A Rule-based Approach to Long-term Routing for Autonomous Sailboats

Johannes Langbein, Roland Stelzer, and Thom Frühwirth

Abstract We present an algorithm for long-term routing of autonomous sailboats with an application to the ASV Roboat. It is based on the A*-algorithm and incorporates changing weather conditions by dynamically adapting the underlying routing graph. We implemented our algorithm in the declarative rule-based programing language Constraint Handling Rules (CHR) [4]. A comparison with existing commercial applications yields considerably shorter computation times for our implementation. It works with real-life wind forecasts, takes individual parameters of the sailboat into account, and provides a graphical user interface.

1 Introduction

Autonomous sailboats perform the complex maneuvers of sailing fully automatically and without human assistance. Starting off by calculating the best route based on weather data and going on to independent tacking and jibing, autonomous sailboats are able to sail through to any destination. Humans merely have to enter the destination coordinates.

The approach described here is planned to be implemented in the control system of the ASV Roboat, an autonomous sailing boat which has been in development by

Johannes Langbein
Faculty of Engineering and Computer Science, Ulm University, Germany e-mail: firstname.lastname@uni-ulm.de

Roland Stelzer
INNOC - Austrian Society for Innovative Computer Sciences, Vienna, Austria e-mail: firstname.lastname@innoc.at

Thom Frühwirth
Faculty of Engineering and Computer Science, Ulm University, Germany e-mail: firstname.lastname@uni-ulm.de

a research team of the Austrian Society for Innovative Computer Sciences (INNOC) since 2006.

So far, weather routing on the ASV Roboat relies on locally measured weather data only. This is proven to be suitable for short distances, respectively short durations, such as regattas over a few miles [12]. In contrast, for long-term missions like ocean crossings, weather conditions cannot be assumed to remain stable until the boat reaches its target. Therefore, a global view with consideration of weather forecasts is necessary.

In this paper, we introduce a long-term weather routing algorithm for autonomous sailboats and show how rule-based programming facilitates a declarative and efficient implementation. In Section 2 we present how we modeled the sailboat routing problem and introduce our routing algorithm. Its implementation is then discussed in Section 3. In Section 3.1, we compare our algorithm to existing commercial solutions and discuss related work. We conclude in Section 4, which also gives an outlook on future work.

2 The routing algorithm

Routing for sailboats, no matter whether they are autonomous or not, can be defined as the *"procedure, where an optimum track is determined for a particular vessel on a particular run, based on expected weather, sea state and ocean currents"* [11]. In this section, we will take a closer look at the parameters required to find an optimum track and present our routing algorithm.

2.1 Modeling long-term sailboat routes

For this work, we distinguish between long-term and short-term routing in the following way: Long-term routing is the task of finding a sequence of waypoints $\mathbf{x}_0 \dots \mathbf{x}_n$ (longitude and latitude coordinates) for a given starting point $\mathbf{x}_{start}$ and time t_{start} and a given destination point $\mathbf{x}_{dest}$, where $\mathbf{x}_{start} = \mathbf{x}_0$, $\mathbf{x}_{dest} = \mathbf{x}_n$ and $\mathbf{x}_k$ is reachable from $\mathbf{x}_{k-1}$ at sea while taking global weather forecasts into account. Short-term routing, in contrary, is the task of finding suitable boat headings to reach the next waypoint, given the current local weather conditions [12].

Several definitions for the term "optimum track" in the quote above are possible (see [12]), yet we want to focus on minimizing the arrival time t_{dest} at the point $\mathbf{x}_{dest}$. In order to calculate the arrival time t_k at any waypoint $\mathbf{x}_k$, we need to take weather data as well as the individual behavior of the sailboat into account.

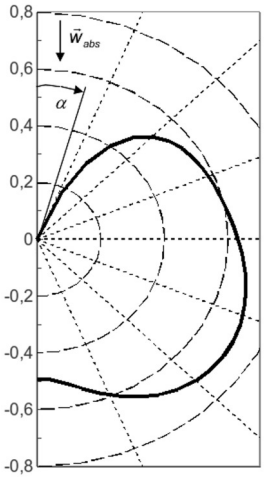

Fig. 1 The normalized polar diagram of the ASV Roboat

2.1.1 Weather data

As long-term routing is typically concerned with distances taking several days or weeks to travel, weather forecasts are required to calculate an optimal route. Weather forecasts are usually made available in the form of GRIB (Gridded Binary) files, a standardized format to store weather data [14]. In a GRIB file, wind conditions are represented as a grid of wind vectors $\mathbf{w}$, containing the wind-speed in north and east direction. A GRIB file can contain multiple forecasts, which are made available for up to 16 days in intervals as small as three hours. The resolution of the wind data typically ranges between 0.5 and 2.5 degrees.

2.1.2 Sailboat behavior

To calculate the time required to travel between two waypoints, we need to know the speed of the sailboat for given wind conditions. This speed can be described as a function of the wind-speed and the angle between the wind and the boats heading, that is to say, the boats velocity $v = \mathrm{v}(\mathbf{w}, \alpha)$, if α denotes the true wind angle of the boat. This function is usually shown in a plot known as *polar diagram*. Figure 1 shows the normalized polar diagram of the ASV Roboat [12], which describes the relation between wind-speed and boat-speed for a given true wind angle. Another factor to take into consideration is the so-called *hull-speed*, which we treat as the approximative maximum speed of the boat. In our algorithm, this maximum speed is configurable by the user.

We approximate the travel time t_{ij} between two locations $\mathbf{x}_i$ and $\mathbf{x}_j$ on a great circle path by taking the wind conditions $\mathbf{w}_i$ and true wind angle α_i at $\mathbf{x}_i$ for the first

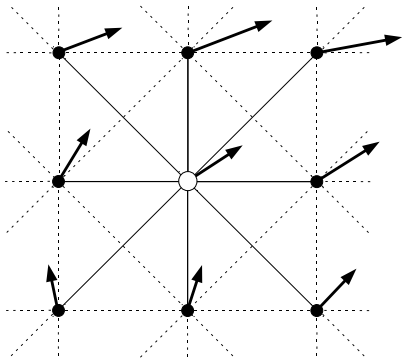

Fig. 2 An exemplary part of the routing graph. Wind vectors for each node are shown as bold arrows, edges to and from the center node as solid lines. Edges connecting the surrounding nodes are denoted as dashed lines.

half of the leg and the wind conditions $\mathbf{w}_j$ and true wind angle α_j at $\mathbf{x}_j$ for the second half. The distance d_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j$ is calculated using the laws of spherical geometry [2]. Together, we get the travel time $t_{ij} = \frac{1}{2} \cdot d_{ij} \cdot (v(\mathbf{w}_i, \alpha_i) + v(\mathbf{w}_j, \alpha_j))$. As sailing directly upwind is not possible, sometimes it is required to beat in order to sail from $\mathbf{x}_i$ to $\mathbf{x}_j$. We incorporate this into the travel time calculation by approximating the boats velocity made good along the great circle path between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the following way: As described in [7], we neglect the time for tacking and compute the velocity made good by using the convex hull of the polar diagram for speed calculations when sailing upwind. The true wind angle at which the boat is required to beat can be configured by the user.

2.1.3 Routing graph

Oceans constitute a continuous search space with an infinite number of possible waypoints. To reduce the search space and make classical shortest-path methods applicable to the routing problem, we chose to discretize the search space into a grid graph with equidistant nodes, representing points on the sea. Each node is connected to its eight nearest neighbors by directed edges. Nodes located on a land mass, which are detected using a binarized world map, are not included in the graph. Nodes at locations with hazardous wind conditions, that is to say, locations, at which the wind-speed is above a user-configurable limit, are omitted as well. Figure 2 illustrates a portion of such a routing graph.

Each node in the graph is annotated with wind vectors, shown as bold arrows in the figure. In most cases, we have multiple forecasts at hand, therefore each node is annotated with one wind vector per forecast. As the location of a node usually does not coincide with a grid point in the GRIB file, bi-linear interpolation is used to calculate the wind vectors for each node. The directed edges of the graph are annotated

with the travel time, calculated as shown in Section 2.1.2. Again, when multiple forecasts are present, each edge is annotated with one travel time per forecast.

The described discretization of the search space means, that the optimal route in the grid model is calculated, which is only an approximation to the true optimal route. Thus, we made the quality of the approximation configurable to the user in that the distance of the nodes in the routing graph can be arbitrarily chosen.

2.2 Calculating the optimal route

We chose the A*-algorithm as basis for our long-term routing as it allows the use of a heuristics for performance gain [5]. Much like Dijkstra's algorithm, the A*-algorithm assigns each node $\mathbf{x}_i$ a cost-value $\mathrm{c}(\mathbf{x}_i)$, which is the time required to travel to this node from the starting point. The algorithm retains an *open list* from which the currently best node is chosen for expansion. In contrary to Dijkstra's algorithm, the best node is not the node having the lowest cost but the node having the lowest value $\mathrm{c}(\mathbf{x}_i) + \mathrm{h}(\mathbf{x}_i)$, where $\mathrm{h}(\mathbf{x}_i)$ is the value of the heuristics for the node $\mathbf{x}_i$. This heuristics gives an estimate for the cost to reach the node $\mathbf{x}_{dest}$ from $\mathbf{x}_i$. We use the distance from $\mathbf{x}_i$ to $\mathbf{x}_{dest}$ divided by the boats hull-speed as heuristic function $\mathrm{h}(\mathbf{x}_i)$. It is *consistent* as it satisfies the triangle inequality $\mathrm{h}(\mathbf{x}_i) \leq t_{ij} + \mathrm{h}(\mathbf{x}_j)$ for two adjacent nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ and also the condition $\mathrm{h}(\mathbf{x}_{dest}) = 0$. We choose this heuristics as it is cheap to compute and its consistency guarantees the optimality of the A*-algorithm [5].

To find an optimal route, the A*-algorithm is run on the routing graph described in Section 2.1.3. As the number of nodes and edges in the routing graph can get very high, we tried to optimize the graph construction to save memory and computation time in the following way:

Some nodes in the open list do not need to be expanded, if the destination node is reached before they are considered. Also, the fact that our heuristics is consistent guarantees that a node will never be expanded more than once by the A*-algorithm [5]. This allows to dynamically construct and deconstruct the routing graph during the execution of the algorithm:

- A node $\mathbf{x}_j$ is not created until one of the eight direct neighbors $\mathbf{x}_i$ in the grid is chosen for expansion.
- Edges are not created until two adjacent nodes are present.
- If a node is added to the closed list, every incoming and outgoing edge to and from this node can be removed from the graph, as nodes in the closed list will not be considered again.

The dynamic construction of the graph also facilitates another optimization: When a node $\mathbf{x}_i$ is expanded, the time $t_{start} + \mathrm{c}(\mathbf{x}_i)$, at which this node will be reached by the boat, is known. Consequently, the forecast in the GRIB file valid at this point in time is known and the calculation of the travel time for outgoing edges of $\mathbf{x}_i$ has to be done only for this forecast and not for all forecasts in the GRIB file.

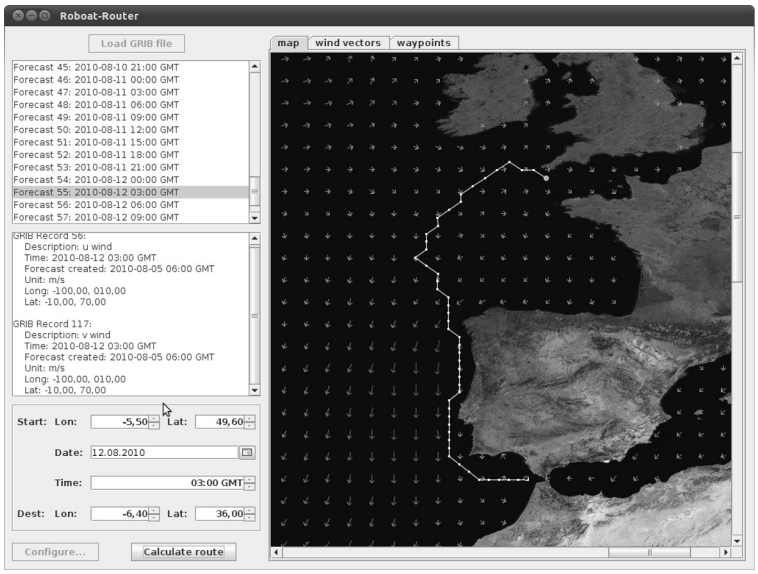

Fig. 3 The GUI showing wind conditions and the calculated route

3 Implementation in CHR

We implemented our algorithm mainly in Constraint Handling Rules (CHR), combined with SWI-Prolog. For an introduction to CHR, we refer to [4]. Our implementation is based on an existing implementation of Dijkstra's algorithm with a Fibonacci heap in CHR [10], which is used as open list to achieve optimal time complexity and was extended to incorporated the use of the heuristic function. Our implementation uses CHR rules for the following tasks:

- Expanding a node: Every time a node is extracted from the open list, a rule is triggering the creation of the neighboring nodes, if they are not yet present.
- Creating edges: If there are two neighboring nodes, a rule creates an edge between the two nodes with the according travel time.
- Labeling neighbor nodes: As soon as edges are present, a rule labels the neighbors of the node expanded last with the according cost and inserts them into the open list.
- Adding node to closed list: If there are no more neighbors to be labeled, a node is added to the closed list.
- Removing edges: A node in the closed list triggers a rule removing all the incoming and outgoing edges of this node.
- Path reconstruction: Once the goal node is reached, a CHR rule reconstructs the shortest path found.

We chose CHR for its declarativity, which allowed us to implement the routing algorithm in a compact and clear fashion: The implementation in CHR consists of

only 17 rules with a little under 1000 lines of code. In addition, CHR allows the routing graph to be constructed and deconstructed dynamically by simply stating the conditions, under which nodes and edges are created or removed, as rules. We furthermore believe, that the implementation in CHR facilitates future adaption and extensions in an easy way, which will help to incorporate changes that might be necessary on the ground of evaluations to come in real-life settings and conditions.

Our implementation can be used from the Prolog command line or via a Java application providing a graphical user interface. The GUI visualizes the wind conditions and allows to define starting and destination points on a world map. It also provides configuration options for the routing and displays the calculated route on the map. Figure 3 shows the GUI application.

3.1 Evaluation and Related Work

There are various commercial applications for long-term weather routing of sailboats like the *BonVoyage System*, *MaxSea*, *Sailplanner* [9], or *SailFast* [8]. The latter two are available as demo version, thus we picked them to compare our routing algorithm to. Both applications offer a GUI similar to ours. While Sailplanner automatically downloads its own wind data and is restricted to one provider (WeatherTech), SailFast allows the usage of arbitrary GRIB files. Sailplanner offers the user to pick from five different resolutions for the routing graph it uses, while SailFast uses an isochron method with an unknown resolution but with configurable time steps. However, they are both closed source applications, not revealing details about the algorithm they use for the routing.

We picked two routes of equal distance between starting point and destination (about 1440 km on a great circle path), one along the east-coast of the U.S. and one in open water to compare the running time of the two commercial solutions to our algorithm. The tests were carried out on a 2.0 GHz Intel Dual Core with 4 GB of RAM while no other applications where running and the wall clock time taken for the computation was measured. The output of Sailplanner indicates a grid width of about 30 km and 20 km for the resolutions “Medium High” and “Ultra High”, respectively. Hence, we chose 30 km and 20 km as the grid resolution for the runs of our algorithm. SailFast is fixed to a 6 hour resolution for the isochron lines in the demo version. The results of the comparison are given in Table 1.

The results show, that our algorithm is considerably faster when calculating routes. A reason for this the could be the use of a heuristic function in our algorithm or the fact, that SailFast and Sailplanner seem to consider points on land in the routing as well, while our algorithm avoids them. However, the most likely reason is the fact, that more than eight different bearings for each point are considered by SailFast and Sailplanner in contrary to our algorithm. The results in Table 1 also indicate the expected trade-off between computation time and quality of the approximation to the optimal route, stemming from the complexity of the A*-algorithm which is

Table 1 Comparison of wall clock computation time required for routing in seconds

	SailFast 6 hours	Sailplanner "med. high"	Sailplanner "ultra high"	Roboat router 20 km	Roboat router 30 km
Route 1	109	40	254	10	6
Route 2	79	37	224	9	5

exponential in the number of way points in the solution and thus the resolution of the grid [5].

The routes computed by our algorithm and SailFast are almost identical, while the route computed by Sailplanner is somewhat different. There are two reasons this difference might stem from: Firstly, Sailplanner uses a different polar diagram, which could not be changed in the demo version available to us. Secondly, Sailplanner was run with wind data from WeatherTech, as it does not allow the import of GRIB files. Our algorithm and SailFast were both run with the same GRIB file from saildocs.com since the data from WeatherTech is not freely available.

In academic research, several methods have been proposed for sailboat routing (see [12] for an overview). A stochastic method for long-term routing based on dynamic programming was presented by Philpott and Allsopp [7], while methods from operations research were used by Papadakis and Perakis [6]. Recent work by the AVALON team [3] uses a routing algorithm similar to ours, however, they do not include weather forecasts in their calculations. To our knowledge, none of the aforementioned approaches have been implemented in a rule-based language like our algorithm and there has not been a published implementation of a long-term weather routing algorithm for sailboats in a declarative language.

A popular algorithm for path planning in continuous search spaces is the Theta*-algorithm [1] which also works on a grid of nodes. It allows for any-angle path planning, creating edges to all nodes in sight of the node currently expanded. An implementation of this algorithm would provide for a more accurate routing as more angles are considered. However, this would require many more nodes to be held in memory, lead to a higher calculation time, and would raise difficulties at choosing the appropriate forecasts for calculating the travel time of an edge. Hence, we chose the A*-algorithm with dynamic construction of the routing graph over Theta*.

The D*-algorithm [13] and its variants are designed for searching in dynamically changing graphs. Their advantage is the fast replanning when costs of edges are modified during the execution of the path, which is particularly interesting if a robot continuously collects new information about its environment along the route. In robotic sailing however, costs are only updated when new forecasts are available, which usually happens only every couple of hours. Furthermore, the D*-algorithm reconsiders nodes in the closed list, which would infer with the dynamic construction and deconstruction of the our routing graph (cf. Section 2.2). For those reasons, we opted against the D*-algorithm in favor of a much simpler implementation and less memory consumption, while accepting the necessity of a full re-routing once new forecasts are available.

4 Conclusion and Future Work

We proposed a long-term routing algorithm which finds an arbitrarily accurate approximation to the optimal route for a sailboat for real-life wind conditions. Our approach takes changing weather conditions into account by dynamically adapting the underlying graph used as input to the A*-algorithm. It is is implemented in Constraint Handling Rules and compares very well to existing solutions. To our knowledge, it is the first published implementation of an long-term weather routing algorithm for sailboats in a declarative or rule-based programming language.

4.1 Future Work

We use deterministic weather forecasts which are only available for a certain time ahead (see Section 2.1.1). So-called *ensemble forecasts* consist of different scenarios that can happen with given probabilities and are usually available for a longer time ahead. Research for sailing yacht races has shown how ensemble forecasts can be used to find optimal routes which perform well under all possible scenarios [7]. An extension of the routing algorithm to handle ensemble forecasts could make the routing more realistic and would allow for accurate planning of even longer routes.

Besides wind conditions, the travel time of the sailboat can be affected by currents as well. Incorporating ocean current data into the calculation of the travel time could also make the routing more accurate and may lead to better routes.

Using a graph where nodes are connected only to their eight direct neighbors puts a considerable restriction on possible paths and can lead to noticeable differences between projected and actual arrival time. This restriction could be lowered, while improving the quality of the calculated route, by using a graph where nodes are connected to 24 neighbors, as presented in [3]. We believe that this could be achieved with a reasonable increase in calculation time by replacing the CHR rules for node and edge generation.

References

1. Daniel, K., Nash, A., Koenig, S., Felner, A.: Theta*: Any-Angle Path Planning on Grids. Journal of Artificial Intelligence Research **39**, 533–579 (2010)
2. Donnay, J.D.H.: Spherical Trigonometry. Interscience Publishers (2007)
3. Erckens, H., Büsser, G.A., Pradalier, C., Siegwart, R.Y.: Avalon: Navigation Strategy and Trajectory Following Controller for an Autonomous Sailing Vessel. IEEE Robotics & Automation magazine **17**(1), 45–54 (2010)
4. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009)
5. Hart, P., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100 – 107 (1968)
6. Papadakis, N.A., Perakis, A.N.: Deterministic Minimal Time Vessel Routing. Operations Research **38**(3), 426–438 (1990)

7. Philpott, A., Mason, A.: Optimising Yacht Routes under Uncertainty. In: Proceedings of the 15th Chesapeake Sailing Yacht Symposium (CSYS 2000) (2000)
8. SailFast LLC: SailFast Version 5.1. http://www.sailfastllc.com/ (2011)
9. Sailport AB: Sailplanner. http://sailplanner.net/ (2011)
10. Sneyers, J., Schrijvers, T., Demoen, B.: Dijkstra's Algorithm with Fibonacci Heaps: An Executable Description in CHR. In: Proceedings of the 20th Workshop on Logic Programming (WLP 2006) (2006)
11. Spaans, J.A.: Windship routeing. Journal of Wind Engineering and Industrial Aerodynamics **19**, 215 – 250 (1985)
12. Stelzer, R., Pröll, T.: Autonomous Sailboat Navigation for Short Course Racing. Robotics and Autonomous Systems **56**(7), 604 – 614 (2008)
13. Stentz, A.: Optimal and efficient path planning for unknown and dynamic environments. International Journal of Robotics and Automation **10**(3), 89–100 (1993)
14. A Guide to the Code Form FM 92-IX Ext. GRIB Edition 1. http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/GRIB1-Contents.html (1994). Accessed 11/28/2010