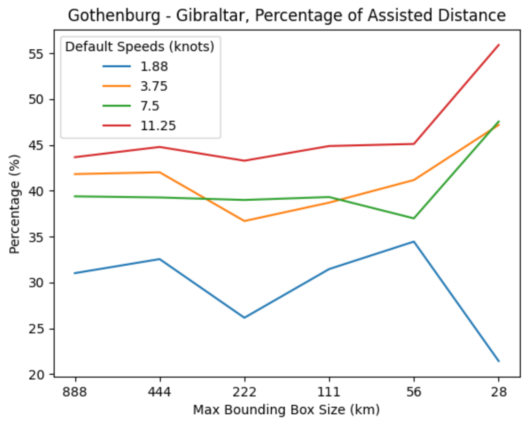
The following plots illustrate the percentage of total path distance covered with engine assistance across three different routes. The paths were planned on forecasted data and then evaluated stepping though the actual historical data.

We used four default speeds to generate four paths for each start and goal coordinate, each a multiple of 15 knots. The 15-knot speed was selected based on CargoKite's estimation that their vessel's maximum calm water speed would be similar.

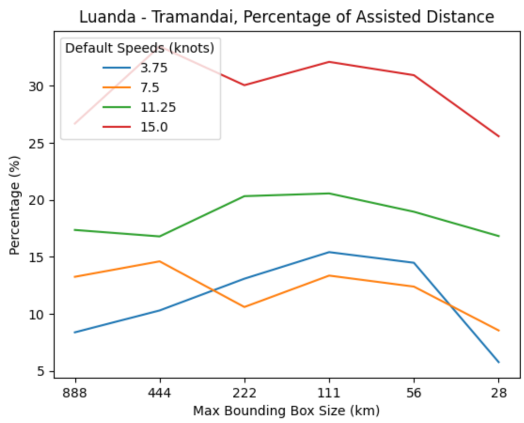
In Figure 4.7b, this speed yielded a high percentage of assisted distance. This could be due to wind speeds being low in the area surrounding the path. Therefore, we tested the multiples .125, .25, .5 and .75 instead. Comparing these shows more siginificant changes between the lines since the path planner tends to choose nodes based on wind data more.



(a) Gothenburg - New York



(b) UK Path



(c) Brazil Path

Figure 4.7: Performance analysis for different paths: (a) GOT - NY path, (b) UK path, (c) Brazil path. Evaluated at different default speeds.

4.2 Expected and Actual Path Duration

The following plots describe how the actual path duration deviates from the expected path duration. The expected path duration is the duration that the algorithm estimates using interpolation, while the actual duration is calculated by stepping through the path as described in section 3.5.

The paths are all planned using the same weather forecast, spanning 0 to 243 hours. The actual recorded data spans 0 to 278 hours, and is used to calculate the actual duration. Both interpolation and forecast innacuracies may cause the expected path duration to be inaccurate.

We observe a consistently higher difference between the expected duration and the actual duration for lower default speeds.

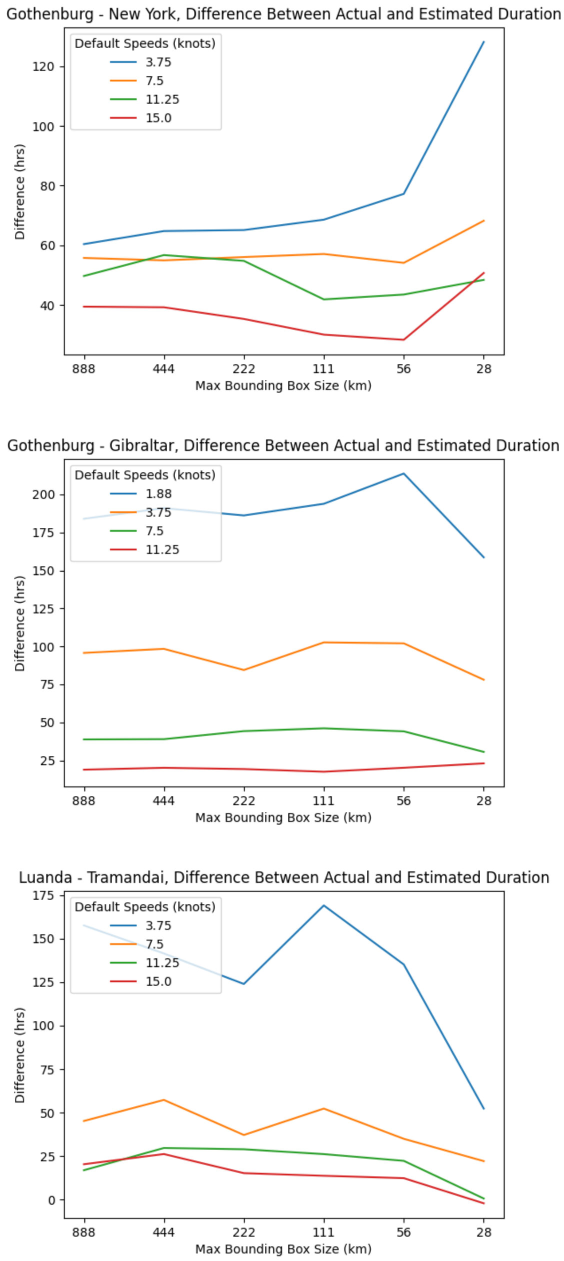


Figure 4.8: Comparison of actual duration and expected duration for different paths: (a) Gothenburg - New York, (b) UK path, (c) Brazil path.

4.3 Memory Consumption and Runtime

This section presents memory consumption and runtime plots for path generation across various levels of quadtree subdivision. Memory usage is reported relative to a baseline consumption of 1.022GB, which is the memory usage of the default quadtree subdivided using border points, including 772.5MB of wind data (Figure 3.1b).

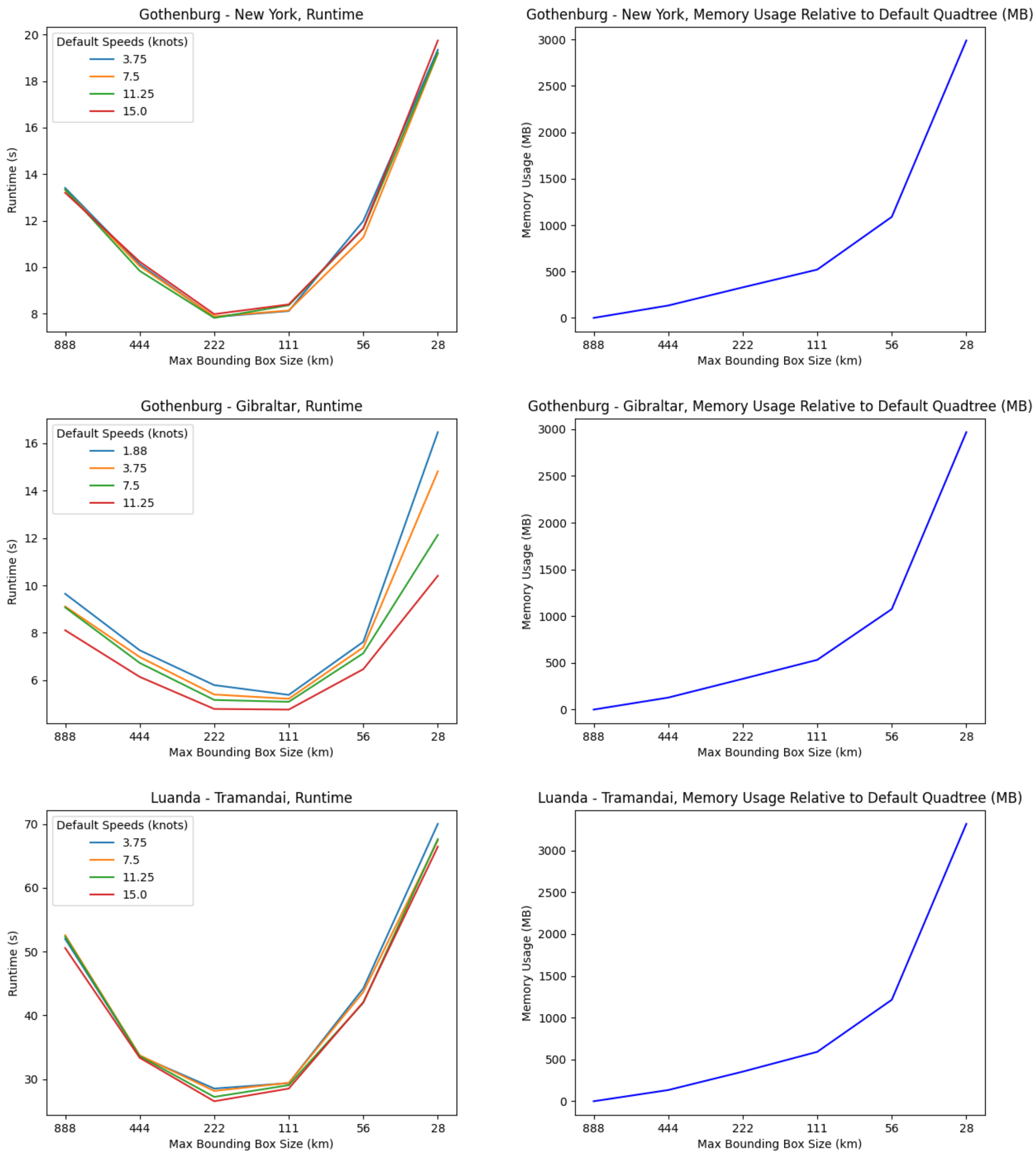


Figure 4.9: Runtime and memory comparison for different max bounding box sizes. The default quadtree subdivided using border points used 1.022 GB of memory, where 772.5 MB was allocated for the wind dataset (figure 3.1b). Note that the quadtree is only subdivided locally around the path, according to an initial partition by Djikstra's Algorithm.

The memory usage of the quadtree representing the entire ocean was recorded under varying maximum bounding box sizes. The measurements were obtained after initializing the quadtree according to Section 3.1, given different maximum allowed bounding box sizes. Each measurement includes the memory allocated by the wind dataset and all frame nodes. Note that bounding boxes share frame nodes, which can result in the same tuple of coordinates appearing up to four times. This happens when a frame node is positioned on a corner of a bounding box.

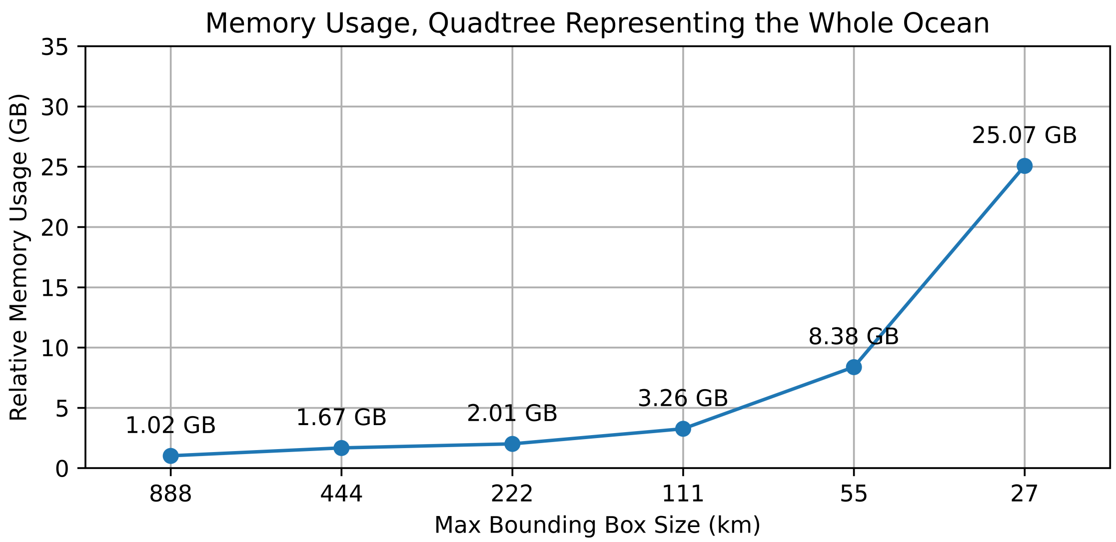


Figure 4.10: Quadtree memory usage for the entire ocean, subdivided to allow different maximum bounding box sizes.

5

Conclusion

This chapter presents the concluding remarks and future research directions stemming from the work presented in this paper. It is divided into the following sections. Sections 5.1-5.2 provide an analysis and interpretation of the obtained results. Section 5.3 suggests interesting approaches and potential avenues for future research. Finally, Section 5.4 provides concluding remarks, summarizing the key insights gained from the preceding discussion.

5.1 Effects of Subdivision

Subdividing the bounding boxes further reveals more wind data that can be utilized by the search algorithm, theoretically enabling interpolation to more accurately estimate travel times. However, it is worth mentioning that the discrepancy between expected path duration and actual path duration seems to rely more on the default speed, as seen in Figures 4.8a - 4.8c.

While the increased resolution reveals more wind data, it also restricts the direction of travel within each bounding box, thereby reducing the range of possible relative boat orientations to the wind. Consequently, this limits the number of relative wind angles that can be utilized. Ideally, we would like to have a continuous range of heading angles for all bounding box sizes, similar to [11], as this increases the algorithm's ability to leverage the wind data effectively.

The number of assisted steps and the difference between the actual and expected path duration do not seem to decrease significantly when we subdivide towards a uniform grid, considering the higher default speeds. In some cases, they are actually lower for bigger bounding box sizes (Figures 4.7a, 4.7b, 4.8a), but that may be due to forecast inaccuracies and random chance rather than the increased heading angle possibilities, requiring further testing to draw stronger conclusions.

Therefore, the interpolation seems to work as long as we assume a high default speed and the surrounding wind isn't too low (as observed in the UK path, Figures 4.7b, 4.8b), where the percentage of assisted steps was high. Additionally, according to runtime results, we do save significant memory and runtime, suggesting that the interpolation approach may be suitable in cases where computational resources are limited, or if one were to handle a big fleet of vessels.

Finally, while higher subdivisions generally improve the accuracy of duration estimation and reduce the variance in wind direction, they also increase the computational overhead and memory requirements of the algorithm. The interpolation algorithm seems to introduce significant runtime overhead for large bounding boxes, but this is reduced once the average bounding box size reduces to about 222-111km (Figures 4.9a - 4.9c). When we approach bounding box sizes close to the highest resolution where the path is planned on a uniform grid, the runtime increases again. We think that it should be possible to vectorize the interpolation operation, reducing the computational overhead observed for the bigger bounding box sizes.

The memory consumption significantly increases when subdividing towards the highest resolution, even if the quadtree is only subdivided locally around the path partition (Figures 4.9d - 4.9f). If one were to plan a similar path using this program on a desktop computer with RAM over 16GB, this would not be a problem, although planning many paths in parallel or longer paths might cause some runtime/memory issues. In that case, running the path planning algorithms with a bigger bounding box size may be a good tradeoff, since it seems to perform similarly in cases with higher default speeds. Based on the data presented in Figure 4.10, significant memory savings could be achieved by increasing the bounding box size to twice its original dimensions when approaching a uniform grid, particularly for larger partitions.

5.2 Default Boat Speed - Handling Assisted Steps

A dilemma arises regarding how to adjust the boat speed during assisted steps. Assisted steps occur when the vessel arrives at a node at a specific time when the wind conditions yield no boat speed or a speed below a certain threshold. There are a couple of situations that can lead to this scenario:

1. **Forecast vs. Actual Data Discrepancy**: If the path is planned using a weather forecast, but the evaluation phase utilizes actual recorded data, discrepancies between the two data sources may arise.
2. **Interpolation Across Lower Resolutions**: When the path calculation is not performed at the highest resolution of the data, and the path between data points is approximated through interpolation, it is possible that a data point at the given time step of traversal during the evaluation phase yields a boat speed below the threshold.

If any of these scenarios occur, it can impact the entire path, as the vessel might arrive later than calculated for subsequent nodes. Consequently, at that later time step, the previously chosen node might no longer be the optimal choice.

To address this issue, a strategy must be implemented to handle assisted steps during the planning phase. The approach employed for the presented results in the paper assumes a static default boat speed when there is no wind provided by an auxiliary electrical engine. This ensures that the vessel maintains a certain speed and always progresses along the path. However, this approach is sensitive to specific wind conditions; if wind speeds are generally low in the area, using the electrical engine is more beneficial. Hence, it is natural to introduce a limit on the distance of the assisted steps, based on the vessel's battery capabilities.

Alternatively, a more dynamic approach could be considered. Since the path is calculated selecting the best nodes at specific time steps, adherence to the sequence of time steps is equally important as the sequence of nodes in the calculated path. Therefore, implementing a dynamic assisted speed that ensures the vessel arrives at the nodes in the path according to the sequence of time steps. We may infer an appropriate boat speed by looking at the mean or variance of boat speeds surrounding the vessel. Using an empirical model to limit the maximum boat speed based on current wind conditions, similar to the approach used in [10] may also improve interpolation accuracy. Currently it is assumed that the boat always reaches its maximum calm water speed regardless of wind conditions.

Furthermore, another method could involve dynamically re-planning the path from the current node, taking into account the updated wind conditions and arrival times. This approach would allow for real-time adjustments to the route, ensuring that the vessel follows the most optimal path based on the latest available information.

5.3 Future Work

This section explores potential future approaches to enhance the work presented in this paper. We begin by detailing algorithms that could be applied on the search space presented in the methodology (Subsection 5.3.1). Following this, suggestions for both admissible and inadmissible domain-specific heuristics are provided in Subsection 5.3.2. These heuristics can guide the search process and discourage unwanted behavior. Finally, Section 5.3.3 examines the possibility of incorporating additional weather parameters.

5.3.1 Applicable Algorithms

Incremental re-planning algorithms have emerged as a method for path planning in scenarios where path optimality can be impacted by dynamic changes in the environment. These algorithms are able to adapt existing paths in response to new data, sidestepping the need for exhaustive recalculations and thereby saving computational resources [25], pg. 10. Notable examples of such algorithms include Focussed Dynamic A* (D*) and D* Lite, widely employed in the domain of robotic path planning [26] [27] [28]. A noteworthy evolution in this field surfaced in 2017 with the introduction of D* Extra Lite, a refined variant of D* that closely mirrors the principles of A* while leveraging search-tree branch cutting techniques to reduce complexity further [29].

The Multi-Resolution Field D* (MRD*) algorithm employs framed quadtrees to represent the environment at varying resolutions based on proximity to the vessel [12]. Areas closer to the vessel are represented at higher resolution, leveraging accurate on-board sensor data and local models. Distant regions use lower resolution due to forecast uncertainty. Adapting resolution based on proximity, enables precise near-term path planning using high-resolution data, while accounting for long-range uncertainty with coarser representations. This aligns with the nature of weather forecasts, which are more accurate for shorter time horizons. This method can be used to create a more sophisticated subdivision process, rather than subdividing the entire partition. Additionally, a metric related to the variance of the data could be used to determine what bounding boxes to further subdivide.

Anytime algorithms are capable of quickly finding feasible sub-optimal solutions within a time window. Examples include Anytime Repairing A* [30] and the Anytime Window A* algorithm by Aine et al. [31]. These algorithms were developed for scenarios where finding an initial solution fast and refining it is preferred, such as in motion planning for robots in a static environment. However, they are not suitable for dynamic environments as they cannot utilize prior search results and bounds on sub-optimality may not hold if graph weights change.

Anytime Dynamic A* (ADA*) is an algorithm that both saves information from previous searches and ensures bounds on sub-optimality hold in a dynamic environment [32]. However, sometimes it can be more computationally expensive to repair the previous solution as opposed to replanning from scratch, particularly in high-dimensional spaces [25].

5.3.2 Heuristics

All of the aforementioned re-planning algorithms share a common feature as they rely on a heuristic function. Measuring the Euclidean distance from a given node to the goal is a trivial A* heuristic. However, determining an appropriate domain-specific heuristic or algorithm parameters for path planning is a nontrivial problem, and in some cases, randomization has been employed to address this [33]. Additionally, designing a single heuristic function to represent a multi-dimensional problem can be demanding. In a 2016 paper by Aine et al., Multi-Heuristic A* (MHA*) is introduced to allow the design of multiple heuristics that are useful in different situations in order to guide the search towards the goal [34].

While Euclidean distance is effective in open environments, this approach becomes less efficient when dealing with maps containing large obstacle regions. In such scenarios, the optimal path often needs to circumvent around obstacles, unless the goal is directly in line-of-sight. To address this limitation, Shah et. al. developed a heuristic that leverages the necessity of passing through corners of obstacles obstructing the straight line path to the goal [9]. This heuristic is particularly useful in complex maritime environments characterized by large landmasses.

Here are some suggested approaches for domain-specific heuristic functions:

- **Maximum Speed Heuristic** (Admissible): By considering the maximum possible speed of the boat, an admissible heuristic can be created by dividing the distance between the current node and the goal by this maximum speed. This ensures that the heuristic does not overestimate the cost to reach the goal.

- **Land Heuristics** (Admissible/Inadmissible): Similar to the approach proposed by Shah et al. [9], these heuristics account for the presence of large land masses or island groups obstructing the straight-line path to the goal. They can help guide the search towards navigating around such obstacles more effectively.
- **Wind Pattern Heuristics** (Inadmissible): Utilize historical wind data to identify areas that are prone to high or low wind conditions on average. Guides the search towards regions that are more likely to have favorable wind conditions, such as seasonal monsoon winds or passage winds.
- **Ocean Current Heuristics** (Inadmissible): Use the direction and speed of ocean currents at the current position, as well as the relative angle towards the goal position to guide towards an area with less impeding currents.
- **Obstacle Heuristics** (Inadmissible): Kambhampati et al. proposed two inadmissible heuristics for quadtrees. These heuristics considers the number of obstacles intersecting the straight-line path between the current node and the goal, as well as the total area of these obstacles.

Combining these heuristics, particularly within a framework like Multi-Heuristic A* (MHA*) [34], could yield more informed search guidance and improved path planning performance. This can be done without sacrificing optimality because of the use of an admissible heuristic in combination with the inadmissible ones.

5.3.3 Introducing More Weather Parameters

Another critical weather parameter influencing a vessel's speed is ocean currents. In contrast to the high variability observed in wind patterns, ocean currents generally exhibit more consistent behavior and lower variability over sizeable ocean territories [35]. This characteristic allows for accurate representation of ocean current data within larger bounding boxes. Integrating ocean current data into the existing system is a straightforward process. By incorporating ocean currents as an additional data layer, the model's accuracy and the quality of the calculated paths will be improved.

5.4 Conclusion

In conclusion, this paper has examined the practical implications of using a framed quadtree for long-distance path planning on the open sea. An interpolation method utilizing the frames of each bounding box was tested and evaluated in the context of wind-powered vessels similar to the one proposed by CargoKite.

The results indicate that the number of assisted steps and the difference between the actual and expected path duration do not significantly decrease with finer subdivisions toward a uniform grid. In some cases, these metrics are even lower for larger bounding box sizes, possibly due to forecast inaccuracies and random chance, suggesting that further testing is required. Therefore, the interpolation method appears to be effective as long as we assume a higher default speed and the wind speed in the local area isn't too low, resulting in a high number of assisted steps.

Given that interpolation heavily depends on a default calm-water speed provided by an auxiliary electric engine, it is crucial to determine an appropriate method for selecting this speed during the path planning process. The choice of default boat speed also impacts accuracy, with higher speeds increasing assisted kilometers and lower speeds potentially overestimating travel times due to interpolation inaccuracies.

Additionally, the results show significant memory and runtime savings using the framed quadtree approach, with the optimal trade-off between memory and runtime occurring at a bounding box size of 222-111km. While the memory savings may not be crucial if one were to plan singular, comparably long paths on a computer with ample RAM, managing a fleet of vessels or planning a path with many intermediate ports could result in substantial runtime and memory savings.

Future research could explore integrating additional weather data, such as ocean currents, and developing a model to adjust the calm-water speed based on current wind conditions. The framed quadtree approach could be even more suitable for ocean currents, given that they can be assumed to be more homogeneous than wind patterns. To improve runtime and reduce inaccuracies caused by forecast deviations, ideas from Multi-Resolution Field D* [12] could be integrated, using higher resolutions near the vessel to leverage accurate local data while accounting for long-range forecast uncertainty with coarser representations farther away. To minimize errors because of forecast deviations it is beneficial to re-plan, building upon the previously calculated path as as soon as a new forecast is obtained.

In summary, the framed quadtree approach shows promise for long-distance autonomous maritime navigation, particularly for wind-powered vessels. Future work should focus on refining default speed selection, incorporating additional weather data, and further optimizing the quadtree framework to improve efficiency and robustness in real-world maritime applications.

Bibliography

[1] J. J. Corbett and H. W. Koehler, "Updated emissions from ocean shipping," *Journal of Geophysical Research: Atmospheres*, vol. 108, no. D20, 2003.

[2] H. Jungen, P. Specht, J. Ovens, and B. Lemper, "The rise of ultra large container vessels: Implications for seaport systems and environmental considerations," in *Dynamics in Logistics: Twenty-Five Years of Interdisciplinary Logistics Research in Bremen, Germany*, pp. 249–275, Springer International Publishing Cham, 2021.

[3] T. Chou, V. Kosmas, M. Acciaro, and K. Renken, "A comeback of wind power in shipping: An economic and operational review on the wind-assisted ship propulsion technology," *Sustainability*, vol. 13, no. 4, p. 1880, 2021.

[4] F. Lian, J. Jin, and Z. Yang, "Optimal container ship size: a global cost minimization approach," *Maritime Policy & Management*, vol. 46, no. 7, pp. 802–817, 2019.

[5] CargoKite Corporation, "Cargokite," Year of the content. Accessed: 11/26, 2023.

[6] J. Langbein, R. Stelzer, and T. Frühwirth, "A rule-based approach to long-term routing for autonomous sailboats," in *Robotic Sailing* (A. Schlaefer and O. Blaurock, eds.), (Berlin, Heidelberg), pp. 195–204, Springer Berlin Heidelberg, 2011.

[7] G. Mannarini, G. Coppini, P. Oddo, and P. Nadia, "A prototype of ship routing decision support system for an operational oceanographic service," *TransNav*, vol. 7, pp. 53–59, 03 2013.

[8] G. Mannarini, R. Lecci, and G. Coppini, "Introducing sailboats into ship routing system visir," in *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–6, 2015.

[9] B. C. Shah and S. K. Gupta, "Long-distance path planning for unmanned surface vehicles in complex marine environment," *IEEE Journal of Oceanic Engineering*, vol. 45, no. 3, pp. 813–830, 2019.

[10] W. Lee, G.-H. Choi, and T.-w. Kim, "Visibility graph-based path-planning algorithm with quadtree representation," *Applied Ocean Research*, vol. 117, p. 102887, 2021.

[11] D. Ferguson and A. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-19*, 2005.

[12] D. I. Ferguson and A. Stentz, "Multi-resolution field d.," in *IAS*, pp. 65–74, 2006.

[13] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory.," in *SODA*, vol. 5, pp. 156–165, 2005.

[14] D. Delling and D. Wagner, "Landmark-based routing in dynamic graphs," in *Experimental Algorithms: 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007. Proceedings 6*, pp. 52–65, Springer, 2007.

[15] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *International Conference on Extending Database Technology*, 2008.

[16] G. Nannicini, D. Delling, L. Liberti, and D. Schultes, "Bidirectional a search for time-dependent fast paths," in *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*, pp. 334–346, Springer, 2008.

[17] L. Zhao, T. Ohshima, and H. Nagamochi, "A* algorithm for the time-dependent shortest path problem," in *WAAC08: The 11th Japan-Korea Joint Workshop on Algorithms and Computation*, vol. 10, p. 40, 2008.

[18] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, pp. 1–9, 1974.

[21] H. Samet and R. E. Webber, "Storing a collection of polygons using quadtrees," *ACM Trans. Graph.*, vol. 4, p. 182222, jul 1985.

[22] D. Chen, R. Szczerba, and J. Uhran, "Planning conditional shortest paths through an unknown environment: a framed-quadtree approach," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 3, pp. 33–38 vol.3, 1995.

[23] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt, "Framed-quadtree path planning for mobile robots operating in sparse environments," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 1, pp. 650–655, IEEE, 1998.

[24] "National oceanic and atmospheric administration." `https://www.noaa.gov/`. Accessed: February 13, 2024.

[25] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, p. 17, 2005.

[26] S. Koenig and M. Likhachev, "D* lite," in *Eighteenth national conference on Artificial intelligence*, pp. 476–483, 2002.

[27] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, pp. 127–145, 1995.

[28] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Proceedings 2002 IEEE international conference on robotics and automation (Cat. No. 02CH37292)*, vol. 3, pp. 3016–3023, IEEE, 2002.

[29] M. Przybylski and B. Putz, "D* extra lite: a dynamic a* with search-tree cutting and frontier-gap repairing," *International Journal of Applied Mathematics and Computer Science*, vol. 27, no. 2, 2017.

[30] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems* (S. Thrun, L. Saul, and B. Schölkopf, eds.), vol. 16, MIT Press, 2003.

[31] S. Aine, P. Chakrabarti, and R. Kumar, "Awa-a window constrained anytime heuristic search algorithm," in *International Joint Conference on Artificial Intelligence*, pp. 2250–2255, Morgan Kaufmann Publishers Inc., 2007.

[32] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm.," in *ICAPS*, vol. 5, pp. 262–271, 2005.

[33] A. Bhatia, J. Svegliato, and S. Zilberstein, "On the benefits of randomly adjusting anytime weighted A," in *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021* (H. Ma and I. Serina, eds.), pp. 116–120, AAAI Press, 2021.

[34] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 224–243, 2016.

[35] Y. Ashkenazy and H. Gildor, "On the probability and spatial distribution of ocean surface currents," *Journal of Physical Oceanography*, vol. 41, no. 12, pp. 2295–2306, 2011.