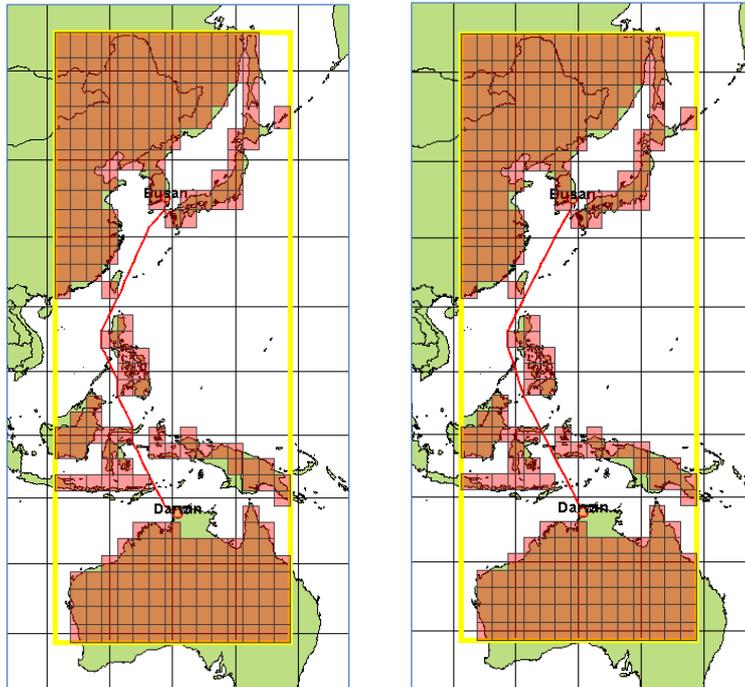


# DESIGN OF A WEATHER DEPENDENT RE-ROUTING ALGORITHM FOR SAFETRANS MONTE CARLO SIMULATIONS



Universiteit Utrecht



**Faculty of Science, Utrecht University**  
Department of Information and Computing Sciences  
*Utrecht, the Netherlands*

**Maritime Research Institute Netherlands (MARIN)**  
*Wageningen, the Netherlands*

April, 2015  
Roel Verwey

**To Marlous,**

**For all the support you gave during those years of hard study**



# TABLE OF CONTENTS

NOMENCLATURE .....	5
ABSTRACT .....	6
1. INTRODUCTION .....	7
1.1. Background.....	7
1.2. Main objective.....	8
1.3. Formulation of the weather re-routing problem .....	9
1.4. Scientific contribution.....	10
1.5. Report structure .....	11
2. RELATED WORK.....	12
2.1. Weather routing for ships .....	12
2.1.1. Calculus of variations (CoV) .....	13
2.1.2. Dynamic programming (DP) .....	14
2.1.3. (Modified) Isochrone method (MIM) .....	16
2.1.4. Isopone method (IP) .....	19
2.1.5. (Multi objective) Evolutionary Algorithms .....	20
2.2. Path planning in grid-based environments .....	23
2.2.1. A* algorithm .....	24
2.2.2. Any-angle pathfinding in grid-based problems .....	27
2.2.3. Optimal any-angle algorithm .....	28
3. SAFETRANS MONTE CARLO SIMULATION.....	31
3.1. A brief history .....	31
3.2. SafeTrans design methods .....	31
3.3. Initializing a MCS-case .....	32
3.4. Description of the MCS-mode .....	33
3.5. Metocean weather database.....	35
3.6. Decision making process.....	35
3.7. Re-route algorithm in SafeTrans .....	38
3.8. Example case .....	41
4. DESIGN ALTERNATIVE RE-ROUTE ALGORITHM .....	43
4.1. Preliminaries.....	43
4.2. 3D any-angle path finding .....	44
4.3. Extracting a path.....	48
4.4. Path cost.....	51
4.5. Correctness and optimality.....	52

4.6. Time complexity .....	52
5. EXPERIMENTS.....	55
5.1. Experimental setup .....	55
5.1.1. Description of the simulation.....	56
5.1.2. Calculation of vessel speed .....	58
5.1.3. Calculation of fuel consumption .....	60
5.1.4. Dijkstra and Anya 3D.....	60
5.1.5. Rapidly-Exploring Random Tree: RRT*.....	60
5.2. Problem instants.....	62
5.3. Route generation without weather .....	63
5.4. Route generation with weather forecast .....	69
5.5. Route performance in weather scenarios .....	71
6. CONCLUSIONS & RECOMMENDATIONS .....	80
6.1. Conclusions.....	80
6.2. Recommendations .....	81
REFERENCES.....	82

## NOMENCLATURE

CDM	Captain's Decision Mimic
C	Path cost function
CoV	Calculus of Variations
D	Decision taken by CDM
DP	Dynamic Programming
DV	Decision variable defined in VOM
ETA	Estimated time of arrival
F	Force in [kN]
Hs	Significant wave height
HPA*	Hierarchical Pathfinding A*
IMDSS	Integrated Marine Decision Support System, i.e. weather forecast database
I	Interval in Anya 3D
IP	Isopone Method
JIP	Joint Industry Project
JPS	Jump Point Search
MCR	Maximum Continuous Rating, i.e. max power output of engine
MCS	Monte Carlo Simulation
MIM	Modified Isochrone Method
MPP	Maximum Principle of Pontryagin
PRA*	Partial Refinement A*
R	Resistance in [kN]
RAO	Response Amplitude Operator
RRT	Rapidly-exploring Random Tree
RRT*	Rapidly-exploring Random Tree Star
SDA	Significant Double Amplitude
T	Tuple in Anya 3D
Tp	Peak period of wave spectrum
TRS	Tropical Revolving Storm
VOM	Voyage Operating Module
VMC	Voyage Motion Climate
U	Ship's control vector, expressed by heading and MCR
X	Ship's position vector, expressed by latitude and longitude

## ABSTRACT

SafeTrans is a software program used to simulate intended marine heavy lift transports with the purpose to obtain statistics on the shipping route, sailing speed and motion behavior. In SafeTrans a Monte Carlo simulation (MCS) can be performed in which a specific transport is simulated in multiple runs subjected to randomly selected historical weather forecast scenarios applicable to the sailing area. Some routes advised by the actual implemented *Dijkstra* re-route algorithm do not reflect realistic decisions. Experience with Safetrans re-routing has shown several problems and limitations of this algorithm design, such as land-crossings, route generation at fixed speed, optimization for sailing time only and lack of route decision memory.

In this study *Anya 3D* is proposed, an improved re-route algorithm for SafeTrans MCS that aims at finding the optimal sailing route for a given weather forecast while meeting user-defined weather criteria and admissible ship engine settings. This route is based on a cost function that includes user-defined weights for fuel consumption, sailing time and late arrival penalty.

The basis for this algorithm is *Anya*, a recently published sketch for an optimal any-angle pathfinding algorithm for 2D environments with grid-based obstacles. This sketched algorithm has been extended for path finding problems in a 3D space and time environment with sea grid cells either blocked or traversable in time depending on the weather conditions. In addition an algorithm is proposed to make a path taut in 3D based on the multi-objective cost function and economical speed.

In simulations, the proposed algorithm is tested and compared to the actual re-route algorithm for route distance, sailing time, fuel consumption and runtime. In addition an implementation is made of the sample-based *RRT\** pathfinding algorithm in order to compare both *Dijkstra* and *Anya 3D*. Experiments have been performed for eight problem instants that contain severe weather conditions, distributed over four shipping routes. The experiments show better route generation with respect to the research objectives for *Anya 3D* in comparison to *Dijkstra* when the amount of obstacles is limited. In addition, runtimes of search queries with limited obstacles is shorter than the actual implemented algorithm. For more complex environments the performance of *Anya 3D* drops as the search space grows exponentially with the number of obstacles. Methods to prune branches of the search tree should be further investigated to avoid exponential growth.

# 1. INTRODUCTION

## 1.1. Background

SafeTrans is a simulation software tool to design and operate marine heavy lift and wet tow transports and offshore installations based on ship hydrodynamics, loading condition, sailing route and seasonal weather influences. The tool has been developed in a Joint Industry Project coordinated by the hydrodynamic research institute MARIN in the Netherlands and is used in the industry for over 10 years by a user group consisting of shipping companies, oil majors, designers and warranty surveyors.

One of the options in SafeTrans is to make a discrete event Monte Carlo simulation (MCS) in which a specific transport can be simulated in multiple runs, each subjected to a randomly selected historical weather forecast scenario within a user-defined departure time frame of interest (e.g. departure date in January). These weather forecasts are presented to a Captain's Decision Mimic (CDM) that makes 3-hour navigational decisions in order to meet the user-defined transport criteria. Such criteria could for example be specific motion restrictions to cargo or maximum towline force for a wet tow. One of the decisions that this CDM can take every 3 hours is the decision to re-route, i.e. to define a new route when the present route will cross severe weather that might exceed the transport criteria.

The focus of this research study is on the re-route algorithm within the CDM. This algorithm was designed in the late 1990s, but experience has shown some shortcomings in the algorithm.

Unfortunately not all routes advised by the re-route algorithm are realistic or representative for decisions made by real captains. Currently re-routing is done with the Dijkstra-algorithm. At each time step a 7 days weather forecast is given for each 2.5 x 2.5 degrees grid block at sea. This results in time-dependent forbidden grid blocks in which criteria are exceeded. Given these time-dependent forbidden sailing blocks, the present ship location and the port of destination a 3D-shortest path Dijkstra is used to determine the shortest sailing route in time and location. After this route is defined a route stretching algorithm is applied to smooth the route and to avoid (is)land crossings. Experience with SafeTrans re-routing has shown the following problems and limitations of this algorithm design:

### **1. Land crossing**

Besides Dijkstra grid route generation the algorithm uses an underlying extensive graph that contains logical sailing connections between fixed geographical locations to make its final route advice. In some occasions this results in land crossing, e.g. when the ship finds its nearest graph vertex at a location separated by a landmass.

## **2. Fixed engine settings**

Sometimes the best policy to avoid bad weather areas is to change route and ship speed simultaneously. This is not possible in the present CDM since these decisions cannot be made at the same time. Therefore re-routing is always performed with fixed engine settings and ship speed is only depending on environmental conditions inside each grid block. In reality a captain will select its route by anticipating ship speed and heading on advancing weather patterns.

## **3. Optimization criteria**

Shortest sailing time is the only objective that the present algorithm uses while re-routing. With the actual level of fuel prices it could well be that the ship owner that carries out a heavy transport is more interested in saving fuel rather than arrival time. It could also be of interest to combine sailing time, fuel consumption and ship motions in a user-specified objective function.

## **4. Decision memory**

Currently no memory is included of earlier made decisions. This could result in a sudden drastic and unrealistic course change in Northern direction when a Southern route was initially chosen for ocean passage whereas in reality a captain would continue its route based on an initial rough voyage plan.

## **1.2. Main objective**

SafeTrans is used for the design of future transports by simulating voyages as detailed and as realistic as possible. Therefore the aim is to design a re-route algorithm that returns a route that could realistically be taken by a real captain that is being presented the same set of information. Since in reality 5 different captains would sail 5 different routes, this practically means that the aim of the algorithm is to find the optimal sailing route for each situation. This goal can be defended by the trend of using navigational decision support tools onboard ships. Such systems become more and more standard policy in onboard decision-making and also present the optimal route to a captain.

The main objective of the research study is defined as follows:

*Design of a re-route algorithm that finds the optimal sailing route for a given weather forecast and a set of user-defined transport criteria based on a cost function that includes fuel consumption, sailing time and ship motions.*

This optimal sailing route is subject to:

- Avoidance of land-crossings
- Reasonable algorithm running time compared to the present re-route algorithm

In order to increase the probability of implementation into the SafeTrans software it is advised to base the design of the algorithm on the present algorithm setup and connections to the other modules in the software.

### 1.3. Formulation of the weather re-routing problem

In SafeTrans all weather information is presented geographically in a 2.5 degrees longitude by 2.5 degrees latitude grid and time-wise in blocks of 12 hours with a weather forecast to 168 hours. Based on the user-defined voyage criteria (i.e. maximum sea state) each cell in the grid is either blocked or unblocked in time for periods of 12 hours with a forecast to 168 hours. The sea state is expressed as the significant wave height in some sector at some period in time and the applicable wave criterion is the maximum allowable wave height that should not be exceeded. In addition to time-dependent blocks, all grid cells that are dominantly covered by land are blocked as well.

Finding the optimum re-route can be reduced to finding the ship route with lowest cost function from the actual vessel position to the destination in a continuous search space with grid-based weather and land obstacle information. In such an optimal path all turning points of the route are corner points (Harabor, 2013). Therefore finding a re-route is a discrete planning problem with the objective to find a feasible sequence of states in the state space to get from the start state to the goal state (LaValle, 2006). In this context a state basically represents the vessel position  $x$  at some time  $t$ .

In path finding this problem is classified as an any-angle pathfinding problem. The continuous search space is the set of all sailable grid cells. This space can be traversed with any vessel heading and the ship's position while traversing grid cells is not constrained to the grid points. In this problem the weather scenario is fully known and does not develop over time.

For the definition of the problem first the ship's control vector should be defined:

$$U = [\theta, MCR] = \text{Ship's control vector}$$

where,

$\theta$	= heading of the ship
$MCR$	= engine settings, i.e. Power output as percentage of the engine's Maximum Continuous Rating

The ship's position vector is defined as follows:

$$X = [lat, lon] = \text{Ship's position vector}$$

where,

$lat$	= latitude position of the ship
$lon$	= longitude position of the ship

These two vectors are subject to the following constraints:

$$U \in U_{admissible}$$

$$X \in X_{admissible}$$

where,

$U_{admissible}$  = Admissible control vector, i.e. range of allowable engine settings

$X_{admissible}$  = Admissible position vector, i.e. navigable sailing area

The objective is to find the sailing route from the actual vessel position to the destination that has minimum costs for the defined objective function. The cost of a potential sailing route is the sum of the weighed parameters for sailing time and fuel consumption. A penalty can be added for late arrival.

$$C = \int_{T_0}^T (C_{Fixed} + C_{Fuel}) dt + \int_T^{T_s} C_{Destination} dt$$

where,

$C_{Fixed}$  = Fixed ship cost per unit time (crew, provisions, insurance, depreciation, overhead)

$C_{Fuel}(MCR(t))$  = Fuel cost per unit time (function of engine setting)

$C_{Destination}$  = Delay penalty per unit time (late arrival)

$T_0$  = Departure time from origin

$T$  = Actual arrival time at destination

$T_s$  = Scheduled arrival time at destination

## 1.4. Scientific contribution

Actual weather routing ship navigation systems often use methods that are single objective and designed for one specific task (Hinnenthal, 2008). In onboard implementations of navigation software the search space is typically discretized with loss of optimality of the solution in order to keep acceptable runtime performance. More advanced methods such as evolutionary algorithms are applied as well, but no application of such methods in commercial software is known. This could be due to the extensive runtime that such methods require.

These shortcomings in ship navigation algorithms are tackled in this study. The problem is approached from a different angle by studying the research field of path planning. This field is extremely vivid in robotics, simulation and the gaming industry where often collision-free paths should be found in limited time

for environments with sometimes many dynamic obstacles. In this study the research fields of ship navigation and path planning are linked to solve our problem optimally and fast.

The basis for the weather re-route algorithm that is designed for our problem is a recently published sketch for an optimal any-angle pathfinding algorithm for environments with grid-based obstacles (Harabor, 2013). Harabor presents the concept of the approach for 2D environments with grid cells that are either blocked or traversable. The algorithm is only sketched and no implementation and experiments have been published. In this study an implementation of this concept is made and tested for a set of problem instants.

An important contribution to Harabor's approach is the extension for the time-dimension. By making the search space 3D a modified definition of the Harabor's search intervals is required. It should be noted that this time dimension differs from the 2 geographical dimensions since obviously travelling back in time is not allowed. Therefore a corridor is made in the search space based on admissible ship control settings.

Extracting paths in 2D is simply done by following parent pointers until the start node is reached. In the 3D version this requires a different approach, since the path should be made taut over a set of line segments instead of points. Therefore an algorithm is proposed to extract 3D paths.

Another extension is made for the path cost definition. In Harabor's approach a grid cell is only evaluated for the path distance, but in our algorithm the cost of traversing a grid cell depends on the balance between the objectives arrival time, fuel minimization and delay penalty that is specified by the user.

## 1.5. Report structure

This report is organized as follows. In chapter 1 an introduction to the problem is presented and the research objective is defined. Chapter 2 discusses related work with respect to weather routing and pathfinding in grid-based environments. Chapter 3 describes SafeTrans and how re-routing is actually performed in a Monte Carlo simulation. In chapter 4 *Anya 3D* is described in detail. The experiments to compare performance of *Anya 3D* to *Dijkstra* and *RRT\** are presented in Chapter 5.

## 2. RELATED WORK

### 2.1. Weather routing for ships

Historically the scientific approach to ship weather routing goes back to the 18<sup>th</sup> century when Benjamin Franklin's North Atlantic Gulf Stream mapping helped to decrease the average transit time between North America and England. In the 19<sup>th</sup> century large amounts of atmospheric and oceanographic data sets were collected to further improve ocean weather routing (NIMA, 2002).

Since the 1950's various numerical methods were developed for route optimization based on weather forecast. Research in the first decades could be classified in the following categories:

- Calculus of variations (Haltiner, 1962; de Wit, 1968; Bijlsma 1975)
- (Modified) isochrone method (James, 1957; Hagiwara, 1989)
- Dynamic programming (Zappoli, 1972; Chen, 1978)

Calculus of variations is a mathematical approach to weather routing and treats it as a continuous optimization problem. The (modified) isochrone method is a more practical approach based on visualizing time fronts on a sea map. Dynamic programming is a recursive algorithm to find the minimum cost route in a discretized grid (e.g. fixed space-time).

The developed methods were mainly implemented in shore based weather route stations to give advice to navigators on a distance. But since the 1990's also onboard systems came into use as improvements in computer power, communication systems and accuracy of weather forecast had enabled this development. The methods had mainly focused on optimizing transit time. But these methods were extended such that optimization on fuel consumption was possible. For example, Klompstra's isopone method calculated fuel fronts instead of time fronts (Klompstra, 1992) to analyze the attainable sailing region based on fixed amount of fuel.

In the past decade evolutionary algorithms came up as optimization method for weather routing. Traditional methods had mainly focused on optimization for one objective such as shortest voyage time or lowest fuel consumption. But since 2004 the bunker prices have increased drastically and have become a considerable expense to shipping lines. For ship owners, this induced different and case-by-case selection criteria in ship routing.

In addition, safety and environmental issues have started playing a major role in the shipping industry, fed by international discussions that have evolved from major disasters with high environmental impact such as the BP Deepwater Horizon oil spill in the Gulf of Mexico in 2010.

As a result weather routing researchers started to develop multi-objective methods that could cope with opposing targets that typically occur in shipping.

For example, finding the shortest sailing time opposes to the minimizing fuel and maximizing safety and comfort. In recent years several research studies have been performed based on multiple objectives. They mainly use evolutionary algorithms in their approaches (Hinnenthal, 2008; Marie, 2009; Szłapczyńska, 2007).

Since the 2000's the use of onboard weather routing systems has become more and more standard practice for sea-going vessels. For example, onboard systems were developed specifically for emergency situations with restricted ship control due to engine blackout or steering problems (Böttner, 2007).

In the following sections the mentioned methods will be explained in more detail.

### **2.1.1. Calculus of variations (CoV)**

Calculus of variations is a mathematical approach to the problem of optimizing ship routing. CoV treats ship routing as a continuous optimal control problem and deals with maximizing or minimizing some defined functional. A functional is a mapping function that assigns a real number to a set of functions belonging to some class. In the case of optimizing ship routing the functional is the objective function and this should be minimized for a set of control functions in order to obtain optimal navigation. Traditionally the objective is to minimize travel time (Haltiner, 1962), but this could also be aimed at minimizing fuel consumption (Bijlsma, 2006).

The optimal control functions (e.g. ship's heading, number of propeller revolutions, propeller pitch settings) can be found by improving an initial feasible route according to the gradients of the objective function until an optimum is reached for the given boundary conditions (e.g. departure/arrival location, allowable ship motions, maximum sailing time). These gradients are the partial derivatives of the functional with respect to the geographical position and the ship's controls. This is a set of linear differential equations that are often mutually dependent in the case of ship routing. As a consequence this approach often results in convergence problems and the solution procedure generally breaks down (Chen, 1978).

Since the minimum of the gradient is analyzed, the approach can result in local minima of the objective function. Therefore the method has to be applied to various initial routes (i.e. settings for heading and number of propeller revolutions) in order to find the global minimum of the objective function.

The method of CoV has been applied to the ship routing problem since the 1960's, but has mainly remained at a theoretical level. Convergence problems, inaccuracies that may arise at higher order differentials and risk of finding local optimum solution routes are the main objections against this approach (Klompstra, 1992). Although there is no practical implementation of the method in operation, the approach is still being further developed and might be promising in the long perspective (Bijlsma, 2006; Abramowski, 2006).

### 2.1.2. Dynamic programming (DP)

In the approach of dynamic programming both space and time are discretized. This enables to break down the problem of finding an optimum ship route into a set of simpler sub-problems. A solution to the total minimum cost route can be found by recursively solving the sub-problems according to Bellman's principle of optimality. This principle basically holds that if a certain control policy (or sequence of space-time-states) is optimal for a given problem, then after removing the initial states the remaining decisions still constitute an optimal policy. Repeating this principle from destination back to departure returns the optimum navigation policy.

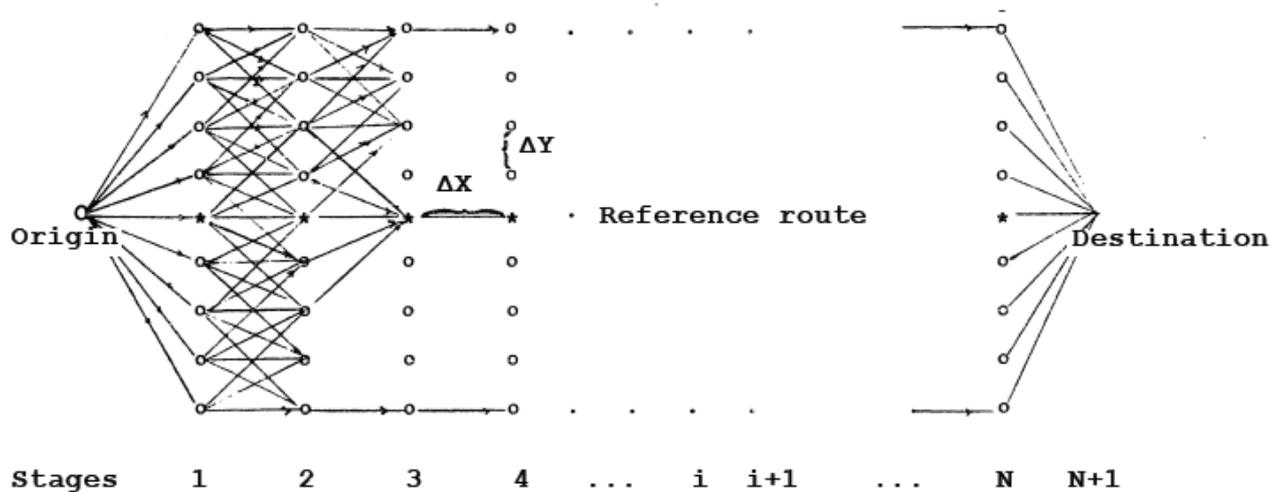
In dynamic programming, a discrete optimization problem is written in a recursive form by expressing each single space-time-state in terms of its connected previous states. For each state, given the cost to reach the connected previous states and given the travel costs between those states and the current state, the lowest cost to reach this current state can be calculated. This concept is applied step-by-step in a recursive manner from destination until the initial states have been reached (the smallest sub-problems) and the lowest cost route can be calculated. Costs are expressed in terms of the defined objective function, which can include operational costs per time-unit, fuel costs, port costs, delay penalties etc.

There are two conditions that must hold for dynamic programming. First, there should be *optimal substructure*. This means that when the shortest route passes state  $S$ , then this route constitutes of the combination of shortest routes from departure to  $S$  and from  $S$  to destination. Second, there should be *overlapping subproblems*. This means that the recursive algorithm should solve the same subproblems over and over again. Such a subproblem could for example be the shortest route from some specific state to the next stage.

Chen has described his implementation of the dynamic programming approach in an extensive PHD-study. He defined the ship routing problem as a multi-stage decision problem of finding the minimum cost route in a predefined grid of coordinates and a discretized time frame (Chen, 1978).

There are several ways to define such a grid. Chen used an East-West Trans-Atlantic route with grid points at constant latitude and longitude increments. More generally this could be done by making a great circle route between departure and destination node or hard point on the route ( $X$ -axis) and divide this line into  $N$  equally spaced legs at distance  $\Delta X$ . At each obtained node, a set of nodes is defined perpendicular to the great circle route, also with equal intermediate distance ( $\Delta Y$ ). Figure 2-1 depicts a schematic version of this grid and the allowable transitions between states from stage to stage.

Also time has to be discretized. For each location, Chen defined a fixed amount of arrival times at equal interval determined by the minimum and maximum sailing speed. The voyage progress in  $X$ -direction is the stage of the algorithm. For each combination of location and arrival time a (space-time) state is formed.



**Figure 2-1.** Ship routing as a multi-stage decision problem using a predefined grid system (Chen, 1978).

Now starting at the departure stage 1, stage by stage the lowest cost route is determined for each space-time-state. When the destination stage  $N$  has been reached, the lowest cost route can be reconstructed instantly.

An important aspect of weather routing is the stochastic variation in ocean environmental forecasting. Chen included a forecast probability distribution in his approach for each state transition. But he used the expected arrival time at each node and neglected any variation in sailing time. There have been other approaches as well. Allsopp for example modeled weather uncertainty by a discretized branching scenario tree in which weather unfolds to an underlying tree (Allsopp, 2000). Each node in the tree is a weather scenario with a specified probability. Basically he extended the deterministic DP-approach with weather scenarios as a state variable. He developed his method for professional long distance yacht racing.

Recently Samsung Heavy Industries has developed an onboard decision support routing system based on the concept of DP (Park, 2008). In this tool, the advised route plan is optimized for sailing time, fuel consumption and motion responses. The route does not only depend on the forecasted weather conditions but also on the performance of the ship's propulsion system. Although the developed system links many aspects of ship performance (e.g. also hull stresses are monitored), it assumes no error in weather forecast.

The accuracy of the solution by dynamic programming depends largely on the fineness of the grid system used for the computation. Therefore it needs many grid points for the search routine in order to obtain an accurate solution, and subsequently it uses a lot of calculation time and memory space (Hagiwara, 1989).

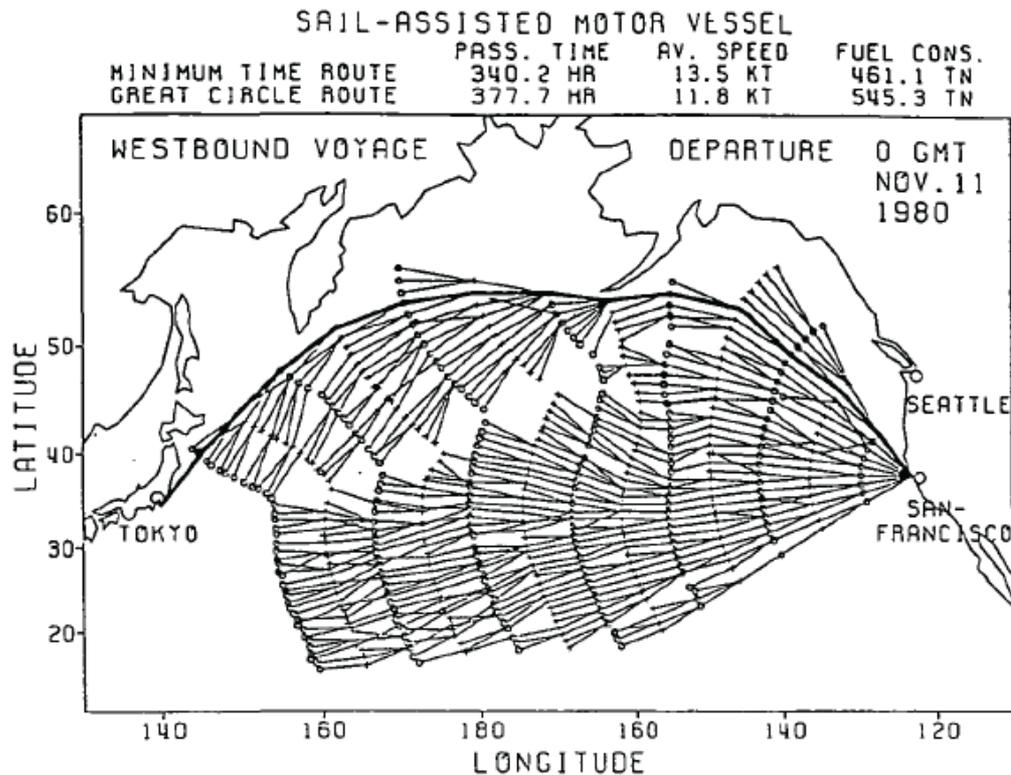
A drawback in DP is that only forward direction is allowed. Avoiding severe weather conditions can only be realized by speed reduction and course

deviation. Although this might be acceptable as a general assumption, there are exceptions in real life where major deviations are necessary and smaller forward steps are required (for example when a hurricane crosses the sailing route).

### **2.1.3. (Modified) Isochrone method (MIM)**

The Isochrone method was originally proposed by R.W. James in 1957 as a practical hand method for navigators to determine minimum time routes (James, 1957). James expressed the ship speed as an empirical function of wave height and direction and determined successive time fronts for a variety of directions based on weather forecasts. Such a time front is called an isochrone and basically is the outer boundary of the attainable sailing region within a specified time frame. So given a specific time step and keeping all ship parameters fixed, the isochrone shows the maximum distance that can be sailed based on the applicable weather forecast. The shape of the isochrone indicates the attainable ship speed in each direction depending on the environmental conditions. An example of an isochrone map is presented in Figure 2-2.

Theoretically each successive isochrone is obtained by sailing in the direction normal to the actual isochrone. The track with the highest projected speed on the isochrone's normal is the advised sailing route (Maximum Principle of Pontryagin, MPP). This original approach was less suitable for computer applications as it resulted in isochrone-loops in case of non-convex isochrones (Szłapczyńska, 2008). Therefore various modifications have been proposed to the original method of James. For example Hagiwara presented the Modified Isochrone Method that was a discretized version of the (continuous) isochrone method (Hagiwara, 1989). Although discretization implies lower accuracy, his approach was more suitable for computational use. The method assumes fixed number of propeller revolutions and fixed propeller pitch along the entire shipping route.



**Figure 2-2.** Example isochrone plots for Pacific voyage San Francisco – Tokyo.

Given a start coordinate  $X_0$ , a set of courses with a small fixed intermediate heading  $\Delta C$  is defined ( $C_0 \pm i\Delta C$ ). These courses are mirrored around the great circle route (heading  $C_0$ ) from start coordinate to destination (or next hard point on the sailing route). Then each course is navigated for a fixed time interval  $\Delta T$  (e.g. 6 hours) along the rhumbline, meaning constant course with respect to all meridians. The applicable environmental conditions determine the sailed distance after this time interval. Note that the ship's heading and course should not necessarily be equal due to ocean current and drift by wind. Therefore the ship's heading should be determined carefully to keep the intended rhumbline course. The set of reached coordinates after  $\Delta T$  determines the first isochrone.

The next iteration is more complex, since the number of routes increases rapidly. For each coordinate of the first isochrone a similar set of courses is defined and sailed for a period  $\Delta T$  with updated weather forecast. From this set of points, the second isochrone should be determined based on MPP. This is a complicated iterative mathematical problem, since according to Pontryagin's principle the normals on the presently unknown 2<sup>nd</sup> isochrone should be calculated to determine this 2<sup>nd</sup> isochrone.

Hagiwara introduced a more practical approach to determine isochrones. He defined sub-sectors based on great circle lines through the departure point  $X_0$ . These lines are centered around the great circle from departure to hard point / destination and are taken such that neighboring lines have equal intermediate distance at the actual isochrone. This idea is visualized in Figure 2-3. Then for each sub-sector (area bordered by two great circle lines and the actual

isochrone) the coordinate is selected with maximum distance from departure coordinate. For practical use also the coordinate can be selected with minimum distance to destination  $x_f$ . This coordinate is determined for each sub-section. The set of points of all sub-sections defines the second isochrone. This process is repeated until an isochrone is defined that approaches the destination  $x_f$  sufficiently close. The final route can be found by going backward from destination to start coordinate  $X_0$  along the route legs between the isochrones.

In addition to time-optimization Hagiwara's method can also be used to optimize for fuel or total costs. An objective function  $J$  is defined that can include cost functions for fixed costs (e.g. crew, insurance, depreciation), fuel consumption or other costs (e.g. delay penalty, port costs). Hagiwara proposed a loop to run the algorithm for various settings of propeller revolutions  $n$  in order to find the ship control settings with minimum cost  $J^*(n^*)$ . By running this loop a curve for total costs  $J(n)$  will be obtained as function of  $n$ . For lower settings of  $n$  (low speed) the penalty costs for late arrival will be determinant, while for higher settings of  $n$  the fuel costs will lead to high values for  $J$ . Basically for time-optimization this loop is only run once for maximum (or economical) number of revolutions.

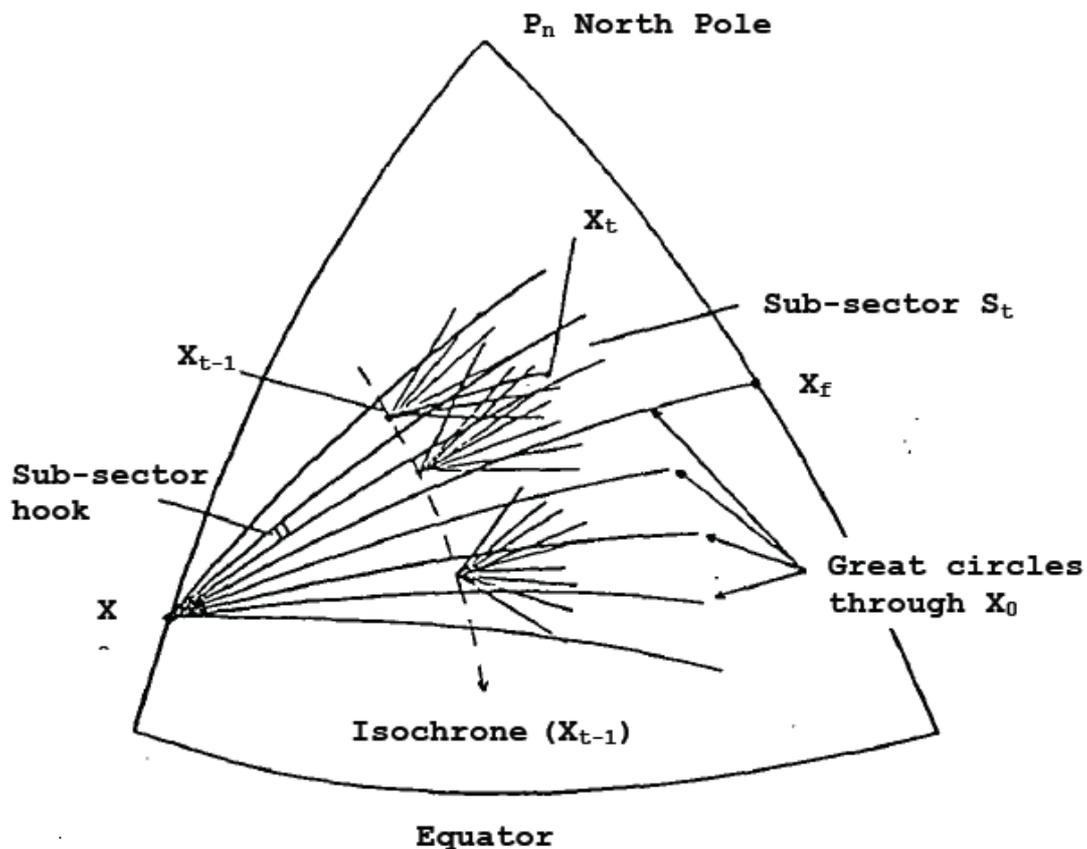


Figure 2-3. Modified Isochrone Method: determine next isochrone.

Errors in weather forecasts are included in the method and are regarded as Gaussian distributed. Including this forecast error enables to estimate the standard deviation of passage time, vessel speed and fuel consumption. The objective function was extended to include these standard deviations appropriately.

It should be noted that since Hagiwara keeps the number of propeller revolutions constant over the entire voyage a sub-optimal solution is found, which is the main drawback of his approach.

Recent developments to MIM have been performed by Szłapczyńska (Szłapczyńska, 2008). Her research mainly focused on avoiding land-crossings, for which she developed an algorithm that analyses a high-resolution bitmap of the sailing area.

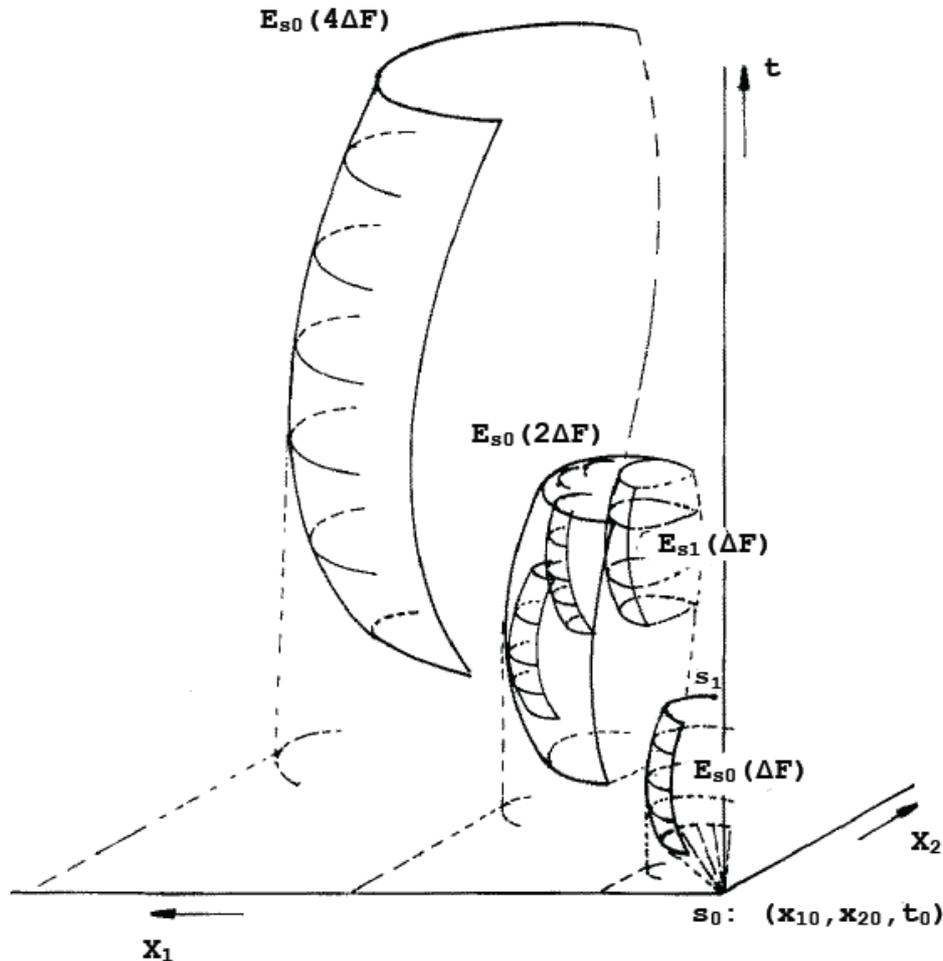
#### 2.1.4. Isopone method (IP)

The isopone method is based on MIM, but instead of calculating time fronts for time steps  $\Delta T$  the method repeatedly computes energy fronts for fuel consumption steps  $\Delta F$  (Klompstra, 1992). In order to do so a time dimension is added to the problem. Whereas MIM uses only the vessel position as state, IP uses both vessel position as well as time for its state. The energy front or isopone is defined as the outer boundary of the attainable sailing region from the departure location after consuming a fixed amount of fuel.

Basically the algorithm is comparable to MIM. The first isopone is determined from the start state  $s_0: (x_{10}, x_{20}, t_0)$ .  $x_{10}$  means longitude at time  $t_0$  and  $x_{20}$  means latitude at time  $t_0$ . Then the next energy front is calculated for a set of  $j$  discretized time steps  $j\Delta T$  bounded by  $T_{min}$  and  $T_{max}$ . For each time step the vessel speed is determined based on the speed-fuel relation and the actual environmental conditions. Similar to MIM the algorithm loops over a set of discretized courses.

The above approach is applied to all points of the first isopone and the envelop of this set of points is the next isopone. This rapidly explodes the number of space-time states and therefore this boundary is discretized in a similar way to MIM. To do this, a set of flat and curved planes is defined perpendicular to the  $x_1$ - $x_2$ -plane such that all planes are crossing start  $x_0$  and destination  $x_f$  and split the current isopone in sufficiently small and equally spread sub-sections. For each sub-section the space-time state that has the largest Euclidean distance to  $x_0$ . This approach results in a 3 dimensional state space as depicted in Figure 2-4.

This process is repeated until the destination is reached, i.e. an energy front is constructed with a projection on the  $x_1$ - $x_2$ -plane that either contains or lies beyond the coordinate of destination.



**Figure 2-4.** *Isopones in a three-dimensional state space (Klompstra, 1922).*

The benefit of this approach over MIM is that variations of sailing speed can be applied in route selection. This provides more flexibility and ultimately results in a better solution, albeit at significantly higher computational costs

### 2.1.5. (Multi objective) Evolutionary Algorithms

In the past decade evolutionary algorithms came up as optimization method for weather routing. These algorithms were mainly developed to find the most suitable route when considering multiple objectives. The approach of evolutionary computation is used in many other path-finding fields as well, such as robotics and aviation. In weather routing examples can be found in (Hinnenthal, 2008; Marie, 2009; Szłapczyńska, 2007).

In evolutionary algorithms the mechanisms of biological evolution are used to find the optimum solution for a given problem. As a starting point a *population* of solutions to a problem is generated and evaluated. This is the first generation in

the algorithm and consists of individuals (single solutions) that are feasible, but not yet optimal. Then, by applying the equivalent biological concepts of *mutation* and *crossover*, a set of new individuals is generated. This group of new individuals, the offspring, will be evaluated by means of a so-called fitness function that incorporates the defined objectives. A subset of this group will then be selected to form the next generation. This process is repeated until the increase in fitness meets some termination condition.

In the problem of weather routing, an individual is a single feasible route from departure to destination. Such a route has a *chromosome structure*, consisting of route points. Each route point consists of *genes*, variables such as location (latitude/longitude) and engine settings. In each generation the new offspring is generated for the best-fit individuals. A crossover is a recombination of two partial solutions that share a common route point and arrival time. A mutation is a random variation to a part of a voyage. Mutation diversifies the genetic pool and improves the chances of finding the optimal solution.

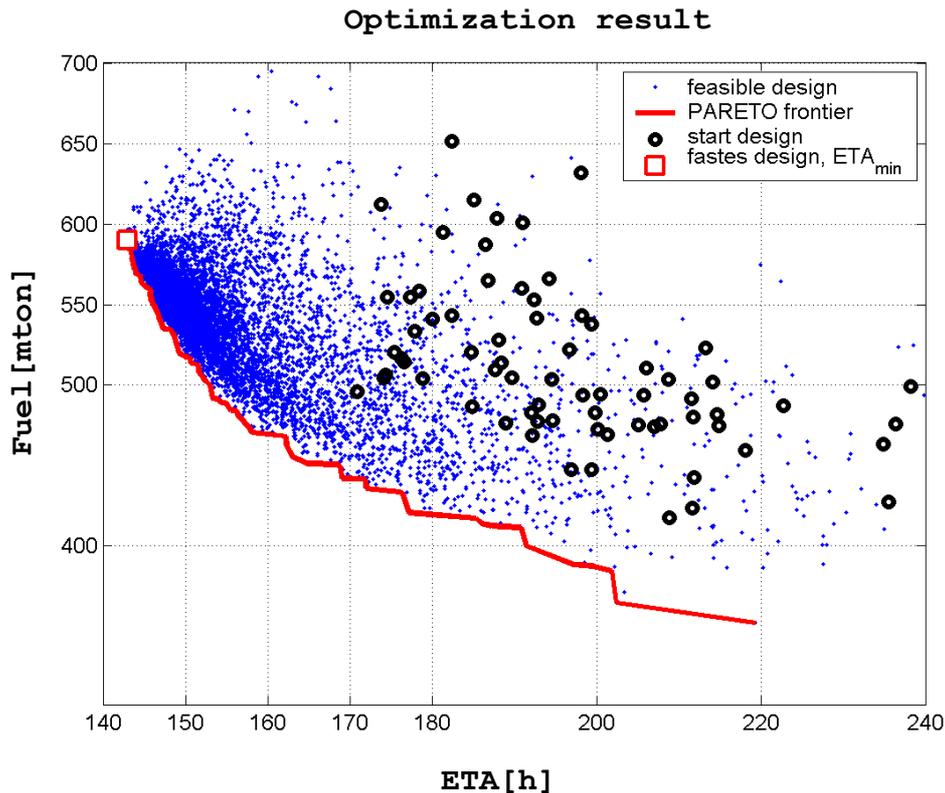
To cope with multiple objectives such as minimizing sailing time and fuel, the generated routes are typically compared by means of *Pareto frontiers*. These are graphs that visualize the trade-off of conflicting objectives based on a large set of example routes. Originally developed for the field of economics by the Italian economist Vilfredo Pareto, these frontiers show the optimum relation between two objectives. When applied to the field of weather routing, ship routes on the Pareto frontier can be regarded as 'equally good' when considering the defined objectives.

An example can be found in Figure 2-5, which shows the Pareto frontier of a specific voyage for the objectives arrival time (ETA; estimated time of arrival) and fuel minimization (Hinnenthal, 2008). A voyage below the red curve is infeasible. This could be because of maximum attainable speed for a given amount of fuel and sailing distance. A voyage above the red curve is feasible, but is not optimal.

Szłapczyńska proposed a multi-objective evolutionary weather routing algorithm MEWRA (Szłapczyńska, 2007, 2013). She defined three goal functions:

1. Minimizing passage time
2. Minimizing fuel consumption
3. Minimizing voyage risks

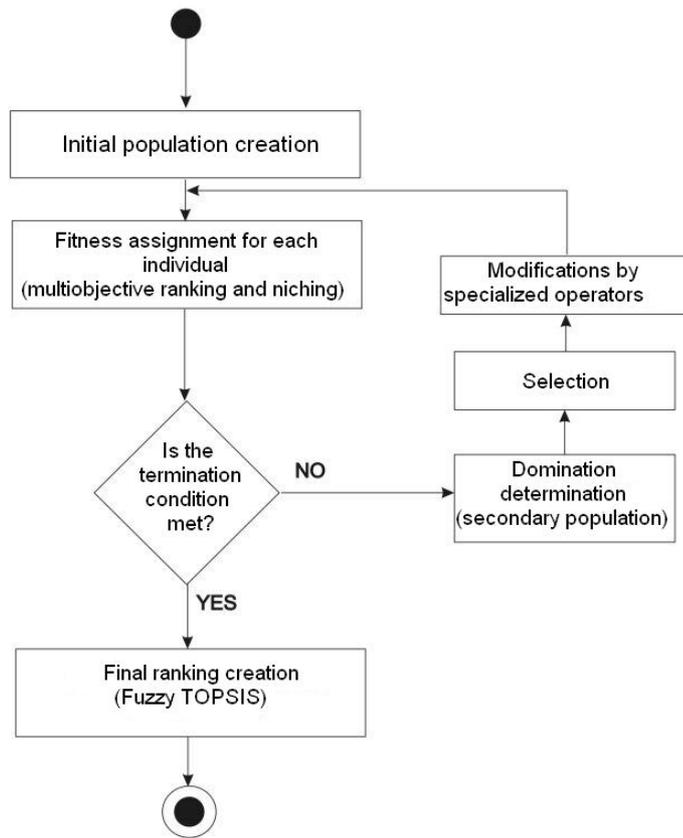
In her algorithm she generated an initial population of 50 randomly generated routes based on the isochrone method. Each generation, the Pareto-optimal routes are stored in a secondary population, to preserve all desirable solutions throughout the generation process. Evaluation of fitness is performed for the three defined objectives and for niching. This is a process of clustering neighboring individuals into groups to maintain multiple optima in the solution space.



**Figure 2-5.** Feasible route designs and Pareto-frontier (Hinnenthal, 2008).

When the final Pareto-optimum set of routes is found, a ranking of the alternative routes is performed. Ranking is done based on the preferences of the decision maker. A possible ranking methods is TOPSIS, based on the principle that the best alternative is simultaneously closest to the best possible solution and farthest from the worst possible solution. A variation to this method is Fuzzy TOPSIS that copes with fuzzy criteria (e.g. linguistic variables such as “good”) and fuzzy weights. Figure 2-6 depicts her multi-objective weather routing algorithm.

The technique of evolutionary algorithms is still very time-consuming. MEWRA is currently being implemented in Polish weather routing software NaviWeather. But finding an optimum route for an Atlantic crossing still takes around 20 minutes on a single computer. Another algorithm design by Marie and Courteille based on multi-objective genetic algorithms takes about 2 hours to perform the general optimization process (Marie, 2009).



**Figure 2-6.** Multi-objective evolutionary weather routing algorithm (MEWRA) by Szłapczyńska.

## 2.2. Path planning in grid-based environments

Path planning has been widely studied in the past decades to find the collision-free shortest path in some environment from a start to a goal location. For example in the fields of robotics, virtual environments in games and crowd simulation such problem instances have to be solved, sometimes in limited time to ensure good real-time performance. Path planning can be computationally hard, environments may contain multiple dynamic obstacles and bodies might have complex shapes or be subject to dynamic constraints.

For grid-based path planning problems the following two classes of search algorithms are widely applied:

- $A^*$ -based approaches (Hart, 1968; Holte, 1996; Botea, 2004; Harabor, 2013)
- Sampling-based approaches (LaValle, 1998; Karaman, 2011)

The  $A^*$  algorithm (Hart, 1968) is a very popular path planning approach as it guarantees returning the shortest path if one exists. Many grid-based search

algorithms are based on the  $A^*$  concept. In  $A^*$  the grid is searched node by node from start node until the goal is reached. The search is steered towards the destination by using a heuristic function that estimates the minimum remaining path costs for the shortest path via this intermediate node. Though optimal and fast for simple 2D search problems, the algorithm has several drawbacks. The search requires large memory overhead for large environments. When the algorithm is run at lower resolution, path may look unnatural. Another drawback is that when the environment contains many obstacles, the runtime can become very large and the search too slow. Therefore many techniques based on  $A^*$  have been proposed to improve the algorithm for specific problems.

Sample-based approaches are very effective for search problems in high-dimensional spaces and involvement of differential constraints. In such an approach a data structure of the configuration space is built by random samples over this space. Such a structure represents collision-free paths. An example of such a sample-based technique is Rapidly-exploring Random Trees (*RRT*) (LaValle, 1998). In *RRT* a search tree of feasible routes is built iteratively in a loop with the start location as root. During each step a randomly selected node in the search space is used to extend the tree. This is done by selecting the nearest node in the tree. Then a new edge and node are added to the graph in the direction towards the random node. This process is repeated for a specified number of samples. Alternatives have been proposed such as growing trees from origin and destination until both trees connect (LaValle, 2011).

Although *RRT* has probabilistic completeness (i.e. provided a solution exists, the probability of finding a solution becomes 1 when the number of samples approaches infinity) it suffers asymptotic optimality. That means that the probability of convergence to the optimal solution is less than 1. To overcome this problem Karaman developed *RRT\**, which is based on *RRT* but the route converges to the optimal solution with the increase of samples (Karaman, 2011). This optimality is obtained by iteratively optimizing the tree locally each time a node is added.

The main drawback of the sample-based techniques is that the solution route remains an approximation of the optimal route, unless an infinite amount of samples is used. In order to obtain an accurate solution a large amount of samples has to be explored and runtime increases linearly with the number of samples. Therefore our re-route algorithm does not utilize this sampling technique but rather uses  $A^*$  as basis for the search.

### 2.2.1. $A^*$ algorithm

Many grid-based path planning algorithms build upon the  $A^*$  algorithm (Hart, 1968). This algorithm is based on the Dijkstra algorithm that works by repeatedly visiting unexplored grid points starting from the origin until the destination point is reached (Dijkstra, 1959). Basically the search expands in outward direction from the start node. In each iteration the grid point with lowest path cost from origin is selected for exploration and removed from the list of unexplored nodes.

Whereas Dijkstra searches the unexplored grid blindly in any direction,  $A^*$  is steered towards the goal during its search by using a heuristic function that estimates the costs to the destination. This estimated cost ( $h$ -value) is added to the cost to reach this grid point ( $g$ -value) in order to obtain a lower bound on the route costs from origin to destination via this node ( $f$ -value). This total cost is used to prioritize unexplored nodes for further investigation.

A more detailed outline of the  $A^*$  algorithm is presented in Table 2-1. The  $A^*$  algorithm starts by initializing the start node's  $f$ -value and add the start node to the open list. In addition an empty closed list is initialized. Then a loop starts that repeatedly examines the node in the open set that has the lowest  $f$ -value. First this selected node is removed from the open list and added to the closed list. Then for each successor of this node the  $f$ -value is (re-)calculated for a route via this node. If this successor was not yet in the open list it will be added for later investigation. If it already was in the open list but a shorter route is found via the currently examined node, then the successor's  $f$ -value and index in the open list are updated. This process of investigating nodes in the open list is repeated until either the destination node is reached or until the open set is empty. In the latter case, no route is found.

#### ALGORITHM: $A^*$

```

1.  FIND_ROUTE(origin, dest)
2.  g(origin) := 0
3.  f(origin) := g(origin) + h(origin, dest)
4.  closedset :=  $\emptyset$ 
5.  openset := {origin}
6.  path :=  $\emptyset$ 
7.  while openset  $\neq \emptyset$ 
8.      current := openset.get[0] //node with lowest f-value
9.      if current = dest then
10.         return RECONSTRUCT_PATH(parent(current), dest)
11.     openset  $\cap$  origin :=  $\emptyset$ 
12.     closedset := closedset  $\cup$  origin
13.     for each n  $\in$  neighbors(current) do
14.         if n  $\cap$  closedset  $\neq \emptyset$  then
15.             continue loop
16.         g_tentative := g(current) + distance(current,n)
17.         if n  $\cap$  openset =  $\emptyset$  or g_tentative < g(n) then
18.             parent(n) := current
19.             g(n) := g_tentative
20.             f(n) := g(n) + h(n,dest)
21.         if n  $\cap$  openset =  $\emptyset$  then

```

```

22.             openset := openset U n
23. return "no route found!"

24. RECONSTRUCT_PATH(n1, n2)
25. path := path U leg(n1, n2)
26. if n1 ≠ origin then
27.     RECONSTRUCT_PATH(n1, parent(n1))

```

**Table 2-1.** A\* algorithm

In order to find the shortest distance in a grid-based environment the heuristic function should be *admissible*. This means that it should never over-estimate the cost towards the goal. If the heuristic is not admissible then there is a risk that nodes that are part of the actual shortest path will be found less interesting and not be selected for investigation (and thus not included in the solution). In addition, it is recommended that the heuristic is *monotone*. This means that with each incremental step towards the goal, the sum of traveled path cost plus heuristic should never decrease. If the heuristic is monotone, each node has to be processed not more than once and after processing the node will be stored in a closed list to exclude this node from further exploration. In this case the blue rows in Table 2-1 can be applied.

For complex search problems A\* might be time consuming as the *open list* of nodes to be explored can grow very large. Techniques to improve the speed of A\* are aimed at (Harabor, 2012):

- Reduction of search space by using abstraction
- Reduction of search space by using map detection
- Accuracy improvement of the heuristic function

*Hierarchical A\** reduces the search space by using the concept of abstraction (Holte, 1996). The idea is to find a solution in an abstract space first and use that solution as guidance of the search in the original search space. This is done by using the abstract solution as a *skeleton* for the search at the low-level solution. The intermediate states of the abstract solution are used as sub-goals on the low-level path. The low-level solutions between the sub-goals are linked to form the final refined path.

Other techniques that utilize abstraction of the search space are Hierarchical Pathfinding A\* (*HPA\**) and Partial Refinement A\* (*PRA\**). In *HPA\** (Botea, 2004) an abstract map of the space is developed in a pre-processing phase by overlaying the grid to form a set of disjunct rectangular *clusters*. A graph is built based on all entrances that connect neighboring clusters. This process of clustering can be performed for multiple levels for which route search can be performed. *PRA\** (Sturtevant, 2005) builds an abstract graph based on cliques and orphans in the low-level graph. Each Clique is transformed into a single node on a higher abstraction level. This process is repeated to create multiple abstraction levels until one single node is obtained.

Abstraction techniques have much better performance compared to  $A^*$  in terms of speed. But a drawback is that they are non-optimal in their returned solutions. Also they are a less suitable for dynamic environments as the abstract graph has to be adjusted after each weather forecast update.

Another technique to improve search speed is to reduce the search space by identifying areas in the grid that can be excluded in the search. A recent approach based on that principle is Jump Point Search (*JPS*), which identifies *jump-points* that skip certain regions that do not need further investigation (Harabor, 2011). *JPS* works well for grids with equally sized grid cells because it uses symmetry in the search space. This is not the case for our problem and therefore it is not suitable for our problem.

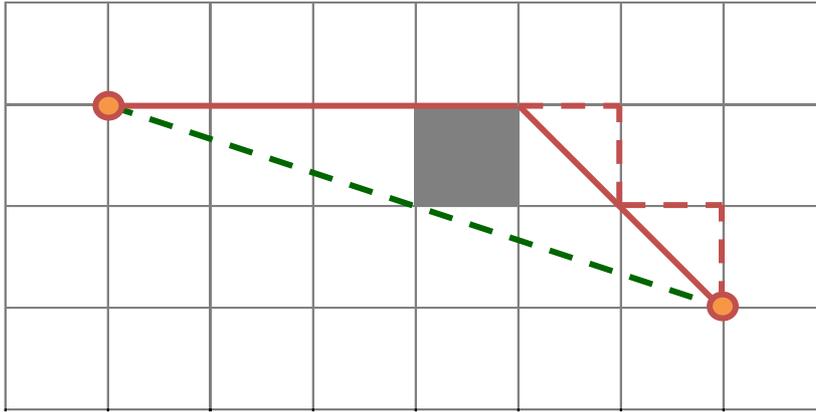
### 2.2.2. Any-angle pathfinding in grid-based problems

Various algorithms have been developed for the problem of path planning in a continuous search space with grid-based obstacles. Such problems allow unrestricted *any-angle* traversing through the environment, but all obstacles are related to a grid. In other words, grid cells are either *traversable* or *non-traversable*. This problem could be solved by constructing a *visibility graph* of the environment and consequently searching this graph. This visibility graph is obtained by making a clique of the set of grid points and removing all edges that overlap any obstacle. But this approach is not recommended for large environments as the number of edges grows quadratically with the number of nodes.

A simple solution to obtain a continuous route through the search space is to apply *string pulling* (Botea, 2004) to the grid-based optimal solution. *Pulling* both ends of the solution path makes the route *taut*, meaning that it cannot become any shorter and each route point is only visible by its adjacent route points. This string-pulled path basically consists of origin, destination and corner points of obstacles. All other intermediate nodes will be deleted from the solution path. String pulling can be performed online while running the search algorithm or in a post-processing phase after a grid-based solution is found.

The main disadvantage of string pulling is that it does not guarantee the shortest path. Figure 2-7 illustrates this principle, showing non-optimality of a string-pulled grid-based shortest path solution.

A number of algorithms utilize the concept of string pulling *online* while exploring grid points. *Field  $D^*$*  (Ferguson, 2006) uses linear interpolation to smooth path during each node exploration. When a node is processed a closed perimeter of edges that connect all the neighbors of this node is checked. The point on this perimeter that gives the lowest path cost from origin to node is used as parent for the currently investigated node. This technique works well, though might return non-optimal paths when obstacles are present.



**Figure 2-7.** Non-optimal solution after string-pulling. The green dotted line is the optimal path from origin to destination. The red dotted line is the solution returned by  $A^*$ . After string pulling the non-optimal solid red line is obtained.

Another technique is  $\Theta^*$  (Nash, 2007), which pulls routes to the root node of the parent in case this root is visible. This string pulling is done after each node exploration.  $\Theta^*$  is a fast and simple algorithm, but explored paths are not monotonically increasing and therefore the algorithm is non-optimal. With each incremental step towards the destination, the estimated path cost may decrease. Therefore intermediate path costs might be overestimated. This is illustrated by the fact that string pulling may shorten the path from origin when the previous grid point is removed from the path.

Another drawback of searching via grid points is that a shortest path via a narrow passage might be omitted. For example when the start node is visible from the destination, but not from some intermediate node due to a blocking obstacle, the optimal shortest route will not be found and a corner node of this obstacle becomes part of the returned solution. In general  $\Theta^*$  returns more accurate solutions than  $A^*$ , but at higher costs due to all visibility checks.

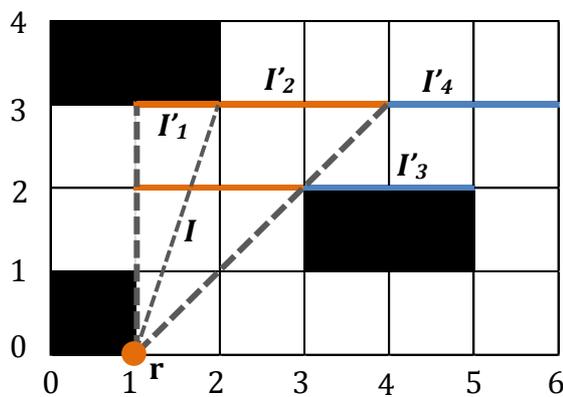
In *Block  $A^*$*  (Yap, 2011) the environment is split up in square shaped blocks for which all optimal internal paths are pre-computed and stored in a database. These blocks are subjected to an  $A^*$  search for which the  $f$ -value of a block is equal to the lowest path cost via any cell inside the block. Each block (or node) exploration involves an update on all internal path costs and  $f$ -value of the neighboring blocks. Basically *Block  $A^*$*  is a hierarchical approach for any-angle traversing, which results in much faster and more accurate solutions to search queries compared to  $A^*$ . Since *Block  $A^*$*  requires pre-computation of distances for each block, the method is not suitable for dynamically changing environments.

### 2.2.3. Optimal any-angle algorithm

Harabor and Grastien sketched an optimal any-angle algorithm for grid-based obstacles, named *Anyra* (Harabor, 2013). In their approach the environment is searched over intervals instead of grid points. Basically an interval is a set of

adjacent edges in one row of the grid that have one common parental grid point as root node that is visible from each point in the interval. While running the algorithm, the lower bound on the path cost via an interval is determined by the sum of the actual path cost from origin to the root node and the lowest estimated route costs from the root node to the destination via any point on this interval.

The approach of searching over edges instead of grid points enables to calculate path costs that are admissible and monotonically increasing. As mentioned in section 2.2.1 this is a requirement for optimality. If the applied heuristic for path cost estimation from the interval to the destination is admissible then the optimal solution is guaranteed. The exploration of intervals is entirely performed online; no pre-computation is required.



**Figure 2-8.** Example instant of Anya.

An interval  $I$  is a set of continuous pairwise visible points from any row of the grid between grid points  $a$  and  $b$ . These points  $a$  and  $b$  are defined at the corner points of obstacles and the intersections with visibility lines of the predecessor interval. All points in the interval have a common root node  $r$  that is observable and is not part of this interval. This root could be an endpoint  $a$  or  $b$ , depending on observability of the interval from the predecessor root node. Therefore membership of points  $a$  and  $b$  of this interval depends on visibility of the predecessor root node.

The tuple  $(I, r)$  is a search node used in the  $A^*$  loop for pathfinding. A successor  $(I', r')$  of search node  $(I, r)$  is a tuple for which each point  $p' \in I'$  is reached by a taut path  $\langle r, p, p' \rangle$  starting at  $r$ , passing through some  $p \in I$  and  $r'$  is the last common point of each of these paths.

An example showing the principles of Anya is presented in Figure 2-8. Tuple  $(I, r)$  with interval  $I$  and root point  $r$  has 4 successors, being the observable tuples  $(I'_1, r)$  and  $(I'_2, r)$  and the non-observable tuples  $(I'_3, r)$  and  $(I'_4, r)$ .

Observability can be concluded from the grey dotted visibility lines from root node  $r$ . For the observable intervals  $I'_1$  and  $I'_2$  the new tuples  $(I'_1, r)$  and  $(I'_2, r)$

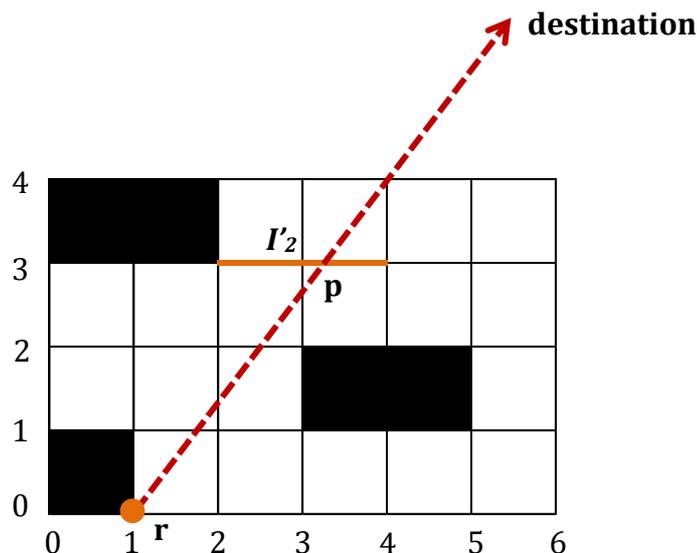
are formed with  $r' = r$  since the root of the parent interval  $I$  is still observable from any point  $p$  on the interval  $I'$ . For non-observable sets of points from  $r$  a half open-interval  $I'$  should be defined that is closed at the corner point  $a$  or  $b$  from where the root node  $r$  is still visible. This specific corner point becomes the root node of this interval.

Each newly generated tuple is ranked based on its  $f$ -value, calculated in analogue to  $A^*$ . The  $f$ -value of a tuple is the lowest cost function of a path from origin via the root node and via any point  $p$  on the interval to the destination, based on an admissible and monotone heuristic function.

$$f(p) = g(r) + d(r, p) + h(p, destination)$$

where  $g(r)$  is the actual path cost from origin to the root node,  $d(r, p)$  is the lowest path cost from root  $r$  to point  $p$  and  $h(p, destination)$  is the estimated costs to the destination according to an admissible and monotone heuristic function. Figure 2-9 shows this concept for the observable interval  $I'_2$ . Given some destination the lower bound on the costs of a path from origin via  $r$  and  $I$  to this destination is via point  $p$ .

The search query begins with a tuple for the start node that is added to the open list. Then repeatedly the tuple with lowest  $f$ -value is explored and added to the closed list, until an interval is discovered that contains the destination node.



**Figure 2-9.** Evaluation of tuple  $(I'_2, r)$ . Point  $p$  on the interval  $I'_2$  gives the lowest estimated path costs from root node  $r$  to the destination via interval  $I'_2$ .

## 3. SAFETRANS MONTE CARLO SIMULATION

### 3.1. A brief history

SafeTrans is a software simulation tool to design and evaluate offshore operations and weather routed marine heavy lift transports (self-propelled and towed). It started as a JIP in 1997 with the goal to improve the design of tow and installation operations by utilizing modern probabilistic and consequence-based design methods, improved weather forecasting and modern PC's (Aalbers, 2001).

In 2001 the software was released and a user group was formed as a platform to exchange user experience and provide feedback to further improve the design tool. The user group is still active these days and consists at present of 18 members from various corners of the heavy lift industry, such as oil majors, marine warranty surveyors and shipping companies.

Ten years of experience with SafeTrans led to the decision to make a major overhaul to the software. This resulted in SafeTrans 5, which was released in 2011 and presented in (Grin, 2011).

### 3.2. SafeTrans design methods

The software is used to determine design loads for a specific transportation taking into account historical weather data for the applicable sailing area and period of the year. Such design loads form the basis for the design of a heavy transport and are used to determine the sea fastening design and structural strength of cargo and ship. They can also be used as input for more advanced fatigue analysis.

The software contains two design methods:

1. *Voyage Motion Climate (VMC)*

Statistical approach based on wave scatter diagrams (tables that contain the joint probability of significant wave height and period for a particular area). VMC models the probability of all sea states on a voyage in order to determine long-term statistics of ship motion responses.

2. *Monte Carlo Simulation (MCS)*

Stochastic discrete-event simulation to determine ship responses based on weather routed voyage simulations. The method uses a 10-year historical weather database and a model of a captain as basis for the simulation.

VMC is a quick and straightforward mode, but cannot account for weather routing. MCS is specifically designed for this purpose and therefore only MCS is considered in this study.

In MCS a heavy transport is simulated by multiple repetitions of that particular voyage. A set of input parameters should be inserted, such as hull shape, loading condition, route settings and voyage criteria. Each single run is subjected to a randomly selected starting date within a user-defined window. Then the historical weather forecast that relates to this starting date is presented to the ship 'master', who modeled in the so-called Captain's Decision Mimic (CDM). This CDM navigates the ship in 3-hour steps until the port of destination is reached, while taking into account the user-defined voyage criteria. For all runs the motion responses are collected to build a long-term distribution of the extreme vessel responses.

A Monte Carlo Simulation is a stochastic discrete-event type of simulation. The stochastic nature lies within the varying environmental conditions between the individual runs. The discrete-events consist of the 3-hour sailing steps (or waiting in case of severe environmental conditions). Typically about 250 runs are made to obtain a statistical refined and reliable long-term response distribution.

### 3.3. Initializing a MCS-case

Before the actual simulation starts, the user has to define input data for this transport. The required information can be categorized as follows:

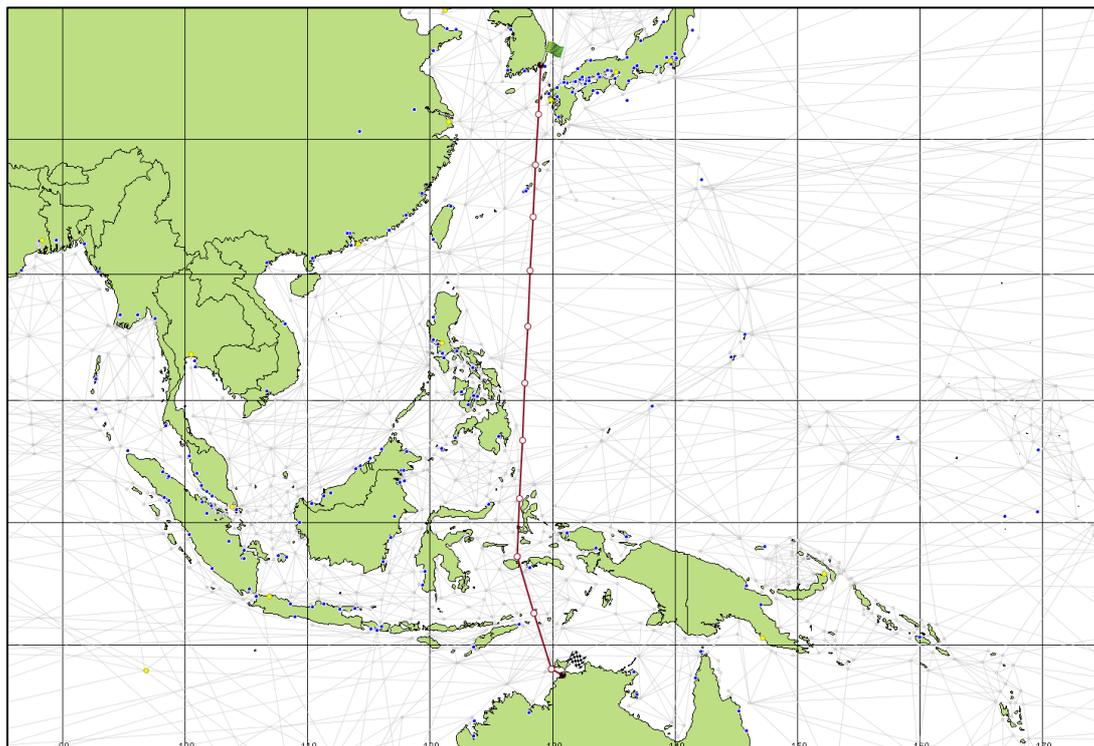
- CDM and risk module settings (delay penalty class, company safety rating, cost settings)
- Vessel description (hull geometry, appendices, rudder details)
- Loading condition (draught, stability, displacement)
- Wind coefficients and profile
- Current coefficients
- Route settings (Departure, destination, wish points, safe havens, forbidden areas)
- Resistance and propulsion settings (resistance curve, trial power and speed, bollard pull, engine MCR)
- Motion signals (parameters to be monitored during the simulation, such as transversal acceleration in a cargo item)
- Voyage criteria:
  - Default required criteria:
    - Maximum significant wave height
    - Maximum wind speed
    - Maximum vertical acceleration at the bow

In addition, user-defined criteria such as maximum allowable vessel roll angle can be entered.

When all input parameters are entered, the ship motion database has to be generated. Ship motions are calculated by means of the 2D-linear strip theory for slender ships with a length-breadth ratio  $L/B > 2$ . The ship motions are expressed as significant double amplitudes (SDA) values, which resemble the

most probable maximum amplitude of the motions. Details of this approach are left out of the scope of this research study but can be found in the SafeTrans 5 theory manual (MARIN, 2011). The ship motion database is used by the CDM during the runs to see if all (motion) criteria are being met given the presented weather forecast and to determine the ship motions for each 3-hour sailing step.

The initial sailing route is generated with the help of an underlying grid, which can be seen as the web of grey lines in Figure 3-1. Here an example route is shown from Busan (Korea) to Darwin (Australia). For ocean crossings, route generation is done based on great circle lines.



**Figure 3-1.** Initial route generation based on underlying grid.

### 3.4. Description of the MCS-mode

Once all input has been generated, the actual simulation can start. Figure 3-2 depicts the flow diagram of SafeTrans MCS as applied to a single simulation run for a heavy lift transport. The simulation starts at the top of the scheme with the first of many runs. The CDM (orange box in figure) consults the IMDSS forecast weather database for waves (significant wave height) and wind (speed and direction). If they exceed the user-specified thresholds then the voyage is delayed. This loop continues until the forecast shows a few days of clear sailing along the user-specified route.

Once the ship has departed its port, it cruises along the preferred route for 3 hours at a speed dictated by the vessel power characteristics. Added resistance due to wind, waves and current is included in this analysis. At the end of this 3

hours sailing step, SafeTrans calculates the ship and cargo displacements, velocities and accelerations based on the nowcast from the weather database at the ship's 3-hour track and the pre-calculated vessel motion lookup tables. These calculated cargo motions are archived into the voyage statistics database and can be used in a later stage to develop statistics for the monitored signals.

SafeTrans then begins a series of checks on the new position. First it checks if a towline break event has occurred (in case of a towed operation). Then the risks are calculated and a check is performed to see if the final destination has been reached. If not, then SafeTrans checks what the present conditions and motions are or whether it is in the influence of a tropical revolving storm (TRS). If the forecasted or actual ship behavior exceeds the thresholds then the power and course settings or route plan are changed. After that, the program returns to the CDM to make a decision for the next period of 3 hours.

When the ship has reached its final destination the actual run terminates and the next run starts. Typically around 250 runs are made to collect sufficient data for the long-term distribution of ship motions.

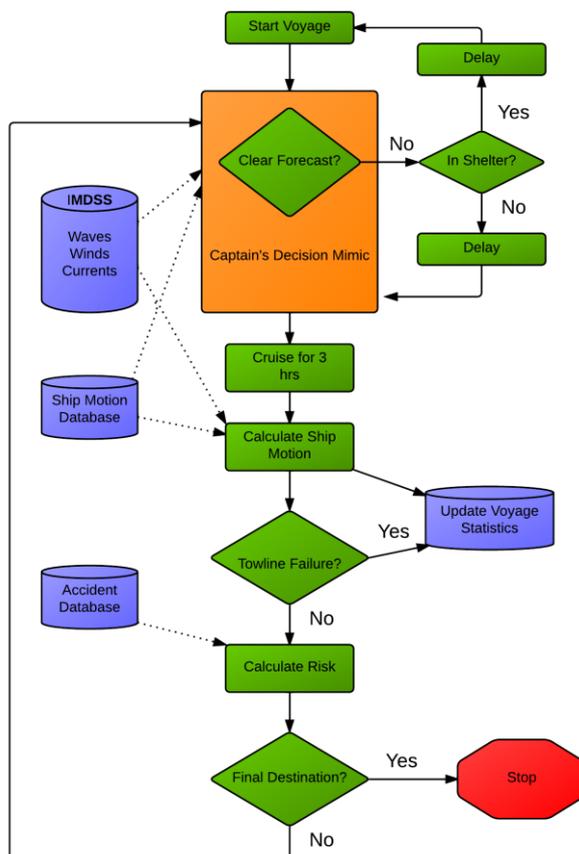


Figure 3-2. Flow diagram of MCS.

### 3.5. Metocean weather database

The weather forecast database that is included in SafeTrans is the Integrated Marine Decision Support System (IMDSS) wind/wave database developed by Oceanweather. It contains time series of spectral parameters of wind, sea and swell. The spectral parameters are:

- Significant wave height ( $H_s$ )
- Peak period ( $T_p$ )

This global database has a 2.5 by 2.5 degrees resolution. This results in trapezium-shaped grid cells and on the equator it resembles a square of 150 by 150 nautical miles. The latitude ranges from 70 degrees South to 75 degrees North and the longitude ranges from 180 degrees West to 180 degrees East. The database covers 10 years of hind cast data (1995 to 2004). The time step within this database is 12 h and for each day a forecast is given in steps of 12 h up to 168 h (7 days) ahead.

In addition to this wind/wave database, the program uses a hurricane database. This global database is provided by NCDC and contains hurricanes (TRSs) from 1972 to 2004. The overlap of forecast database and hurricane database could cause problems with double presence of hurricane waves and winds, because also the forecast database contains TRSs. To avoid double occurrence effects, MCS-runs with a TRS within the forecast database will be stopped and denoted as unfinished.

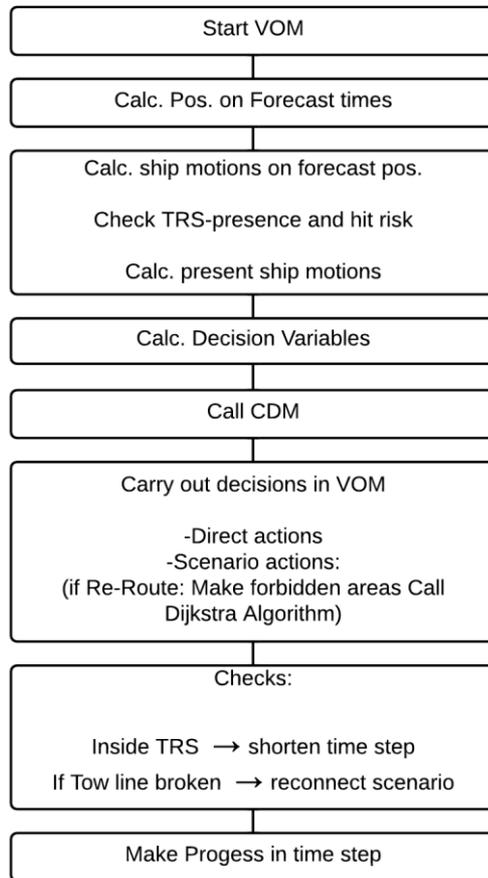
There is also a current database present in the software. It contains the seasonal averaged ocean currents and is obtained from ocean drifters. The database has a global coverage and the spatial grid is 2.5 by 2.5 degrees as well. The database does not contain tidal streams.

### 3.6. Decision making process

All vessel operations are controlled in a central coordinating module, called the Vessel Operating Module (VOM). The VOM may be interpreted as the mathematical helmsman on the vessel. It is the central object that is linked with the CDM (decision making), the weather forecast and ship motions databases, ship controls (course, speed and motions) and the route plan.

The steps taken by the VOM to determine and carry out decisions for the next 3 hours phase are illustrated in Figure 3-3. Decision-making in MCS is done by transforming a set of decision variables (DV) into a set of decisions (D). The VOM collects information such as the actual vessel condition and weather forecast to calculate the actual DV probability vector. In total, there are 13 DVs that quantify the actual status of the vessel relevant to the transport operation. These DVs were selected based on an extensive survey among 23 experienced seafarers in the field of offshore installations, tow operations and heavy lift transportations.

Once the DVs have been determined, the VOM calls the CDM to calculate the decision probability vector  $D$ . In total 10 decisions can be taken by the CDM.



**Figure 3-3.** VOM decision making process.

Table 3-1 contains an overview of all 13 decision variables and 9 decisions. The 10<sup>th</sup> decision is to continue to destination without any change in settings and route plan. The values of the DVs range from 1 to 10, where 10 is extremely bad. The decisions *Change route* (No. 2) and *Avoid hurricane* (No. 4) are highlighted, since these are the decisions to make a new route plan based on Dijkstra shortest path for a given weather forecast. In the next section the procedure to make a new route plan is explained in more detail.

The transformation from the decision variables to the decision(s) is carried out as follows:

$$D = K + DV * A$$

Where,

$D =$  Vector with basic decision probabilities

- $K$  = Vector of initial constant values such that  $(K + 1 \cdot A)$  gives the decisions for perfect conditions.
- $DV$  = Vector with decision variable probabilities
- $A$  = Matrix (13 x 10) representing the transfer coefficients from  $DV$  to  $D$ .

Decision variables	Decisions
1. Time weighted forecast	1. Head for shelter
2. Sheltering potential	<b>2. Change route</b>
3. Re-routing potential	3. Change course / speed
4. Delay risk	<b>4. Avoid hurricane</b>
5. TRS hit risk	5. Go to survival mode
6. Value of consequences	6. Shorten tow line*
7. Vulnerability	7. Rendering winch on*
8. Crew quality class	8. Reduce tow line tension*
9. Comfort situation	<i>When in shelter:</i>
10. Slamming / green water risk	9. Wait for weather
11. Excessive roll / capsize risk	
12. Tow line break risk*	
13. Tow line bottoming risk*	

**Table 3-1.** Decision variables (DV) and decisions (D).

Matrix  $A$  has been obtained from a multiple regression analysis on the results of the enquiry among the seafarers. After the decision vector is calculated, the values are normalized. The decision probabilities are calculated based on 12-hours updates of the weather forecast, while decisions are carried out for 3-hour periods. Therefore the decision vector is scaled, such that the most probable decision probabilities are maintained. It should be noted that multiple decisions can be taken simultaneously, for example change route and avoid TRS.

The actual decision made by the CDM is randomly selected while considering the probability of each decision. To achieve this, a cumulative distribution is generated for all relevant combinations of decisions. Then a uniformly distribution random number is picked in the range [0,1] and the corresponding decision(s) are returned to the VOM.

The VOM carries out the decisions made by the CDM. First, checks are made to ensure the requests satisfy basic physical constraints, e.g. VOM prevents a requested power reduction to reduce towline tension when power reduction has already been applied to reduce speed for comfort. Furthermore, the decision to change speed & heading for comfort may not bring an improvement, and thereto the realization of that decision is based on a scenario in which the optimum

situation is sought. Also the decision to re-route may not be feasible and in such a case the vessel will stay on its original track. The decision *Wait for weather* (No. 9) only applies if the vessel is still in port or in shelter.

### 3.7. Re-route algorithm in SafeTrans

Weather re-routing will be performed when the decision *Change route* (No. 2) or *Avoid hurricane* (No. 4) has been taken. Then a communication scheme as illustrated in Figure 3-4 will be started to make a new route plan.

Various objects are involved in this process. As explained in the previous section first the CDM determines which decisions should be taken based on the decision variables. The VOM then instructs the Helmsman to carry out these decisions. In case of the decision *Change route* the Helmsman informs the Meteorologist to make a list of time-dependent forbidden areas based on the given weather forecast. Then the VOM delegates the process of making a new route to the Navigator. The navigator forwards this job to the RoutePlanner object. He informs the RoutePlanner on the forbidden areas and he defines a re-route area. Finally, a Reroute object is used by the RoutePlanner to perform the actual path finding job.

The re-route area is a square-shaped box that acts as search boundary and contains a set of 2.5 by 2.5 degrees grid cells around the actual ship route, covering the full seven days forecast period plus a margin in which the re-routing track has to be found. For this search space a 3D-graph (2D-space and 1D-time) is constructed as follows:

1. Nodes: For each time interval the corner locations of all grid cells are added to the set of nodes, with overlapping corners merged to a single node. For each node, there is a time interval to the next time layer of maximum 24 hours to the start of a next day.
2. Edges: For each single node there are directed edges to the 8 nodes around this node in the same time layer. In addition, there are 8 *L-shaped* directed edges similar to the knight movement in the game of chess. Finally there is 1 directed edge to the same coordinate in the next time layer (waiting for weather update at the start of the next day).

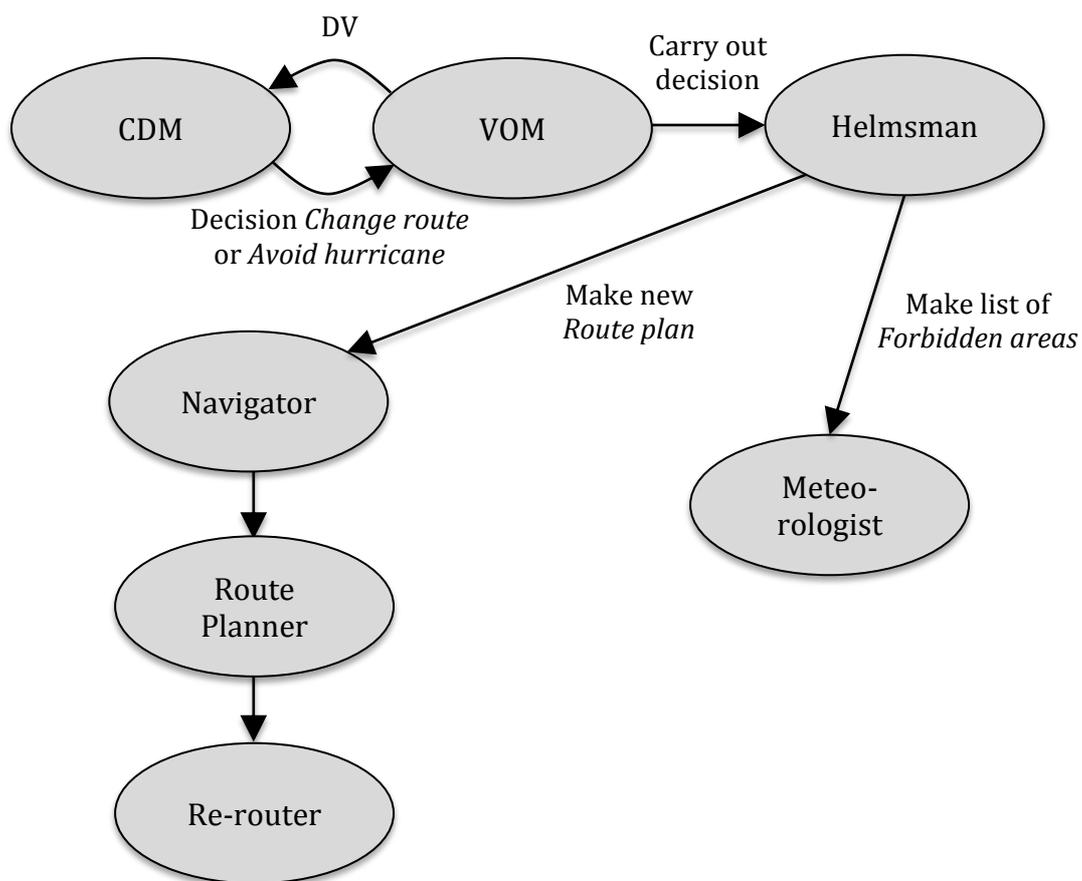
Edges are blocked when they cross a forbidden area. An area can be statically or dynamically (time-dependent) blocked:

1. Static:
  - Land
  - User-defined forbidden areas
2. Dynamic:
  - Time-dependent forbidden areas where wave height or wind speed exceeds the operational criteria
  - Time-dependent grids where a hurricane is present

When cells do not represent land or forbidden areas, it is either *on* or *off* for 12 hour blocks up to 168 hours. A grid cell in time is *off* if the forecasted significant wave height or the wind speed for that area exceeds the user-defined thresholds. A grid cell is also marked *off* if a hurricane is forecasted.

Weather re-routing is done based on an implementation of the famous *Dijkstra*-algorithm (Dijkstra, 1959). A shortest time path is searched in the grid based on a fixed ship speed defined by the user. Only for the directed edges to the next time layer, the ship speed is zero.

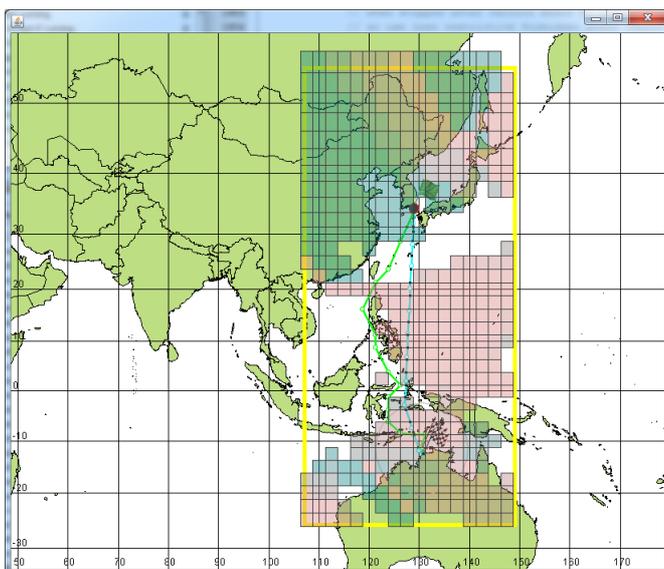
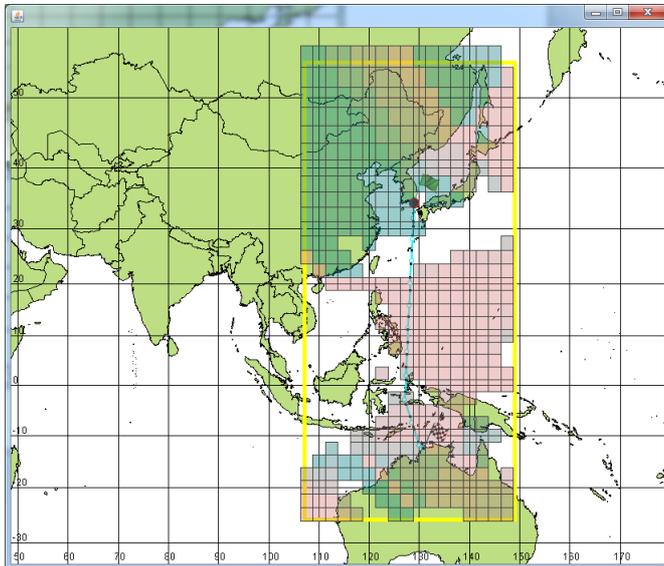
In a loop the graph nodes are checked for shortest travel time to reach that specific node. The loop terminates when a shortest route is found. If the algorithm cannot find a route, the search process starts over with less restrictive forbidden areas. There is a safety factor used for this relaxation, which is requested by the user as part of the simulation input.

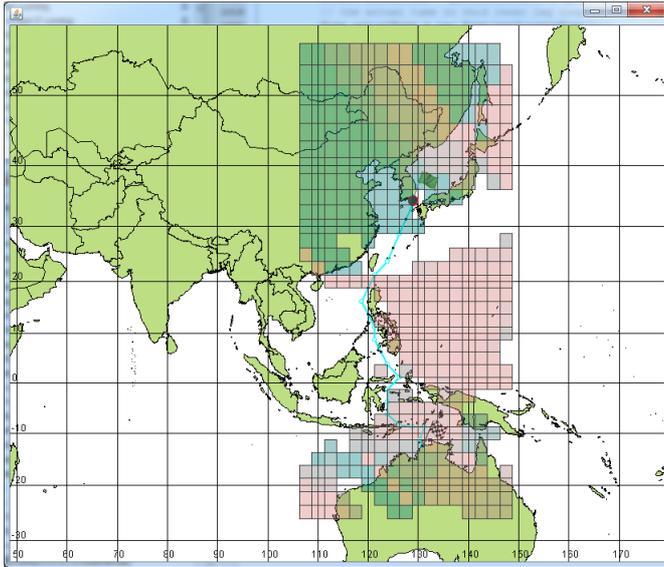


**Figure 3-4.** Work flow for making a new route plan.

On Figure 3-5 an example is given of a re-route result based on a given weather forecast for a voyage from Busan (Korea) to Darwin (Australia). The yellow square indicates the search boundary. The colored grids are time-dependent forbidden grid cells. The time-dependency is shown by the color of the grid cell. The initial route plane is the blue-colored line shown in the top figure. This route is identical to the route as presented earlier in section 3.3.

The middle figure shows the route that is found by the Dijkstra-algorithm. This is the green-colored line in the figure. This route replaces the actual route plan, as shown in the bottom figure. When the vessel leaves a 2.5 by 2.5 degree grid cell, a route stretching procedure is applied to the route. This is done because the Dijkstra route consists of a set of straight line sections that link consecutive grid cells. The route stretching procedure returns a smooth route that avoids islands, forbidden areas and unsafe coast areas.





**Figure 3-5.** Re-routing in MCS. Top: Blue route is the initial route that exceeds weather criteria; Middle: Green route is Dijkstra-route that meets weather criteria; Bottom: New Dijkstra-route implemented.

### 3.8. Example case

An example of a MCS-study is presented in this section. It is a self-propelled heavy lift transport from Busan (Korea) to Darwin (Australia). The simulation characteristics and the vessel specifications are given in Table 3-2 and Table 3-3. The initial sailing distance is 2914 nm. The calm water speed at 85% power is around 16 kn which results in a voyage duration of 182 h (about 7.5 days).

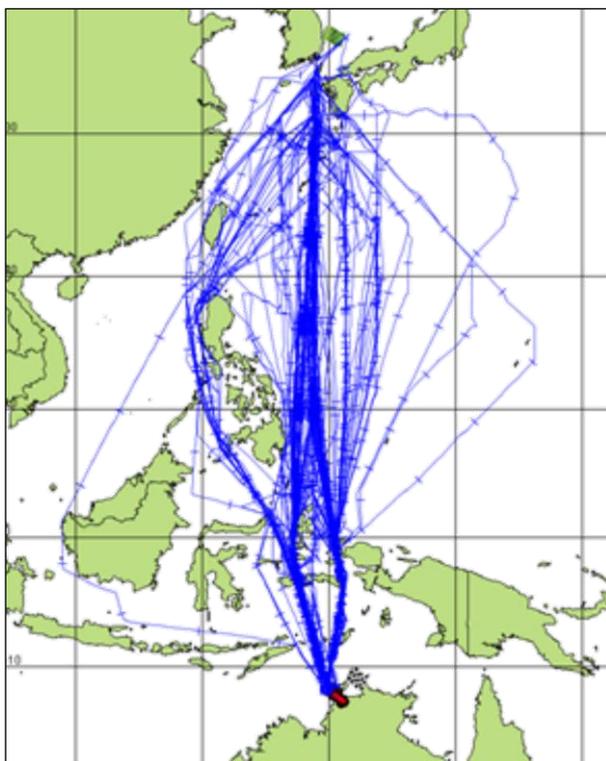
The results of all 300 simulation runs are plotted in Figure 3-6. Approximately 80% of the runs did not do any re-routing and followed the initial route plan. The re-route distance was typically within 400 nm, with some exceptions up to 700 nm. The longest voyage duration had a total voyage duration of more than 21 days. As can be seen there are some example runs that have a large deviation of the original route. This is most likely the result of hurricanes in combination with strict weather criteria, forcing the vessel to deviate significantly.

Vessel specifications	
Type:	Heavy lift vessel
Length:	127.40 m
Beam:	22.80 m
Draft:	5.60 m
Depth to main deck:	13.00 m
Transverse stability:	2.00 m
Natural roll period:	18.5 s

**Table 3-2.** Main dimensions of the heavy lift vessel.

MCS settings	
Type:	Self-propelled
Period:	All year
From:	Busan (Korea)
To:	Darwin (Australia)
Nr. of runs:	300
Safe havens:	Busan, Nagasaki (Japan), Manila (Philippines), Darwin
Transport criteria:	
Hs:	4.5 m
Wind:	45 knots (abt. Beaufort 9)

**Table 3-3.** MCS settings for the MCS-study.



**Figure 3-6.** Results of 300 Monte Carlo simulations from Busan to Darwin.

## 4. DESIGN ALTERNATIVE RE-ROUTE ALGORITHM

In the previous chapter the principles of the actually implemented re-route algorithm have been presented. In this chapter an alternative re-route algorithm is proposed that meets the research objective as defined in section 1.2. First the terminology is explained that is used throughout the chapter. This is followed by a description of the re-route algorithm. Special considerations are made for extracting a path and determining its cost, since this requires a specific check during each node exploration.

Harabor and Grastien's optimal any-angle algorithm *Anya* will serve as basis for the design of the re-route algorithm (Harabor, 2013). *Anya* considers grid-based obstacles in a continuous space and returns an optimal path solution for search queries. In our problem the search space is extended by the time dimension and the cost function is modified to allow for a trade-off between sailing time, fuel costs and late arrival.

### 4.1. Preliminaries

The terminology that is used to describe the algorithm is defined in this section. The definitions are 3D (space + time) extensions to the definitions used in the 2D (space) *Anya* algorithm in order to make the algorithm suitable for environments changing in time.

The search area is a cubic shaped bounded region of adjacent non-overlapping *grid cells*. Each grid cell is formed by the Earth surface planar subdivision into squared shaped cells of 2.5 degrees latitude by 2.5 degrees longitude and the time dimension subdivided into blocks of 12 hours. The range of 0 to 168 hours might contain forbidden 12 hour blocks depending on the user-defined voyage criteria. The intersections of all 2.5 degrees spaced latitudes and longitudes with the 12 hour time intervals are the *grid points* in the search space. The search area is defined as a box from the grid of the actual vessel position to the grid that contains the port of destination including a 20 degrees margin. The time dimension is restricted by the navigational reach according to the ship's speed range. An example of an Earth planar subset is depicted in Figure 3-5.

A cubic shaped grid cell is an open set of *interior points*. The 6 faces of a grid cell are each open intervals of *intermediate points*. The 12 edges are open intervals of *corner points* and the 8 vertices where 3 edges meet are *grid points* and are *discrete points* of the grid. Each point  $p = (lat, lon, t)$  is a coordinate where  $lat \in [latitude\ min, latitude\ max]$ ,  $lon \in [longitude\ min, longitude\ max]$  and  $t \in [0, time\ max]$ . The union of all grid cells is the *search grid* in which an optimal route should be found.

A grid cell represents either land or sea. All land grid cells are *non-traversable*. The sea grid cells are either *traversable* or *non-traversable* depending on whether the environmental conditions meet the transport criteria (wind and

significant wave height). These conditions are kept constant for each 12 hour time interval.

A *path leg* is an orthodromic (or Great circle) curve between two corner points. This is the shortest direct path between those two points over the surface of the spherical shaped Earth. The Earth is not an exact sphere; therefore a small correction should be made compared to the spherical distance. In this study the WGS84 Earth model is used for this purpose. WGS84 is an international geodetic standard for the Earth model that also serves as reference coordinate system for the satellite navigation system GPS. Details of WGS84 can be found in (NIMA, 1997). An orthodromic line has a varying heading along its curve, except for the directions North (0 rad) and South ( $-\pi, +\pi$  rad). The grid cells as defined in the weather database are split by loxodromic curves, meaning lines with a constant heading. For our experimental visualizations the Mercator mapping is applied. In this mapping the loxodromic lines are straight and the orthodromic lines are curved.

An *any-angle path*  $\pi$  is a sequence of points  $\{p_1, \dots, p_k\}$  where  $p_1$  is the actual vessel position,  $p_k$  is the point of destination and each  $p_i$  is visible from  $p_{i-1}$  and  $p_{i+1}$ . The cost of a path  $\pi$  is the sum of the costs of the *path legs*  $\{p_i, p_{i+1}\}$ , being a function of sailing time, fuel consumption and late arrival. The points  $p_1$  and  $p_k$  are the *start* and *end point* of the path; all other points are *turning points* in the route. Each turning point  $p_i$  is visible from  $p_{i-1}$  and from  $p_{i+1}$ , but  $p_{i+1}$  is not visible from  $p_{i-1}$ . In an optimal path all turning points are corner points (Harabor, 2013).

## 4.2. 3D any-angle path finding

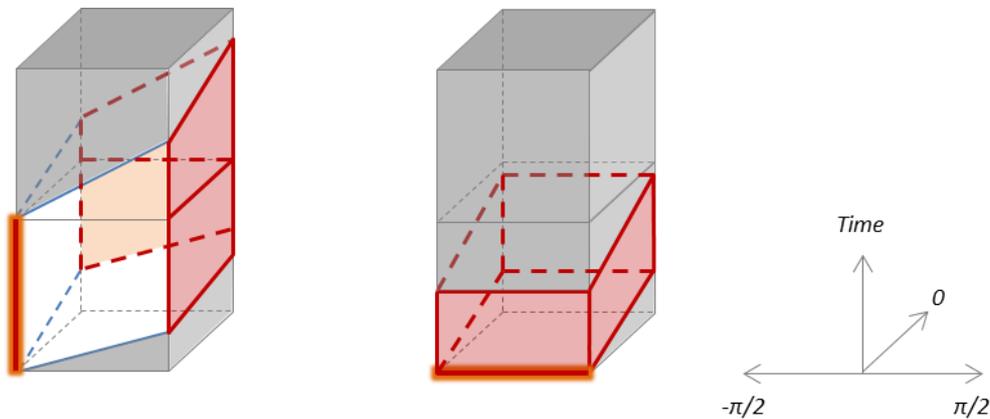
*Anya* explores the search space by  $A^*$  over a set of search nodes. The general idea of *Anya* is to find a path from origin to destination via a set of intervals. This idea allows the cost estimate function for explored paths to be admissible and monotonically increasing while searching the environment. This is a requirement for finding the optimal solution for a search problem. This requirement may not be met in a discretized search space for which intermediate path cost estimates might exceed the final path cost. The principles of  $A^*$  and *Anya* have already been explained in respectively section 2.2.1 and 2.2.3.

For the stepwise description of the algorithm the following definitions of a root  $r$  and an interval  $I$  are required. Whereas in the 2D *Anya* algorithm a root is a grid point and an interval is a line segment, in our 3D algorithm a root is a line segment and an interval is a surface.

**Definition 1:** A root  $r$  is a set of contiguous pairwise visible corner points of one edge of a grid cell. All points of a root have either equal latitude and longitude (space root) or equal time (time root). The set is at least at one end of the range bounded by a discrete point of this grid cell.

**Definition 2:** An interval  $I$  is a subset of contiguous pairwise visible points of one face of a grid cell. An interval is physically bounded by Earth coordinates  $a$  and  $b$ . The points with coordinates  $a$  and  $b$  might be corner points. All other points in the interval are intermediate points. For each coordinate in the range from  $a$  to  $b$  the points are bounded by times  $t_{min}$  and  $t_{max}$ .

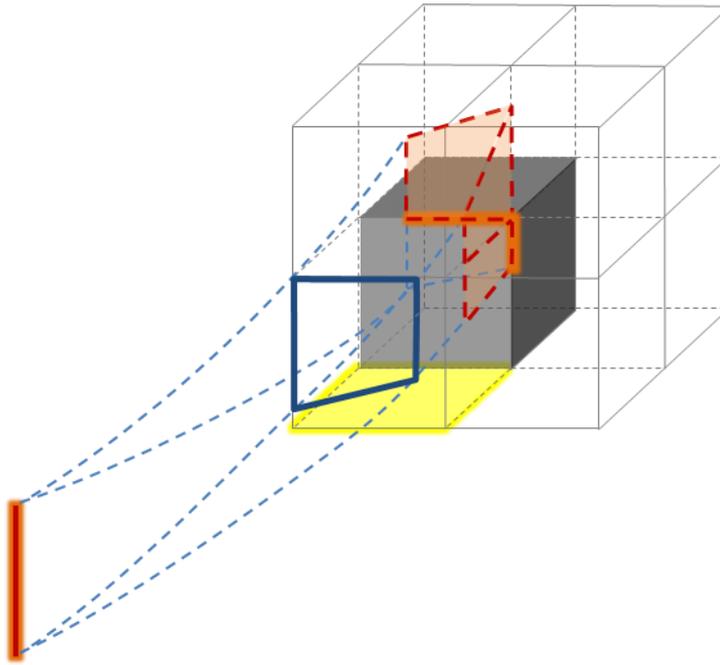
The identification of intervals is done by checking the projection of the visibility lines from the root onto each face within a 2D grid. This idea is visualized in Figure 4-1. The visibility lines are determined space-wise by the range of vessel headings and time-wise by the range of vessel speeds. In the left figure the space root is the orange glowing vertical line. The time range of this root is 12 hours and the headings range over  $\pi/2$  radians. The maximum vessel speed from time  $t=0$  and the minimum vessel speed from time  $t=12$  hours determine the points in space-time that can be attained by the vessel in this grid cell. For this root the set of points on the faces that are not adjacent to the root edge form the intervals for this grid. The same idea is applied to the time root, as depicted in the right figure.



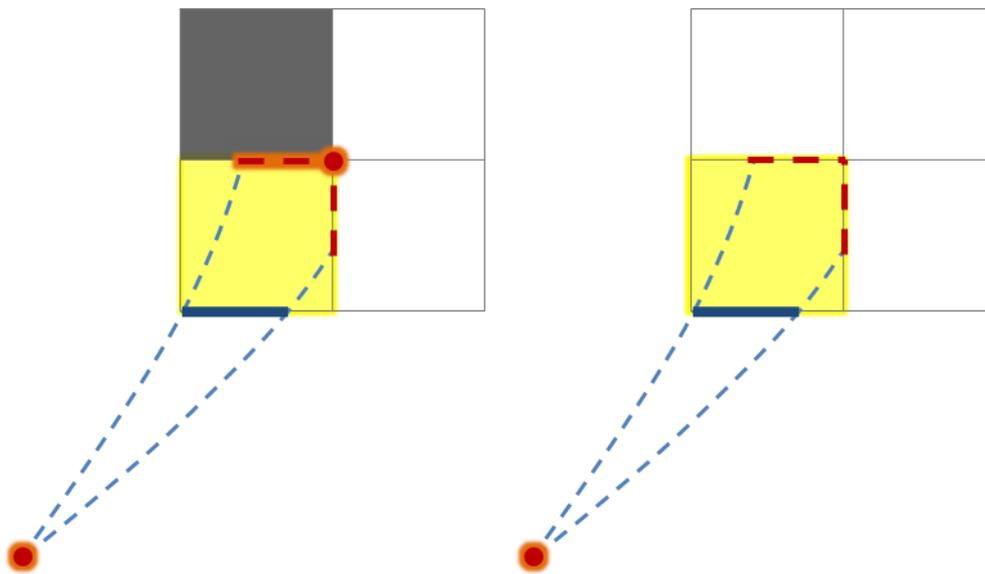
**Figure 4-1.** Space root (left) and time root (right). For the space root its corresponding intervals are visualized by red (dotted) lines for heading range  $[0, \pi/2]$  and speed range  $[v_{min}, v_{max}]$ .

**Definition 3:** A search node  $T = (r, I)$  or  $T = (r, r', a, v)$  is a tuple where  $r$  is a root,  $I$  is an interval,  $r'$  is a successive root and  $a$  is the range of vessel headings and  $v$  is the range of sailing speeds corresponding to root  $r'$ . Each point  $p \in I$  or  $p \in r'$  is visible from  $r$ . The root  $r$  always contains the most recent turning point on the shortest path from start to any point on either  $I$  or  $r'$ .

The algorithm starts with adding three tuples to the open set of search nodes to be investigated:  $T(\text{vessel position}, r_0, a_{1/2/3})$ . The actual vessel position at time of route search is the start space root. The next space root  $r_0$  is located at the grid point that is closest to the Great circle line from vessel position to destination. Headings  $a_1, a_2$  and  $a_3$  are the headings into each of the three grid cells adjacent to  $r_0$  except the grid cell that contains the actual vessel position. The cost of each start search node is the sum of the sailing cost from origin to  $r_0$  and the heuristic cost estimate based on the Great circle distance and the economic sailing speed to the destination.



**Figure 4-2.** Exploration of new tuples. Visibility lines from the root are projected on the faces of the yellow 2D grid base to find new intervals and roots.



**Figure 4-3.** Spatial exploration of new tuples for two time layers.

New search nodes are found by exploring the first search node in the open set. This search node exploration for intervals is depicted in Figure 4-2 and Figure 4-3. In Figure 4-2 the root is presented by the orange glowing vertical line and the interval by the thick blue lines. Given this root and interval, a range of sailing speeds  $v_{min}$  and  $v_{max}$  and a range of vessel headings  $a_{min}$  and  $a_{max}$  can be defined. These vessel speeds and headings are depicted by the blue dotted lines and can be regarded as visibility lines from the root through the interval. Figure 4-3 is a 2D representation of this example. It shows the top views of the two time layers.

The successor intervals are determined for the traversable grid cells above the yellow highlighted base. The successors are the projections of the visibility lines onto the faces of these grid cells. Note that a split is made for different time blocks. Each intermediate point in a discovered interval should reach any point on the original root node via a taut local path.

Whenever an interval is part of a face of a blocked grid cell, it is not traversable and should be omitted. In this situation the interval edges that coincide with grid edges form new roots in the search. Such a new root contains a turning point in any path via this root. In Figure 4-2 the new roots are the two orange glowing red dotted lines that border the blocked grey colored grid cell. For a space root its range of ship speeds is determined by the visibility lines from the original root. The range of vessel headings is the 'shadowed' area by the blocked cell, i.e. the non-visible headings from the root. The vessel heading and speed range for the new time root is determined exactly opposite. For this example weather information for each of the eight grid cells should be known in order to identify successor intervals and roots.

The *Anya* algorithm forms new search nodes for tuples of root and interval combination only. For observable intervals the first tuple type in Definition 3 is analogue to this approach. But in *Anya* intervals are also made for line segments that border blocked grid cells. For such cases a new root node is defined that has a pointer to the original root. In our 3D algorithm we treat such cases differently by defining only new roots in combination with a vessel heading range. No intervals on successive faces are defined. This is the second tuple type in Definition 3. This is done because in our algorithm the search space is explored cell by cell and successor intervals are part of successive cells.

For each combination of root and successor interval or root, a search node is added to the open set. The open set is kept sorted based on the estimated path cost of each search node. For this purpose the point on the interval or root is selected that gives the lowest path cost from root to destination via this interval or root. Therefore this cost estimate provides a lower bound on the cost of the final path.

A heuristic is used to determine the minimum path cost to the destination through the unexplored environment. This is based on the extracted taut path from origin to destination via the successor interval or root under investigation. Taut means sailing Great circle legs all route points with the economic sailing speed according to the cost function. Path extraction and route cost calculation are described in detail in the following sections.

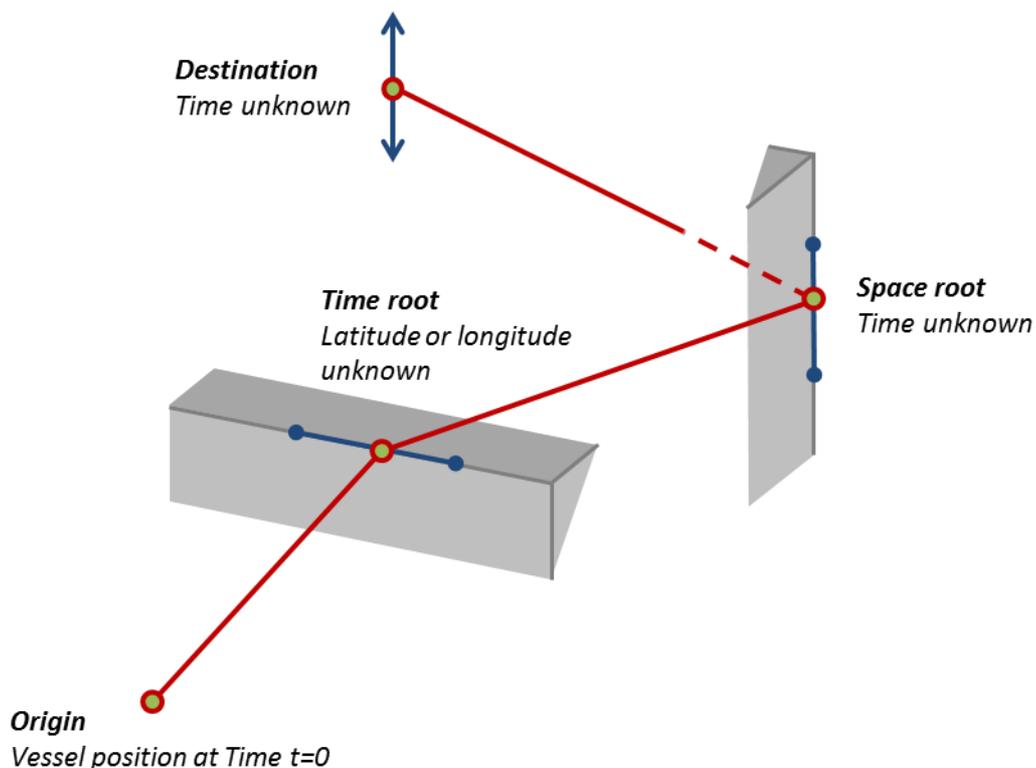
Each iteration the search node with lowest cost estimate is investigated for new tuples. When the destination is visible by a new root or by any point on an interval except its endpoints  $a$  or  $b$  then the solution path is found. The sailing route is found by following the root parental pointers to the actual vessel position. The taut path along these roots is the solution sailing route for the vessel. Finding this taut path is explained in the next section.

### 4.3. Extracting a path

Determining a path is not as straightforward as for the *Anya* algorithm. This is due to the fact that roots in *Anya* are nodes and roots in our 3D version are line segments. Finding a shortest path via nodes is straightforward, but via line segments requires a different approach. In addition to that the taut path over the line segments might slightly shift each iteration a new root is added. This phenomenon is captured in our algorithm.

A new root is either fixed in time (time root) or fixed in space (space root). The destination is only fixed in space. The actual vessel position at the start of the search is fixed in both time and space. This enables us to make the path taut and determine the path cost. Therefore the search should start at the vessel position and made taut towards the destination. The principles are depicted in Figure 4-4.

A space root fixates 2 degrees of freedom (latitude and longitude) while a time root only fixates 1 degree of freedom (time). As a result space roots are always parallel, while time roots can be perpendicular. This principle enables to find the optimum latitude and longitude for a time root. We define a shortest route plane between each pair of consecutive space roots. The intersection of such a plane with a time root is a route point. When there is no intersection, the point on the time root closest to the plane is selected as route point.



**Figure 4-4.** Principles of extracting a path.

When multiple time roots are present between two space roots, this process of finding intersections should be performed iteratively. For each iteration (intersection calculation) the route plane is split up in two parts and the previously found route should be optimized again. This is because each new time root can modify the presently discovered route.

For the best time intersection for a space root a slightly different approach is used. Line segments of constant speed are drawn between each pair of consecutive route points (That is: Origin and all the time roots for which a location is discovered) via the space roots. Then for each set of consecutive space roots between two route points, iteratively the best route point per space root is determined. Also for this process, after each iteration the previously discovered route should be optimized again.

There is no fixed arrival time at destination, since the destination is also a space root. Therefore the times for the final set of consecutive space roots are determined based on the arrival times with lowest cost. Also this process requires local path optimization to the previous time root for each newly investigated space root towards the destination.

When the path has to be determined for an intermediate interval this interval can be regarded as a time root initially to determine the intersection point with the interval plane. Once this is determined it can be treated as a space root to find the optimum arrival time at this interval plane.

The principles of this path extraction algorithm are presented in Table 4-1. In lines 2-6 all path roots and the origin are added to a set  $A$ , which forms the basis for the path. Then this path is optimized starting from origin towards the destination. When a time root is followed by a space root, the locations for the previous consecutive time roots can be determined (lines 9-11). This is done iteratively for each successive time root (lines 21-24; 30). Each iteration the discovered path should be locally optimized (lines 25-29).

After the locations for this set of time roots are determined, the times for all consecutive space roots that preceded these time roots can be determined (lines 12-14). A similar iterative process is followed to stepwise calculate the arrival times per space root (lines 32-35; 41). Each iteration a local optimization of the discovered arrival times is performed (lines 36-39).

If applicable, the times for the final set of space roots is determined based stepwise toward the destination (lines 15-17). This can be done based on optimum sailing speed, since there is no fixed arrival time at destination (lines 43-47; 52). Local optimizations are performed after each iteration (lines 48-51). The algorithm stops with returning the solution path (line 18).

**ALGORITHM: Extracting path for specific root**

```
1. EXTRACT PATH(origin, root)
2. p := {root}
3. while p ≠ origin
4.     A := A U p
5.     p := p.parentRoot
6. A := A U origin // A(0)=origin
7. i := 1
8. do
9.     if A(i).type = timeRoot and A(i+1).type = spaceRoot
10.        x := index set A last spaceRoot before A(i)
11.        FIND LOCATIONS(x, i)
12.        if x ≠ 0 then
13.            y := index set A last timeRoot before A(x)
14.            FIND TIMES(y, x)
15.        if A(i) = root then
16.            z := index set A last timeRoot before A(i)
17.            FIND TIMES DESTINATION(z, i)
18.            return A // Algorithm stops
19.        i := i+1

20. FIND LOCATIONS(x, i) // Find locations for time roots
21. a := x+1
22. while a ≠ i+1
23.     A(a).latlon := point on line A(a) with minimum
24.                    distance to plane [A(a-1), A(i+1)]
25.     b := a-1
26.     while b ≠ x
27.         A(b).latlon := point on line A(b) with minimum
28.                        distance to plane [A(x), A(b+1)]
29.         b := b-1
30.     a := a+1

31. FIND TIMES(y, x) // Find times for space roots
32. path := {A(y), A(y+1), ... , A(x), A(x+1)}
32. a := y
33. while a ≠ x+1
34.     A(a).time := point on line A(a) with minimum cost
35.                 to sub path {A(a-1), ... , A(x+1)}
```

```

36.     b := a-1
37.     while b ≠ y
38.         A(b).time := point on line A(b) with min. cost
39.             to sub path [A(y), ... , A(b+1)]
40.         b := b-1
41.     a := a+1

42. FIND TIMES DESTINATION(z, i) //Find times for space roots
43. path := {A(z), A(z+1), ... , A(i)}
44. a := z+1
45. while a ≠ i
46.     A(a).time := point on line A(a) with minimum cost
47.         to sub path {A(a-1), A(a), ... , A(i)}
48.     b := a-1
48.     while b ≠ z
49.         A(b).time := point online A(b) with min. cost
50.             to sub path {A(z), ... , A(b+1)}
51.         b := b-1
52.     a := a+1

```

**Table 4-1.** Path extraction for specific root

#### 4.4. Path cost

The path cost is calculated based on an objective function that includes vessel costs, fuel costs and late arrival penalty. The vessel costs and late arrival penalty are a function of sailing time. The fuel cost is a function of time and engine settings. The final leg  $\{k-1, k\}$  is determined according to a heuristic that connects the lowest cost point on root or interval with the destination point following the Great circle path at the cost-optimal sailing speed.

Given a path  $\pi = \{p_1, \dots, p_k\}$  with  $p_1$  being the vessel position and  $p_k$  being the destination, the cost for this path can be calculated as follows:

$$C(\pi) = \sum_{i=1}^{k-1} (C_{Fixed} + C_{Fuel,i}) (T_{i+1} - T_i) + C_{Destination} (T_k - T_s)$$

where,

$C_{Fixed}$	= Fixed ship cost per unit time (crew, provisions)
$C_{Fuel}(MCR(t))$	= Fuel cost per unit time (function of engine setting)
$C_{Destination}$	= Delay penalty per unit time (late arrival)
$T_i$	= Time at $p_i$
$T_k$	= Arrival time at destination
$T_s$	= Scheduled arrival time at destination

## 4.5. Correctness and optimality

In a stepwise approach the correctness and optimality of the algorithm is shown. First it is shown that if the optimal path to an intermediate or corner point exists, this path appears in the search tree (Theorem 1). Then if a grid cell is being investigated that contains the destination, the optimal any-angle path from vessel position to destination is found (Theorem 2). This proof is analogue to the proof sketched for *Anya* in (Harabor, 2013).

**Theorem 1:** *For any point  $p$  that appears on a grid cell face or edge, there exists a search node in the tree that corresponds to the optimal path from vessel position to  $p$  if such a path exists.*

**Proof:** Consider an optimal path  $\pi = \{p_1, \dots, p_k\}$  from  $p_1$  to  $p_k$ . Point  $p_{k-1}$  is a point on the previous root before  $p_k$ . By induction, there is a search node in the search tree that represents the optimal path from  $p_1$  to the root that contains  $p_{k-1}$ . When this search node is expanded as explained in section 4.2, either a new interval or root is discovered that contains  $p_k$ . When  $p_k$  is in an interval it will use the original root for forming a new search node, since the entire interval is visible from the original root. When  $p_k$  is on a root, it will be a new corner for the future path. Such a root will either search the *space shadow* from the original root in case of a space root or the *time shadow* in case of a time root. Via the path extraction algorithm the path is made taut again from  $p_1$  to  $p_k$ . Since this point  $p_k$  is obtained via visibility lines from the previous root, this taut path is optimal as well.

**Theorem 2:** *The first search node that finds the destination corresponds to the optimal path from vessel position to destination.*

**Proof:** The cost of a search node is a lower bound on any path cost to the destination via this interval or root. This is because the cost is based on the point  $p$  on interval or root that results in the lowest path cost. A path via any other point than  $p$  on this interval or root is more expensive. When this search node is expanded, the costs via the successor intervals and roots are equal or higher than the cost estimation for actual search node. Therefore the path costs are always lower bounded and monotonically increasing, while the search progresses. Path cost of any search node that is part of a sub-optimal path will eventually exceed the optimal path cost. On the other hand, path cost of search nodes that guide to the optimal path will never exceed this optimum.

## 4.6. Time complexity

Since runtime of the algorithm is important for the efficiency of a Monte Carlo Simulation, its time complexity is also investigated. The algorithm basically is an implementation of  $A^*$  which is a heuristic graph search algorithm. In  $A^*$  the time complexity depends on the method of searching the environment, i.e. the complexity of the search space.

$A^*$  starts with an empty closed set and open set that contains the start state of the ship. Then a while-loop runs in which iteratively the search node with lowest estimated path cost from vessel position to destination is explored. Each explored node is added to the closed set; each newly discovered node is added to the open set. In this way a tree of route points (latitude, longitude, time) is being built. This loop terminates when a corner or edge of the destination grid cell is reached.

The time complexity of  $A^*$  is determined by the while-loop that is executed  $O(|V|)$  times and the inner-loop search in each iteration for the lowest cost node in the open set, which also runs  $O(|V|)$  times when implemented as an array. This could be improved to  $O(\log|V|)$  depending on choice of data structure that is used for the open set. For a tree the neighbour exploration of a search node is executed  $O(|E|)$  times over the entire algorithm. In this context,  $V$  is the number of search nodes in the tree and  $E$  is the number of edges between the search nodes. The upper bound for runtime now becomes:

$$O(|V|^2 + |E|) = O(|V|^2)$$

The runtime of the algorithm is linear in the length of the path if the shortest path to the destination does not contain any obstacles. But in worst case when the search space is unbounded the search tree has exponential growth. If we avoid pruning the search tree, e.g. to avoid risking loss of optimality by cutting off branches that contain the optimal route, then the upper bound on the search space becomes:

$$|V| = O(b^d)$$

where,

- $b$  = branching factor (average nr of successors per search node)
- $d$  = shortest path length (number of traversed grid cells along the optimal path)

The branching factor is largest when obstacles are present in the environment. Then not only intervals could be added as search nodes, but space and time roots as well. Per 12 hour time block a combination of up to 2 intervals, 4 space roots and 2 time roots might be added which will be largest in case of a chessboard pattern of blocked grids. The amount of time blocks that are traversed in a grid cell obviously depends on the sailing speed. When the destination can't be reached from the origin and sailing time is not bounded, this implementation will not terminate. This could occur when the destination is in a blocked cell, e.g. due to a land obstacle or very stringent weather criteria.

In order to preserve optimality, the test experiments will be performed for the optimal variant of the algorithm at the risk of exponential growth of the search space and runtime. In this version all discovered search nodes will be included in

further search. When weather or land obstacles are present in a problem instant this might result in overlapping points in the search space, i.e. points that could be reached via multiple paths.

The runtime can be reduced by bounding the growth of the search space. This can be accomplished by discretizing the search space to the set of faces and edges of the grid and only store the optimal path to the each discovered face and edge. The parental root and path cost for a face or edge is updated when a lower cost path is found. In such an implementation, during each node exploration the closed set should be checked if it contains neighbour nodes. When the search space is discretized and time and space are bounded, the amount of search nodes becomes finite.

The drawback of such a discretization is loss of optimality, which could reduce the accuracy of the algorithm solution. The exponential behaviour depends on the number of obstacles that is present in the search space. Experiments will be performed to analyse the impact of obstacles on runtime and to test if discretization should be further investigated.

## 5. EXPERIMENTS

The actually in SafeTrans implemented *Dijkstra* algorithm and the proposed *Anya 3D* re-route algorithm as described in the previous chapter have been implemented in a test environment for testing purposes of their route solutions and runtime performance. The nearly optimal sample-based *RRT\** algorithm has been implemented as well and serves for comparison of the generated routes by both algorithms. *RRT\** has been introduced in 2.2 and returns nearly optimal routes at high runtime costs.

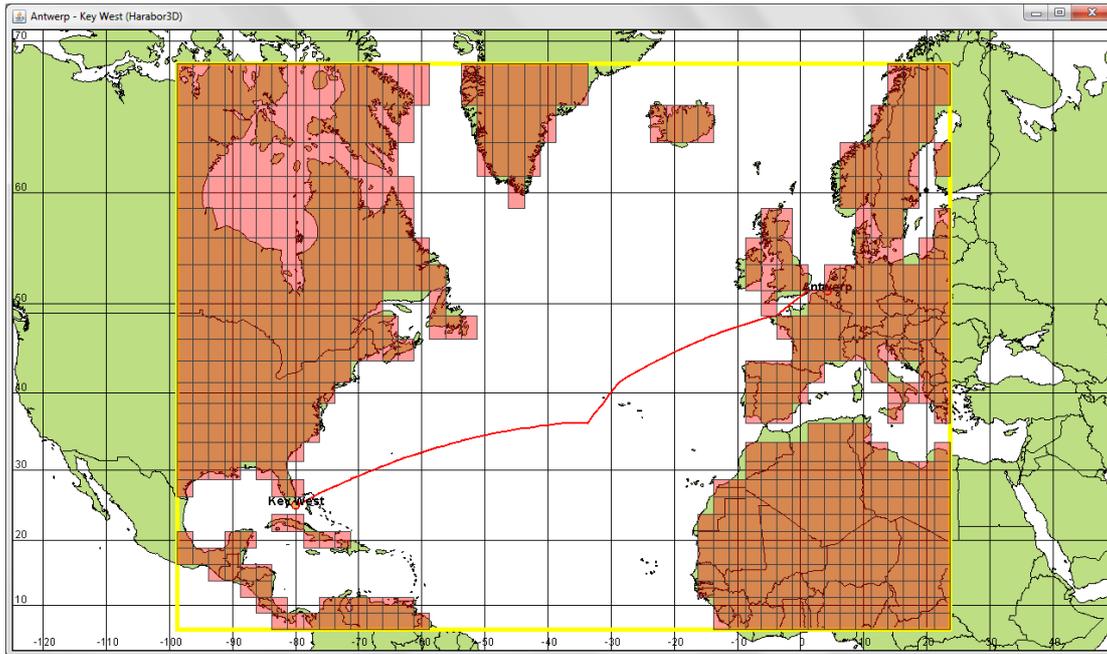
The algorithms have been tested for a set of experiments. These experiments are problem instants that have been selected from the SafeTrans benchmark database and consist of 4 different routes, each with 2 different weather scenarios. For each experiment *Dijkstra* and *RRT\** are run for the default speed settings while *Anya 3D* returns routes that are either optimized for time, speed or for a combination of time and speed including a late arrival penalty. The solutions are compared for route length, sailing time, fuel consumption and algorithm runtime.

In the first experiment routes are generated for environments without considering any weather restrictions. This should return the shortest route over sea for each algorithm and can be easily checked on accuracy of distance. In a second experiment the algorithms will be tested for weather routing in the 8 weather scenarios. A third experiment is carried out to test the *Dijkstra* and *Anya 3D* route solutions in changing weather. This is done by simulating the voyage while updating the weather on a daily basis.

All runs are executed on a 2.9 GHz Intel Core i5 with 8 GB of RAM memory. The experiments are all run without updating visualizations at runtime and with CPU-usage by other applications as low as possible.

### 5.1. Experimental setup

In order to perform all experiments a test environment has been written in Java code. For geospatial visualization and analyses of the generated route solutions the open source package *BBN OpenMap* library has been used, which is also written in Java and is used in the SafeTrans software. Figure 5-1 presents an example screenshot of this environment for one of the problem instants. The search boundary is indicated by the yellow square. This is a 20 degrees margin around the ports of departure and destination. The red line shows the solution returned by the actual re-route algorithm for an example problem instant. In this figure only the static forbidden land areas are visualized (pink coloured cells). The time-dependent dynamic forbidden areas are not shown, but the solution (red line) clearly shows that the algorithm has navigated around these areas. The green land mass is only informative to the user, but not used by the algorithms.



**Figure 5-1.** Test environment for re-route algorithm. The red line is the solution by the actual re-route algorithm for an example problem instant.

### 5.1.1. Description of the simulation

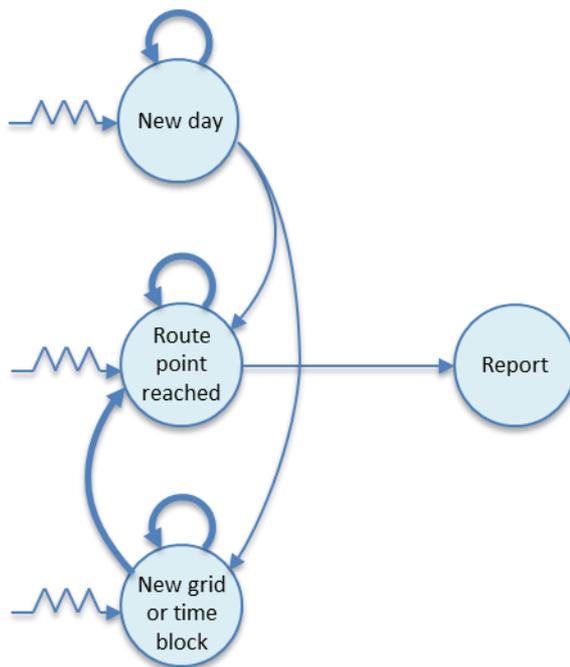
For the third experiment a discrete event simulation has been set up to simulate sailing of the route solutions in weather forecasts that change over time. In each simulation run a route is generated prior to departure by the *Dijkstra* and *Anya 3D* algorithms. During simulation of this voyage, the route is kept fixed while the weather is updated daily. In this way the quality of this route solution can be determined for changing weather.

The discrete event simulation is set up based on the method described in (Law, 2007). The time-advance mechanism used in this discrete event simulation is *next-event based*. In this approach the simulation is run by consecutively handling events at discretized points in time until the simulation is finished. During each event handling the state of the simulation might change and new events might be scheduled if required. When an event is finished the simulation jumps to the next event in time. Therefore event times should be maintained in a list.

The events for this simulation are presented in Table 2-1 and the event graph is depicted in Figure 5-2. In the event graph a thin arrow indicates that a new event could be scheduled immediately, thus without any intervening time. A thick arrow means that an event could possibly be scheduled later in time. A thin jagged arrow indicates an event schedule during initialization of the simulation.

Nr	Event type	Description
0	New day	Start of new day
1	Route point reached	Ship has reached a route point of the route plan
2	New grid or time block	Ship enters new grid cell or 12h time block
3	Report	End of simulation; report printed

**Table 5-1.** List of events.



**Figure 5-2.** Event graph.

At the start of the simulation the main program invokes the initialization routine. In the initialization phase the events *New day*, *Route point reached* and *New grid or time block* are scheduled, the route plan is generated, the simulation clock is set to zero and the system state and statistical counters are initialized.

After initialization the timing routine checks in the event list which event should be carried out. Each event except the *Report* event starts with an update of the vessel position and the sailed route. At each *New day* event the weather conditions are updated. At departure this immediately leads to a *Route point reached* and *New grid or time block* event since the port of departure is part of the route plan and a new time block is entered by starting the simulation. A *Route point reached* event might change the engine MCR settings since this is route leg dependent. This type of event may reschedule itself when the vessel speed is non-zero. When the route point that is reached equals the port of destination the simulation is finished and a *Report* event is scheduled at the same simulation time.

During a *New grid or time block* event the environmental conditions (wind speed, significant wave height) and fuel rate are updated and the vessel speed is recalculated. Fuel rate and vessel speed are also updated for each *Route point reached* event, since the engine settings could be changed for the next route leg.

After each event an update is made for the vessel position and the counters:

- Total sailing distance
- Total sailing time
- Total fuel consumption

### 5.1.2. Calculation of vessel speed

The ship speed can be calculated for a given ship calm water resistance curve, environmental sea state and engine power output (MCR setting). This is the speed where the thrust force by the propeller equals the sum of the resistance forces in the ship's longitudinal direction. It should be noticed that only forces in longitudinal direction are considered. All transversal forces and moments in the horizontal plane are ignored in our experiment. Furthermore current forces, wind and wave direction and rudder forces are neglected. The method is similar to the implemented method in SafeTrans as described in the SafeTrans 5 theory manual (MARIN, 2011).

$$F_{propeller\ thrust} = R_{calm\ water} + R_{wave\ drift} + R_{wind}$$

where,

$$F_{propeller\ thrust}(v_{ship}, MCR) = \text{Effective propeller thrust as function of ship speed and MCR setting}$$

$$R_{calm\ water}(v_{ship}) = \text{Calm water resistance at sailing draft}$$

$$R_{wave\ drift}(v_{ship}, H_s, T_p) = \text{Wave drift force as function of ship speed, significant wave height and peak period}$$

$$R_{wind}(v_{wind}) = \text{Wind force}$$

For each ship speed ( $v_{ship} = \{0, 1, 2, \dots, v_{ship, Max\ calm\ water\ curve}\}$ ) the 3 resistance components and the propeller thrust are discretized and stored in arrays. The propeller thrust is assumed to be linear with the ship speed:

$$F_{propeller\ thrust}(v_{ship}, MCR) = MCR^{1.1} \left( T_{BP} + \frac{(R_{service} - T_{BP})}{v_{BP}} v_{ship} \right)$$

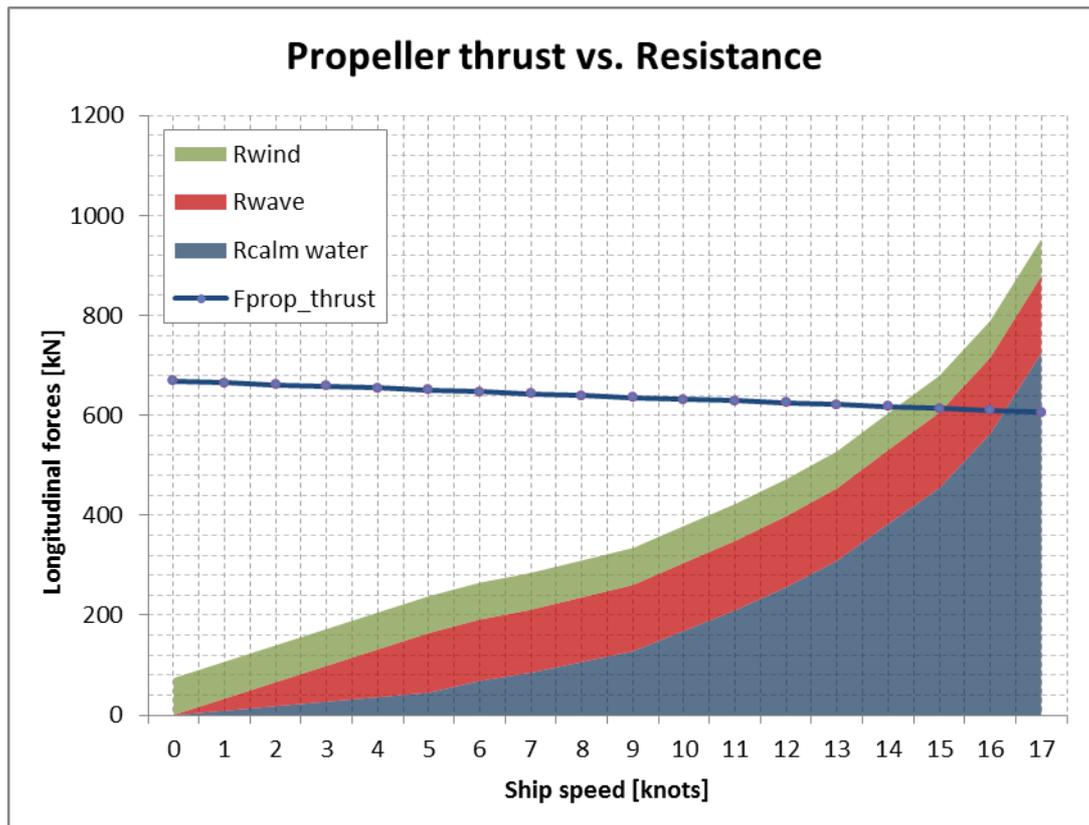


Figure 5-3. Propeller thrust versus resistance forces.

In this formula the ship's bollard pull  $T_{BP}$ , the ship's service speed  $v_{BP}$  and ship's calm water resistance at service speed  $R_{service}$  are fixed values and case specific.

The calm water resistance curve is fixed during each simulation, assuming no change in ship's draft during the voyage.

The wave drift forces in the ship's longitudinal direction are pre-computed and cached in a database for a series of combinations of ship speed  $v_{ship}$ , significant wave height  $H_s$  and peak period  $T_p$ . During the simulation, the actual wave drift force per ship speed is obtained by linear interpolation between the four nearest  $H_s$ - $T_p$  combinations.

The wind force depends on the variable wind speed  $v_{wind}$ . The other parameters density of air  $\rho_{air}$ , frontal cross sectional wind area of the ship including cargo  $A_{front\ ship}$  and the non-dimensional frontal wind coefficient  $C_{front\ ship}$  are kept fixed during each simulation.

$$R_{wind}(v_{wind}) = \frac{1}{2} \rho_{air} v_{wind}^2 A_{front\ ship} C_{front\ ship}$$

The equilibrium speed is calculated in a loop, starting at zero knots with steps of 1 knot until the total resistance force exceeds the thrust force. Then by linear interpolation the ship speed that gives equilibrium in longitudinal forces is determined.

### 5.1.3. Calculation of fuel consumption

Between each set of two events the consumed fuel for that leg is calculated in metric tonnes.

$$\Delta Fuel(MCR, t) = Fuel Rate * \Delta t$$

where,

$$\begin{aligned} Fuel Rate (MCR) &= \text{Fuel rate in [ton/hr]} \\ \Delta t &= \text{Time between two events in [hr]} \end{aligned}$$

The fuel rate depends on the engine settings and is calculated based on the default method described in the SafeTrans 5 theory manual (MARIN, 2011). Above the default economic MCR setting of 85% the fuel consumption increases.

$$\begin{aligned} 0 \leq MCR \leq 0.85: & \quad Fuel Rate (MCR) = P_b(40MCR + 131)/10^6 \\ 0.85 < MCR \leq 1.0: & \quad Fuel Rate (MCR) = P_b(199 - 40MCR + 131)/10^6 \end{aligned}$$

where,

$$\begin{aligned} P_b &= \text{Brake power in [kW]} \\ MCR &= \text{Power setting between 0 and 1 (respectively 0\% and 100\%)} \end{aligned}$$

### 5.1.4. Dijkstra and Anya 3D

The actual in SafeTrans implemented algorithm has been outlined in 3.7. This is a *Dijkstra* route search that works similar as  $A^*$  except that no heuristic is used to steer the search towards the destination. For an outline of  $A^*$  see Table 2-1. Each node in the grid has 16 neighbouring nodes in the same “day” layer and 1 time node in a next day layer that allows further search after waiting for one day. In total 10 day layers are implemented in the search grid. A basic version of the *Anya 3D* algorithm as described in the previous chapter is also implemented.

### 5.1.5. Rapidly-Exploring Random Tree: RRT\*

$RRT^*$  is used for comparison of the *Dijkstra* and *Anya 3D* solutions for the first and the second experiment.  $RRT^*$  is a sampling-based approach and has been introduced in 2.2. The algorithm has probabilistic completeness, i.e. provided a solution exists, the probability of finding a solution becomes 1 when the number of samples approaches infinity. As a result this algorithm can find very accurate solutions, but the main drawback is its runtime which increases linearly with the number of samples. The algorithm that is implemented in the test environment is given in Table 5-2 and is an implementation of the algorithm introduced by Karaman (Karaman, 2011).

The algorithm works by incrementally growing a tree towards randomly sampled points in the search space. The tree starts with the departure port. Each iteration a random location is sampled from the free space (line 6). For this location the nearest point in the tree is found based on lowest route costs (line 7) and a new potential tree node is found from this nearest point in the direction of the sampled point (line 8). If the edge to this new point is in the free space then it is added to the tree. Then an attempt is made to find a lower cost path from the origin to this new node by looping over all nodes in the tree (lines 14-18). The node that gives the lowest cost path for the new node is set as parent (line 19).

Each newly added node might also introduce lower cost paths to existing nodes in the tree. This is done by checking near nodes within a specified radius (line 10) for a lower cost path via the new node. If local optimizations are found the tree is rewired (lines 20-24). The search radius shrinks logarithmically with the size of the tree.

The loop runs for 100.000 iterations. When all samples are explored all nodes in the destination grid cell are checked and the lowest cost node is returned as solution (lines 25-29).

In this implementation the functions NEAREST and NEAR utilize brute-force to find near nodes in the tree. This requires  $O(n)$  time and  $O(1)$  space, which results in large computation time when the number of samples increases. Since the purpose of using *RRT\** in this study is for comparison of the accuracy of the solutions rather than computation time, no effort is put in improving the speed of this search procedure.

#### ALGORITHM: RRT\*

```

1.  FIND_ROUTE(origin, destination, step)
2.  sailingroute := ∅
3.  V := {origin}
4.  E := ∅
5.  for i=1,...,n do
6.    x_random := SAMPLE_FREE;
7.    x_nearest := NEAREST(G(V,E), x_random)
8.    x_new := STEER(x_near, x_random)
9.    if OBSTACLE_FREE(x_nearest, x_new) then
10.     X_near := NEAR(G(V,E), x_new, radius(V))
11.     V := V ∪ x_new
12.     x_min := x_nearest
13.     c_min := cost(nearest) + cost(L(x_nearest, x_new))
14.     for each x ∈ X near do
15.       if OBSTACLE_FREE(x, x_new) AND
16.         cost(x) + cost(leg(x, x_new)) < c_min then

```

```

17.         x_min := x
18.         c_min := cost(n) + cost(L(x, x_new))
19.     E := E ∩ {(x_min, x_new)}
20.     for each x ∈ X near do
21.         if OBSTACLE_FREE(x_new, x) AND
22.         cost(x_new) + cost(leg(x_new, x)) < c_min then
23.             x_parent := parent(x)
24.             E := (E \ {(x_parent, x)}) ∩ {(x_new, x)}
25. return RECONSTRUCT_PATH(destination, x_closest)

26. RECONSTRUCT_PATH(rp1, rp2)
27. Sailingroute := sailingroute ∩ leg(rp1, rp2)
28. if rp2 ≠ origin then
29.     RECONSTRUCT_PATH(rp2, parent(rp2))

```

**Table 5-2.** RRT\*

## 5.2. Problem instants

In 2012 a thorough study utilizing the SafeTrans benchmark cases was carried out to analyse the CDM performance in comparison to real seafarers. This benchmark consists of different combinations of ship types and sailing routes and is used for testing new SafeTrans releases. From this study 8 problem instants were selected for which the route choice by CDM showed typical behaviour compared to the real captains. This set of problem instants is presented in Table 5-3.

For each of the four voyages two weather scenarios have been selected. The first two voyages are self-propelled transportations by a heavy lift vessel. These weather scenarios also contain hurricane forecast data. The third voyage is a tug-tow combination, characterized by the lower ship speed. The last voyage is an LNG carrier.

Nr	From	To	Distance	V <sub>ship</sub>	Start date	Criteria	
						H <sub>sign.</sub>	V <sub>wind</sub>
			(nm)	(kn)		(m)	(kn)
1	Busan (S. Korea)	Darwin (Australia)	2928	15.0	28-07-1996	4.5	45 kn
2					22-12-1996		
3	Antwerp (Belgium)	Key West (USA)	4120	16.0	31-12-2997	8.0	50 kn
4					20-12-2000		
5	Rio de Jan. (Brazil)	Offshore Angola	3221	8.0	24-05-1996	4.5	28 kn
6					09-09-2000		
7	Sakhalin (Russia)	San Diego (USA)	4341	19.0	10-11-2000	5.5	45 kn
8					08-12-2002		

**Table 5-3.** Selected problem instants.

### 5.3. Route generation without weather

In a first set of experiments routes are generated without considering any weather restrictions. This enables to compare the returned routes to the optimal Great-circle distance via land grid corners. For *Dijkstra* and *Anya 3D* each simulation is run 20 times to average algorithm runtime. *RRT\** is run only once since this algorithm is only used for comparison of route length, sailing time and fuel consumption. The input settings and algorithm solutions for the four routes are presented in Table 5-4 to Table 5-11. The tables compare route distance, sailing time, fuel consumption and algorithm runtime for each run.

For each voyage, the first table shows the input values for the ship's operational speed range with incremental steps of 1 knot. In the actual simulations, the applied speed ranges follow from the MCR settings and have been set to [25% MCR, 100% MCR] for all route generations. Per ship speed the corresponding calm water resistance is given. The required propulsion power to meet that ship's calm water resistance is calculated and expressed as the engine's MCR setting. For calculation of the fuel consumption the fuel rate corresponding to that MCR setting is presented as well. In the simulations linear interpolation is used for calculation of values at intermediate ship speeds.

The costs per ship speed are presented in the four final columns:  $K_{fixed}$ ,  $K_{fuel}$ ,  $K_{delay}$  and  $K_{total}$ .  $K_{fixed}$  is the time dependent fixed cost and represents expenses regardless sailing, such as crew, provisions, insurance, depreciation and overhead. It depends linearly on sailing time and is normalized for the required sailing time for the shortest route at the ship's default speed. The shortest sailing time per ship speed is given in the column left of  $K_{fixed}$ . For fuel consumption a similar approach for cost calculation is used. This is normalized for the required amount of fuel for the shortest route while sailing at the ship's default speed. The delay penalty is normalized for a delay of 5 days with respect to the expected arrival time at departure.

Search nodes and sample points of respectively *Dijkstra* and *RRT\** are valued for arrival time at that node's location based on the shortest discovered path. The cost of a search node in *Anya 3D* is the sum of the fixed and fuel cost to reach the lowest cost point on its root or interval and a heuristic cost to reach the destination from that lowest cost point. As a heuristic the fixed, fuel and delay costs to the destination are calculated for the economical speed. This economical speed depends on the simulation settings and corresponds to the pink coloured cells. E.g. for the route Busan to Darwin the economical speed is 13 knots when optimizing the route for fuel, 17 knots when optimizing for time and 15 knots when giving equal weight to time, fuel and delay penalty.

For all four routes *Anya 3D* returns the optimal solution. Near optimal solutions within 5 nm of the optimum are found by *RRT\**. This is accomplished by taking 100.000 samples. These route lengths are up to 2 percent shorter compared to the *Dijkstra* routes. Some routes show land crossings. This typically occurs in the first route from Busan to Darwin, which contains many islands that are considered sea area in the weather database. The *Dijkstra* algorithm only

considers grid nodes. Edges are not checked for any land crossing, while the final route solution follows a path of edges. In *Anya 3D* the interior points of the grid are also considered for the lowest cost path to each root or interval.

To avoid land crossings in *Anya 3D* it is proposed to divide grid cells that contain land masses into smaller sections and only include sea sections in the search space. This division should not necessarily be restricted to rectangular shaped sections; the same search principle could be applied to any convex subdivision. A drawback of this approach is that the size and complexity of the search space increases, which results in more complex algorithm coding and longer search queries.

The fuel consumption for *Dijkstra*, *RRT\** and *Anya 3D* optimized for time, fuel and delay shows a linear relationship with the sailing distance, since the obtained vessel speed is equal for these three cases. These results confirm equal calculation of fuel consumption in the algorithms. Optimizing the route for fuel shows fuel savings between 5 and 11 percent per voyage. This is only depending on the sailing speed, since no weather routing is involved and therefore the shortest path is always returned by *Anya 3D*. There is a non-linear relation between route costs and sailing speed. As a result fuel saving could considerably increase sailing time. For the route Busan to Darwin 10.6% fuel is saved compared to *Dijkstra*, whereas sailing time increases with 13.1%. For the other three cases there is a larger difference between fuel savings and increased sailing time. For Rio de Janeiro to Angola this ratio is worst due to the relatively slow fuel-optimized sailing speed of the tug-tow combination. In this case 4.5% fuel is saved compared to *Dijkstra*, whereas sailing time increases with 31.6%.

Optimization of sailing time shows larger differences in the balance between decrease of sailing time and increase of fuel consumption. For the routes Antwerp to Key West and Sakhalin to San Diego sailing time drops at a faster rate than fuel consumption rises. For example Antwerp to Key West shows 6.9% decrease of sailing time compared to 5.8% increase of fuel. But this is less balanced for the other two cases. The worst result is on the route Busan to Darwin, for which sailing time decreases with 13.5% at the cost of 47.2% more fuel consumption. For each individual case a trade-off between fuel and sailing time can be made to determine the economically optimal sailing speed depending on actual costs.

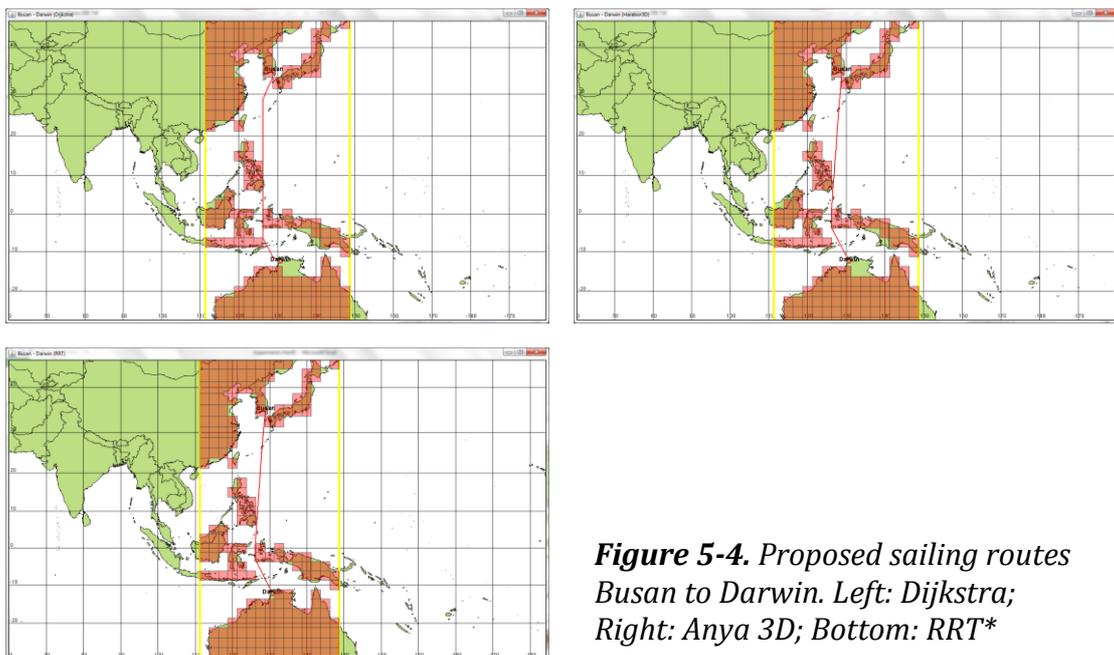
Runtime of the search is minimum for *Anya 3D* in all example cases. The decrease in runtime is largest for the open ocean crossings in Antwerp to Key West and Rio de Janeiro to Angola. Since no island grids block the shortest route, the search is directly steered towards the destination. This is supported by the number of explored nodes, which is less than 1% compared to *Dijkstra*. Runtime and number of explored nodes is expected to be more favourable for *Dijkstra* when it would also be implemented as *A\** with a heuristic that steers towards the goal. For Busan to Darwin, runtime of *Anya 3D* is on average 4.3% less than *Dijkstra*, with a number of explored nodes of 8% compared to *Dijkstra*. This indicates that node exploration in *Anya 3D* is more expensive than *Dijkstra*.

Speed (knots)	R <sub>calm water</sub> (kN)	MCR (%)	Fuel rate (t/h)	Speed (knots)	Time (hours)	K <sub>fixed</sub> -	K <sub>fuel</sub> -	K <sub>delay</sub> -	K <sub>total</sub> -
6.0	68.7	0.111	1.188	6.0	488	2.50	1.69	2.44	6.63
7.0	85.0	0.135	1.197	7.0	418	2.14	1.46	1.86	5.46
8.0	106.5	0.167	1.208	8.0	366	1.88	1.29	1.42	4.59
9.0	128.0	0.198	1.219	9.0	325	1.67	1.16	1.08	3.91
10.0	168.9	0.256	1.239	10.0	293	1.50	1.06	0.81	3.37
11.0	209.7	0.313	1.259	11.0	266	1.36	0.98	0.59	2.93
12.0	256.3	0.378	1.300	12.0	244	1.25	0.93	0.41	2.58
13.0	309.3	0.451	1.387	13.0	225	1.15	0.91	0.25	2.32
14.0	382.8	0.550	1.527	14.0	209	1.07	0.93	0.12	2.12
15.0	454.5	0.647	1.754	15.0	195	1.00	1.00	0.00	2.00
16.0	563.4	0.790	2.208	16.0	183	0.94	1.18	0.00	2.12
17.0	725.3	1.000	2.984	17.0	172	0.88	1.50	0.00	2.38

**Table 5-4.** Resistance, propulsion and cost function for route Busan to Darwin.

EXPERIMENT 1A:		Route generation without weather				
		Route: Busan - Darwin (2928 nm)				
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	2987	2928	2928	2928	2930
		100%	98.0%	98.0%	98.0%	98.1%
Sailing time:	[days]	8.30	7.18	9.38	8.13	8.14
		100%	86.5%	113.1%	98.0%	98.1%
Average speed:	[knots]	15.0	17.0	13.0	15.0	15.0
Fuel consumption:	[mton]	349.2	513.9	312.4	342.4	342.7
		100%	147.2%	89.4%	98.0%	98.1%
Runtime:	[ms]	168	156	181	147	1282045
Open set:	[-]	2452	76	76	73	-
Explored nodes:	[-]	2139	165	172	165	69984

**Table 5-5.** Route results from Busan to Darwin



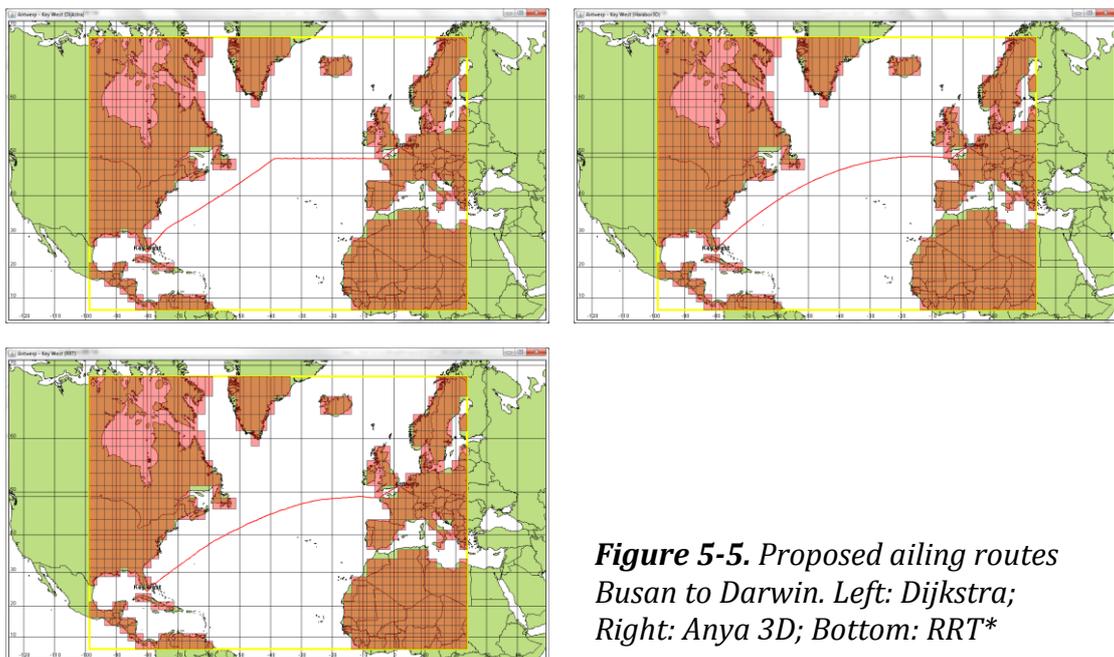
**Figure 5-4.** Proposed sailing routes Busan to Darwin. Left: Dijkstra; Right: Anya 3D; Bottom: RRT\*

Speed (knots)	R <sub>calm water</sub> (kN)	MCR (%)	Fuel rate (t/h)	Speed (knots)	Time (hours)	K <sub>fixed</sub> -	K <sub>fuel</sub> -	K <sub>delay</sub> -	K <sub>total</sub> -
6.0	50.5	0.102	1.003	6.0	687	2.67	1.86	3.58	8.10
7.0	67.2	0.136	1.013	7.0	589	2.29	1.61	2.76	6.66
8.0	86.6	0.177	1.025	8.0	515	2.00	1.43	2.15	5.57
9.0	108.6	0.225	1.039	9.0	458	1.78	1.29	1.67	4.73
10.0	133.5	0.281	1.056	10.0	412	1.60	1.18	1.29	4.06
11.0	159.3	0.341	1.074	11.0	375	1.45	1.09	0.98	3.52
12.0	187.7	0.411	1.095	12.0	343	1.33	1.02	0.72	3.06
13.0	218.5	0.489	1.125	13.0	317	1.23	0.96	0.50	2.69
14.0	252.6	0.581	1.212	14.0	294	1.14	0.96	0.31	2.41
15.0	295.8	0.698	1.306	15.0	275	1.07	0.97	0.14	2.18
16.0	346.4	0.841	1.438	16.0	258	1.00	1.00	0.00	2.00
17.0	398.6	1.000	1.633	17.0	242	0.94	1.07	0.00	2.01

**Table 5-6.** Resistance, propulsion and cost function for route Antwerp to Key West.

EXPERIMENT 1B:		Route generation without weather				
		Route: Antwerp - Key West (4120 nm)				
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	4163	4120	4120	4120	4125
		100%	99.0%	99.0%	99.0%	99.1%
Sailing time:	[days]	10.84	10.10	13.21	10.73	10.74
		100%	93.1%	121.8%	99.0%	99.1%
Average speed:	[knots]	16.0	17.0	13.0	16.0	16.0
Fuel consumption:	[mton]	374.2	395.8	356.5	370.3	370.8
		100%	105.8%	95.3%	99.0%	99.1%
Runtime:	[ms]	384	128	128	125	1421253
Open set:	[-]	4744	52	52	52	-
Explored nodes:	[-]	6282	56	55	56	72110

**Table 5-7.** Route results from Antwerp to Key West.



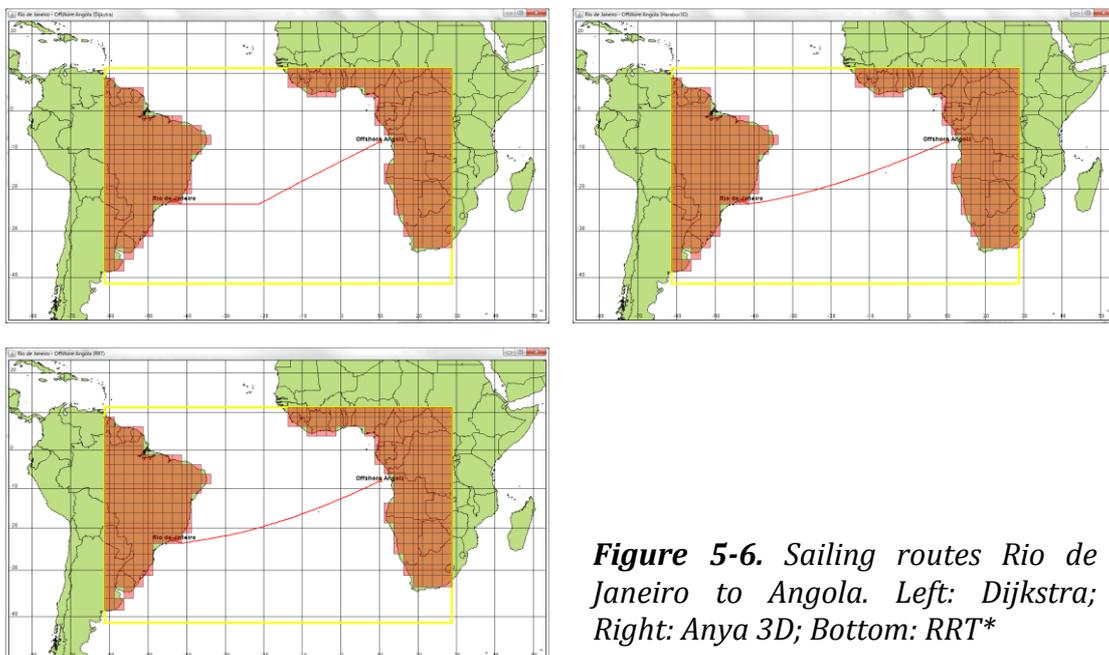
**Figure 5-5.** Proposed sailing routes Busan to Darwin. Left: Dijkstra; Right: Anya 3D; Bottom: RRT\*

Speed (knots)	R <sub>calm water</sub> (kN)	MCR (%)	Fuel rate (t/h)	Speed (knots)	Time (hours)	K <sub>fixed</sub> -	K <sub>fuel</sub> -	K <sub>delay</sub> -	K <sub>total</sub> -
2.0	22.0	0.245	1.774	2.0	1715	4.00	2.31	10.72	17.03
3.0	50.7	0.269	1.869	3.0	1143	2.67	1.63	5.95	10.25
4.0	89.9	0.300	2.016	4.0	858	2.00	1.32	3.57	6.89
5.0	138.9	0.339	2.167	5.0	686	1.60	1.13	2.14	4.87
6.0	197.4	0.387	2.226	6.0	572	1.33	0.97	1.19	3.49
7.0	265.0	0.444	2.613	7.0	490	1.14	0.97	0.51	2.63
8.0	341.5	0.512	3.066	8.0	429	1.00	1.00	0.00	2.00
9.0	427.0	0.593	3.945	9.0	381	0.89	1.14	0.00	2.03
10.0	522.1	0.691	4.892	10.0	343	0.80	1.28	0.00	2.08
11.0	626.3	0.808	6.028	11.0	312	0.73	1.43	0.00	2.16
12.0	741.6	0.951	7.536	12.0	286	0.67	1.64	0.00	2.31
13.0	866.9	1.124	9.660	13.0	264	0.62	1.94	0.00	2.55

**Table 5-8.** Resistance, propulsion and cost function for route Rio de Janeiro to offshore Angola.

EXPERIMENT 1C:		Route generation without weather				
		Route: Rio de Janeiro - Offshore Angola (3221 nm)				
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	3264	3221	3221	3221	3225
		100%	98.7%	98.7%	98.7%	98.8%
Sailing time:	[days]	17.00	10.93	22.37	16.78	16.80
		100%	64.3%	131.6%	98.7%	98.8%
Average speed:	[knots]	8.0	12.3	6.0	8.0	8.0
Fuel consumption:	[mton]	1250.8	2133.8	1195.0	1234.4	1236.1
		100%	170.6%	95.5%	98.7%	98.8%
Runtime:	[ms]	353	116	131	125	2002072
Open set:	[-]	2639	20	20	20	-
Explored nodes:	[-]	4166	29	29	29	87895

**Table 5-9.** Route results from Rio de Janeiro to offshore Angola.



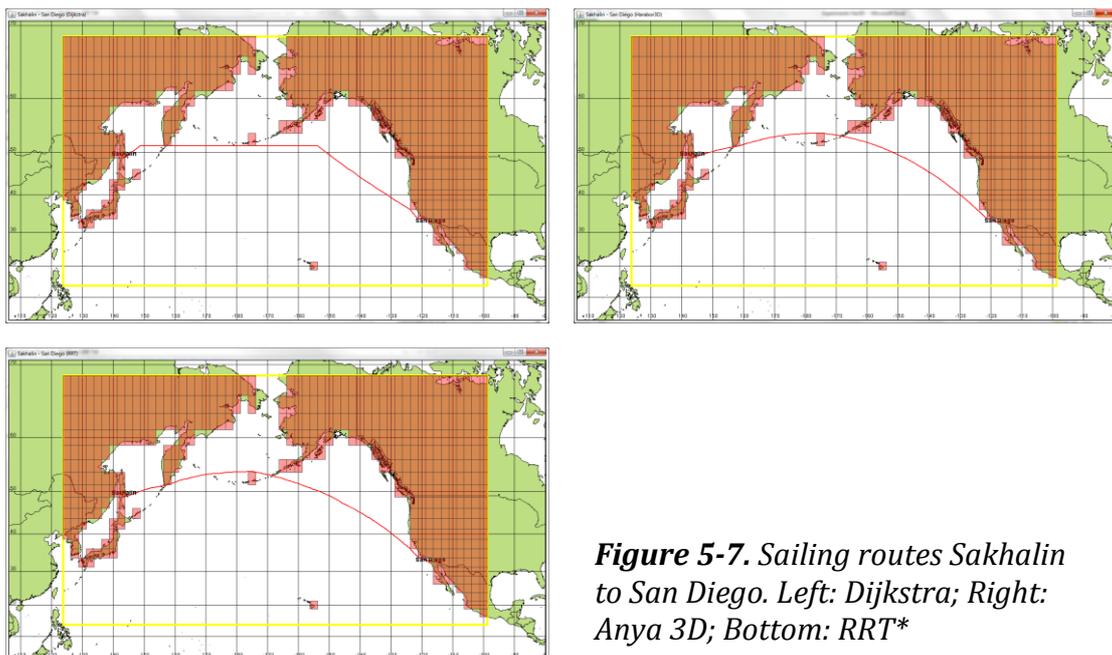
**Figure 5-6.** Sailing routes Rio de Janeiro to Angola. Left: Dijkstra; Right: Anya 3D; Bottom: RRT\*

Speed (knots)	R <sub>calm water</sub> (kN)	MCR (%)	Fuel rate (t/h)	Speed (knots)	Time (hours)	K <sub>fixed</sub> -	K <sub>fuel</sub> -	K <sub>delay</sub> -	K <sub>total</sub> -
10.0	372.9	0.177	3.258	10.0	460	1.90	1.38	1.82	5.09
11.0	442.8	0.213	3.293	11.0	418	1.73	1.26	1.47	4.46
12.0	529.7	0.259	3.336	12.0	383	1.58	1.17	1.18	3.93
13.0	615.3	0.307	3.381	13.0	354	1.46	1.10	0.93	3.49
14.0	718.4	0.366	3.437	14.0	329	1.36	1.04	0.72	3.11
15.0	826.5	0.432	3.499	15.0	307	1.27	0.99	0.54	2.79
16.0	946.3	0.508	3.571	16.0	288	1.19	0.94	0.38	2.51
17.0	1080.5	0.597	3.820	17.0	271	1.12	0.95	0.24	2.30
18.0	1225.9	0.699	4.129	18.0	256	1.06	0.97	0.11	2.14
19.0	1381.5	0.816	4.499	19.0	242	1.00	1.00	0.00	2.00
20.0	1543.9	0.948	5.025	20.0	230	0.95	1.06	0.00	2.01
21.0	1724.9	1.104	5.762	21.0	219	0.90	1.16	0.00	2.06

**Table 5-10.** Resistance, propulsion and cost function for route Sakhalin to San Diego.

EXPERIMENT 1D:		Route generation without weather				
		Route: Sakhalin - San Diego 4341 nm)				
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	4424	4341	4341	4341	4344
		100%	98.1%	98.1%	98.1%	98.2%
Sailing time:	[days]	9.70	8.89	11.30	9.52	9.53
		100%	91.7%	116.5%	98.1%	98.2%
Average speed:	[knots]	19.0	20.3	16.0	19.0	19.0
Fuel consumption:	[mton]	1047.6	1125.1	968.7	1027.8	1028.7
		100%	107.4%	92.5%	98.1%	98.2%
Runtime:	[ms]	796	312	586	343	3267001
Open set:	[-]	3767	193	481	239	-
Explored nodes:	[-]	7124	1121	2845	1165	87458

**Table 5-11.** Route results from Sakhalin to San Diego.



**Figure 5-7.** Sailing routes Sakhalin to San Diego. Left: Dijkstra; Right: Anya 3D; Bottom: RRT\*

## 5.4. Route generation with weather forecast

In the second set of experiments two weather forecasts per shipping route are investigated. Route generation is done based weather criteria for significant wave height and wind speed applicable to each specific trade. The criteria settings and route results for all eight weather scenarios are presented in the top part of Table 5-12 to Table 5-15.

For the route Busan to Darwin the first weather forecast returns a westbound solution for *Anya 3D* whereas both *Dijkstra* and *RRT\** follow an eastern route around the bad weather area. The optimized routes for time and fuel indeed return better results than *Dijkstra* considering respectively sailing time and fuel consumption. But the returned route is longer as well. For *Anya 3D* optimization settings equal to *Dijkstra* and *RRT\** still the Western route is returned, which requires further investigation. It should be noted that this implementation of *RRT\** keeps the vessel speed equal, which is similar to *Dijkstra*. In addition, *Dijkstra* and *RRT\** do only check the free space for route points; edges are not checked for obstacles. In contrary to this, *Anya 3D* does only allow passage through the free space and does not allow diagonal passage between two blocked grid cells that border along one edge. This could explain why *Dijkstra* and *RRT\** make the same navigational decisions, while *Anya 3D* selects the westbound route.

In the second weather forecast all routes follow the same westbound pattern. The *Anya 3D* solutions for the time and time/fuel optimized routes are optimal around the land grid corners. For fuel-optimization a minor course change is followed north of the Philippines. In addition a time root occurs along the route, which explains the average sailing speed of 12.81 knots. Fuel consumption for this route is indeed lowest of all cases; 9.7% lower than *Dijkstra* at 15.6% increase of sailing time. The gain in sailing time for the *Anya 3D* time-optimized route is 13.2% with respect to *Dijkstra*, at 47.6% additional fuel consumption. Runtime of all *Dijkstra* and *Anya 3D* runs in both weather scenarios is almost equal.

The route from Antwerp to Key West shows interesting results. For both weather scenarios the *Anya 3D* optimized routes for time and time/fuel follow a southern route compared to the others. Though realistic when considering real captain route decisions, these routes return longer sailing times. The returned routes for fuel optimization are shorter and save more than 5% fuel compared to *Dijkstra*. These routes are similar to *RRT\**. In all cases runtime of *Anya 3D* is lower than *Dijkstra* search.

The tug-tow combination from Rio de Janeiro to offshore Angola shows exponential growths of the search space in this *Anya 3D* implementation. For the first weather scenario no solution is returned for the fuel and time/fuel optimizations. The time-optimized route is returned at runtime 13 times longer than *Dijkstra*. To obtain this solution 3 times more nodes had to be explored. This bad performance of the algorithm could be caused by the scattered nature of the obstacles in this problem instant.

In the actual implementation of the *Anya 3D* algorithm the size of the search space increases exponentially with the number of obstacles. When an obstacle is present, new search root nodes are added around the corners of this obstacle. The space behind this obstacle will consequently be searched by all new root nodes that have overlapping visibility lines. For example, a single blocked grid cell would result in 4 root nodes that continue search behind this cell. It is therefore recommended that the faces and edges of the search space will be discretized and only store the lowest cost path. More expensive search nodes to edges or faces should be excluded from further investigation. Depending on the applied discretization this could result in loss of optimality. Therefore further investigation would be required to carefully study how to prune branches from the search tree.

The second weather forecast for the route Rio de Janeiro to Angola shows a slightly more northern route for *Anya 3D* compared to *Dijkstra* and *RRT\**. The bound of route lengths is within 1.4% of the *Dijkstra* route. Because of the large speed range of this transport, sailing time can be reduced more than 20%. *Anya 3D* did not find a route for its fuel-optimal sailing speed of 8 knots. Its lowest cost path encountered some time roots, resulting in an average speed of 8.98 knots and 14.6% more fuel than *Dijkstra*. Average runtimes for all for the *Anya 3D* runs are 40% lower compared to *Dijkstra*.

The first weather scenario of the final route from Sakhalin to San Diego returned similar route patterns for all three algorithms. The *Anya 3D* solutions were 3% shorter, while *RRT\** showed improvement by 8.2%. This route is more north and rejected by both *Dijkstra* and *Anya 3D*. The fuel-optimized route required 8.5% less fuel at 14.4% longer sailing time. The time-optimized route decreased sailing time by 5.6% at 3.9% more fuel consumption. The route optimized for time, fuel and delay passed some time roots and therefore did not meet the economical speed of 19 knots.

The second weather scenario showed a more southern route for *Anya 3D*. It requires further investigation why the algorithm did not return the northbound routes by *Dijkstra* and *RRT\**. This could be caused for the reasons as the east/west routing in the first weather scenario for Busan to Darwin.

In general, more space roots were investigated than time roots. This is also expected since land grid cells only induce space roots (1 per 12 hour time block). Time roots are only caused by weather obstacles. But such type of obstacles typically also causes space roots. Though being in minority, the time root inclusion in the final route solutions by *Anya 3D* is visible by their effect on the average sailing speed. In many cases this speed is not equal to the optimal speed according to the cost function.

## 5.5. Route performance in weather scenarios

The third and final set of experiments is setup to test how the proposed routes based on weather forecasts perform in the actual weather evolving over time. For this experiment a time-event based simulation is implemented as described in 5.1.1. In this simulation the weather is updated at the start of each new day. All simulation results are presented in the bottom part of Table 5-12 to Table 5-15.

For the first route, Busan to Darwin, performance of the *Anya 3D* time and time/fuel optimized routes is poor. Simulated sailing time to the destination is more than 25% higher than expected based on the route plan. The performance of *Anya 3D* optimized for speed is approximately 2% off for both weather scenarios. The *Dijkstra* solution for the first scenario is 3% off, but for the second scenario, simulated sailing time has exceeded the time according to plan by 14%.

The second route has better overall performance. Except for the time-optimized *Anya 3D* route all routes are approximately in a 6% bound exceeding the planned sailing time. The fuel-optimized route exceeds sailing time by 15.8% in the first and 10.7% in the second scenario. On the other hand, due to the lower sailing speeds the consumed fuel during the simulations has decreased by the same rate.

The relatively slow tug-tow combination from Rio de Janeiro to the offshore location near Angola performs worst of the problem instances. All simulated sailing speeds are higher than the algorithm route plans, resulting in early arrivals. The various runs show increased sailing speeds of more than 25%, except *Anya 3D* optimized for time. For both scenarios this simulation shows the best forecasted route at an increase of speed by approximately 14%.

The last route, Sakhalin to San Diego, has the most accurate route plans based on the simulations. All simulated sailing time, speed and fuel consumption is within a 5% bound with respect to the route plans, except *Anya 3D* in the first weather scenario. This run shows 6.9% longer simulated sailing time.

The simulations clearly show that the routes generated for lower speeds perform worse than the routes at higher speed. This could be explained by the fact that sailing at lower speed results in relatively long sailing time at decreased reliability of the weather forecast. Considering a specific route point, the slower voyage arrives later at this location than the voyage at higher speed. Since the reliability of the weather decreases in time, this lower speed route plan is less accurate.

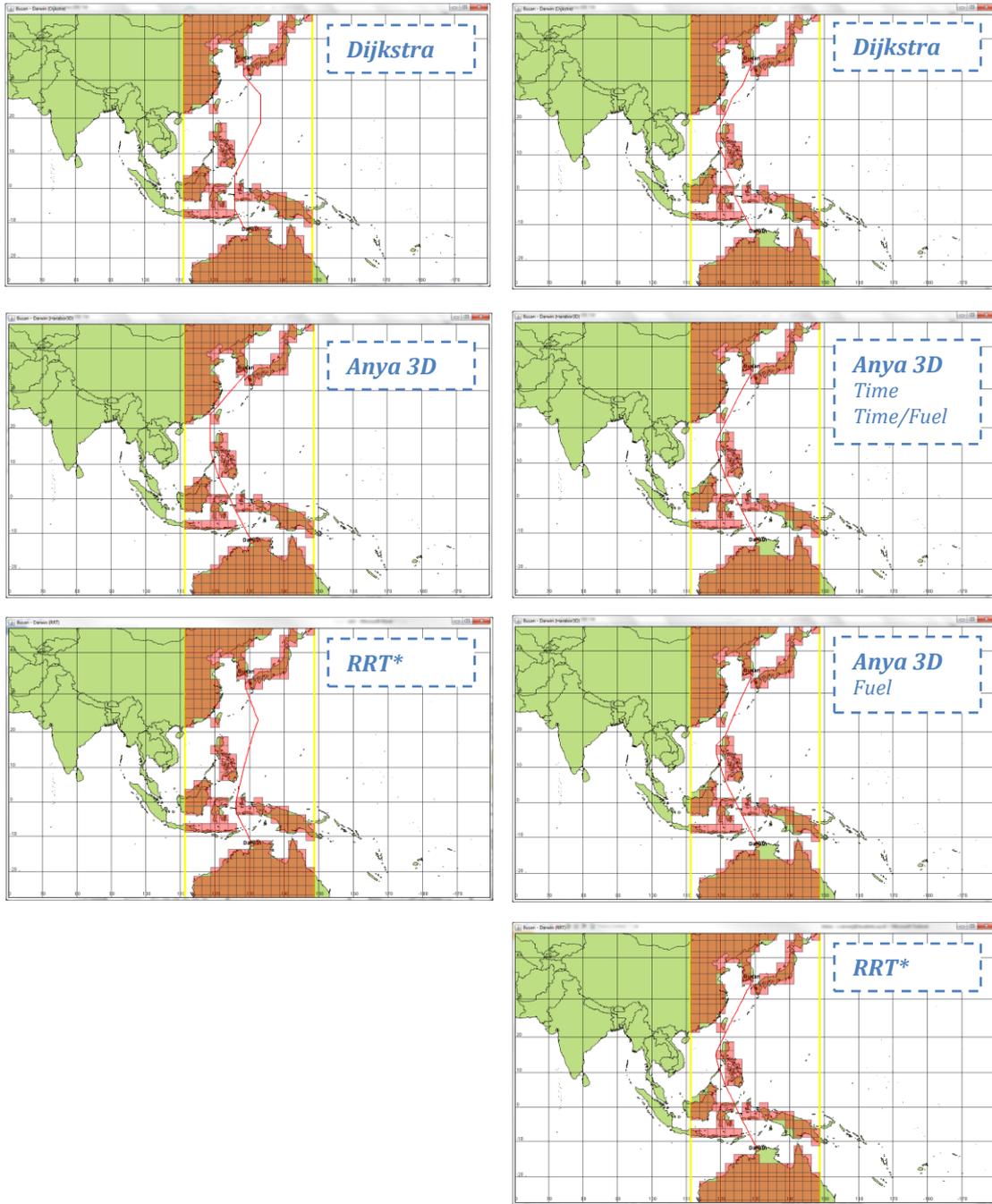
It is recommended to further investigate the uncertainty of the weather forecast. Unfortunately such statistics are not included in the present weather database. Incorporating reliability of the weather forecast in route generation could for example be done by running multiple search queries in randomly changed weather forecasts in a Monte Carlo simulation. A heuristic could be used to select the most reliable route from the set of generated routes.

EXPERIMENTS 2A + 3A		Route generation and simulation with weather criteria				
Departure: 28 July 1996		Route: Busan - Darwin				
		Criteria: 1) Hs ≤ 4.5 [m]				
		2) V <sub>wind</sub> ≤ 45 [knots]				
2A: Route generation						
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	3161	3214	3214	3214	3014
		100%	101.7%	101.7%	101.7%	95.3%
Sailing time:	[days]	8.78	7.88	9.98	8.93	8.37
		100%	89.7%	113.7%	101.7%	95.3%
Average speed:	[knots]	15.00	17.00	13.42	15.00	15.00
Fuel consumption:	[mton]	369.6	564.2	358.8	375.9	352.4
		100%	152.6%	97.1%	101.7%	95.3%
Runtime:	[ms]	206	203	209	268	1217604
Open set:	[-]	2227	109	93	136	-
Explored nodes:	[-]	2364	371	411	582	65643
3A: Simulation in weather						
Simulated sailing time:	[days]	9.07	7.99	12.75	11.47	
rel. to expected time		103.3%	101.4%	127.7%	128.5%	
Simulated sailing speed:	[knots]	14.52	16.77	10.51	11.68	
rel. to expected speed		96.8%	98.7%	78.3%	77.9%	
Simulated fuel consumption:	[mton]	381.9	571.9	471.3	482.8	
rel. to expected fuel		103.3%	101.4%	131.3%	128.5%	

EXPERIMENTS 2A + 3A		Route generation and simulation with weather criteria				
Departure: 22 Dec 1996		Route: Busan - Darwin				
		Criteria: 1) Hs ≤ 4.5 [m]				
		2) V <sub>wind</sub> ≤ 45 [knots]				
2A: Route generation						
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	
Route distance:	[nm]	3201	3148	3160	3148	3166
		100%	98.4%	98.7%	98.4%	98.9%
Sailing time:	[days]	8.89	7.72	10.28	8.72	8.79
		100%	86.8%	115.6%	98.1%	98.9%
Average speed:	[knots]	15.00	17.00	12.81	15.03	15.00
Fuel consumption:	[mton]	374.2	552.5	338.0	372.9	370.2
		100%	147.6%	90.3%	99.6%	98.9%
Runtime:	[ms]	162	178	178	181	1243915
Open set:	[-]	2651	122	60	61	-
Explored nodes:	[-]	1940	203	111	127	54411
3A: Simulation in weather						
Simulated sailing time:	[days]	10.12	7.88	13.28	9.19	
rel. to expected time		113.8%	102.2%	129.2%	105.3%	
Simulated sailing speed:	[knots]	13.18	16.63	9.92	14.28	
rel. to expected speed		87.9%	97.8%	77.4%	95.0%	
Simulated fuel consumption:	[mton]	394.6	564.6	437.4	392.5	
rel. to expected fuel		105.4%	102.2%	129.4%	105.3%	

**Table 5-12.** Weather routing results from Busan to Darwin. Top: departure 28 July 1996; Bottom: departure 22 December 1996.

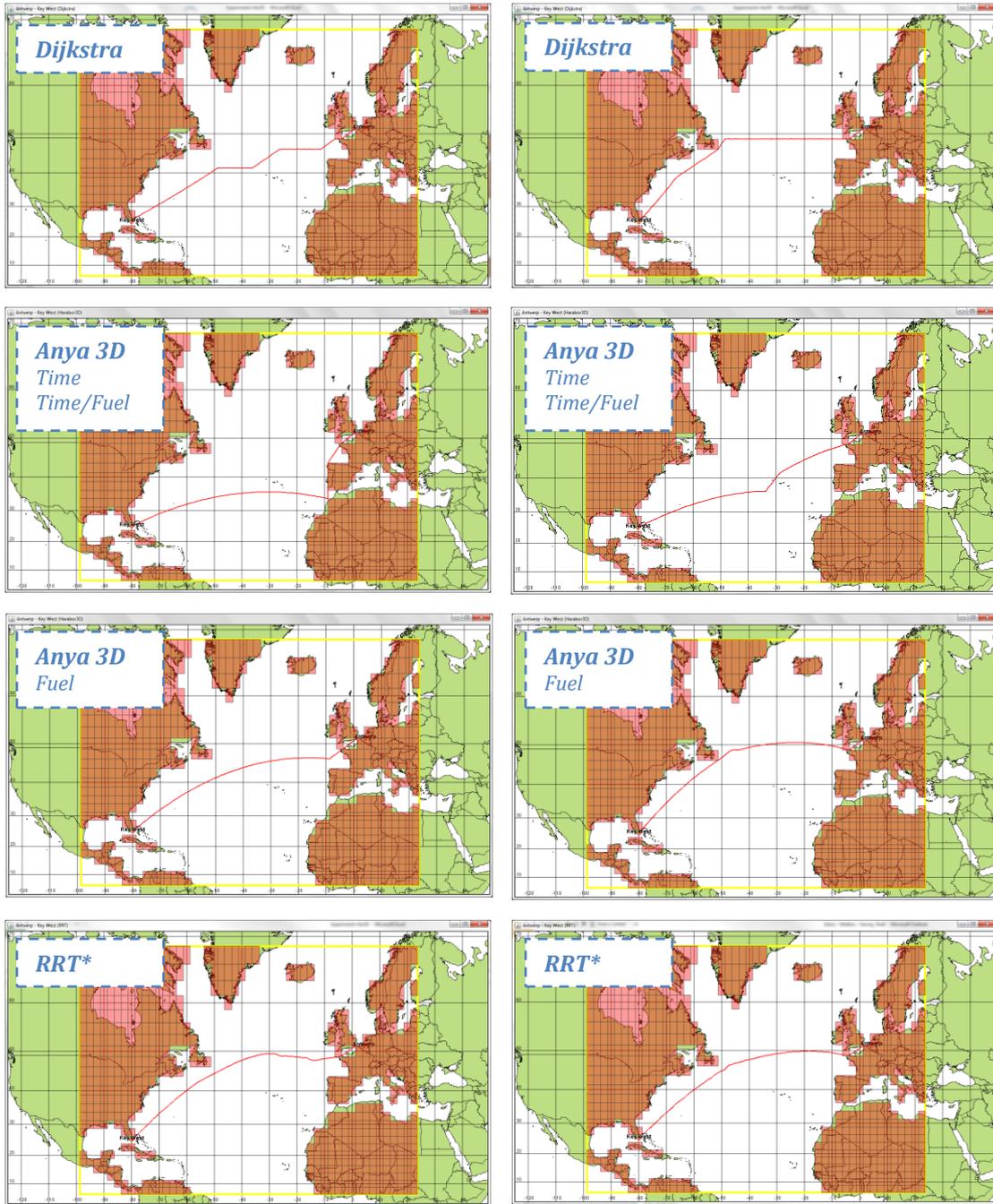


**Figure 5-8.** Weather routing results from Busan to Darwin. Left: departure 28 July 1996; Right: departure 22 December 1996.

<b>EXPERIMENTS 2B + 3B</b>		<b>Route generation and simulation with weather criteria</b>				
Departure: 31 Dec 1997		Route: Antwerp - Key West				
		Criteria: 1) Hs ≤ 8.0 [m]				
		2) V <sub>wind</sub> ≤ 50 [knots]				
<b>2B: Route generation</b>						
<b>Algorithm:</b>		<b>Dijkstra</b>	<b>Anya 3D</b>			<b>RRT*</b>
<b>Optimization:</b>			<b>Time</b>	<b>Fuel</b>	<b>Time/Fuel</b>	
Route distance:	[nm]	4270	5018	4188	5018	4167
		100%	113.7%	94.9%	113.7%	94.4%
Sailing time:	[days]	11.12	12.30	13.64	13.00	10.85
		100%	107.0%	118.7%	113.1%	94.4%
Average speed:	[knots]	16.00	17.00	12.79	16.08	16.00
Fuel consumption:	[mton]	383.8	482.0	366.2	453.6	374.5
		100%	121.5%	92.3%	114.4%	94.4%
Runtime:	[ms]	776	650	265	525	1543904
Open set:	[-]	5686	410	116	395	-
Explored nodes:	[-]	5340	2921	608	2206	61260
<b>3B: Simulation in weather</b>						
Simulated sailing time:	[days]	11.77	12.57	15.80	13.32	
rel. to expected time		105.9%	102.2%	115.8%	102.4%	
Simulated sailing speed:	[knots]	15.11	16.63	11.05	15.70	
rel. to expected speed		94.5%	97.8%	86.4%	97.6%	
Simulated fuel consumption:	[mton]	406.3	492.7	424.2	464.8	
rel. to expected fuel		105.9%	102.2%	115.8%	102.5%	

<b>EXPERIMENTS 2B + 3B</b>		<b>Route generation and simulation with weather criteria</b>				
Departure: 20 Dec 2000		Route: Antwerp - Key West				
		Criteria: 1) Hs ≤ 8.0 [m]				
		2) V <sub>wind</sub> ≤ 50 [knots]				
<b>2B: Route generation</b>						
<b>Algorithm:</b>		<b>Dijkstra</b>	<b>Anya 3D</b>			<b>RRT*</b>
<b>Optimization:</b>			<b>Time</b>	<b>Fuel</b>	<b>Time/Fuel</b>	
Route distance:	[nm]	4206	4345	4153	4345	4138
		100%	102.9%	98.3%	102.9%	98.0%
Sailing time:	[days]	10.95	11.31	13.14	11.82	10.78
		100%	102.8%	119.4%	107.4%	98.0%
Average speed:	[knots]	16.00	16.01	13.17	15.32	16.00
Fuel consumption:	[mton]	378.0	403.3	359.5	384.8	371.9
		100%	106.2%	94.7%	101.3%	98.0%
Runtime:	[ms]	847	291	172	302	1317850
Open set:	[-]	5776	298	99	208	-
Explored nodes:	[-]	5250	865	167	587	60867
<b>3B: Simulation in weather</b>						
Simulated sailing time:	[days]	11.45	12.00	14.55	12.54	
rel. to expected time		104.5%	106.1%	110.7%	106.1%	
Simulated sailing speed:	[knots]	15.31	15.09	11.89	14.44	
rel. to expected speed		95.7%	94.3%	90.3%	94.2%	
Simulated fuel consumption:	[mton]	395.2	427.0	397.9	408.0	
rel. to expected fuel		104.5%	105.9%	110.7%	106.0%	

**Table 5-13.** Weather routing results from Antwerp to Key West. Top: departure 31 December 1997; Bottom: departure 20 December 2000.

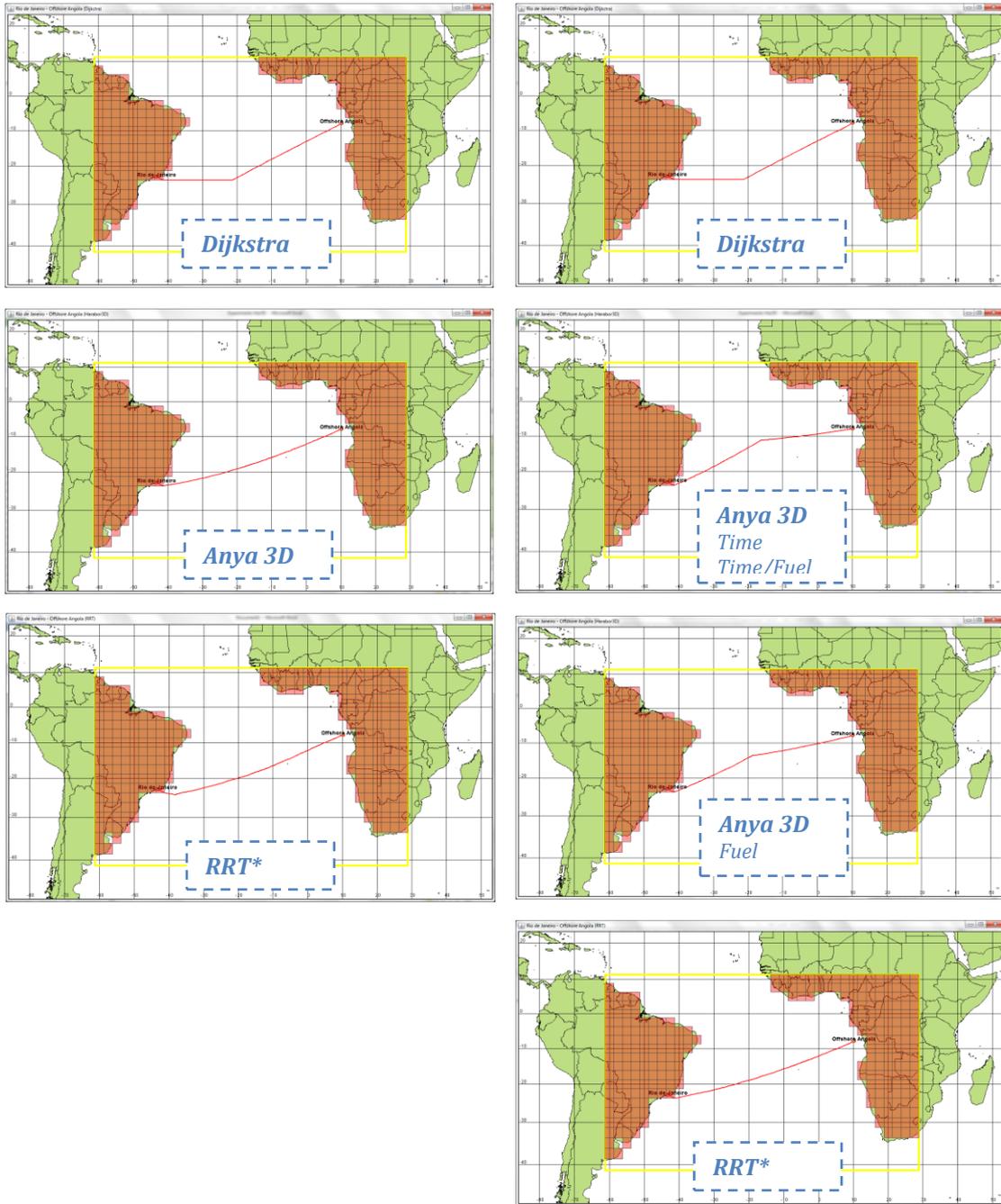


**Figure 5-9.** Weather routing results from Antwerp to Key West. Top: departure 31 December 1997; Bottom: departure 20 December 2000.

<b>EXPERIMENTS 2C + 3C</b>		<b>Route generation and simulation with weather criteria</b>				
Departure: 24 May 1996		Route: Rio de Janeiro - Offshore Angola				
		Criteria: 1) Hs ≤ 4.5 [m]				
		2) V <sub>wind</sub> ≤ 28 [knots]				
<b>2C: Route generation</b>						
<b>Algorithm:</b>		<b>Dijkstra</b>	<b>Anya 3D</b>			<b>RRT*</b>
<b>Optimization:</b>			<b>Time</b>	<b>Fuel</b>	<b>Time/Fuel</b>	
Route distance:	[nm]	3264	3221	-	-	3240
		100%	98.7%			99.3%
Sailing time:	[days]	17.00	10.93	-	-	16.88
		100%	64.3%			99.3%
Average speed:	[knots]	8.00	12.28	-	-	8.00
Fuel consumption:	[mton]	1250.8	2133.8	-	-	1241.9
		100%	170.6%			99.3%
Runtime:	[ms]	359	4811	-	-	1719377
Open set:	[-]	2562	3402	-	-	
Explored nodes:	[-]	4243	11448	200000+	200000+	84638
<b>3C: Simulation in weather</b>						
Simulated sailing time:	[days]	13.43	9.54	-	-	
rel. to expected time		79.0%	87.3%			
Simulated sailing speed:	[knots]	10.13	14.08	-	-	
rel. to expected speed		126.6%	114.6%			
Simulated fuel consumption:	[mton]	945.4	1862.5	-	-	
rel. to expected fuel		75.6%	87.3%			

<b>EXPERIMENTS 2C + 3C</b>		<b>Route generation and simulation with weather criteria</b>				
Departure: 09 Sept 2000		Route: Rio de Janeiro - Offshore Angola				
		Criteria: 1) Hs ≤ 4.5 [m]				
		2) V <sub>wind</sub> ≤ 28 [knots]				
<b>2C: Route generation</b>						
<b>Algorithm:</b>		<b>Dijkstra</b>	<b>Anya 3D</b>			<b>RRT*</b>
<b>Optimization:</b>			<b>Time</b>	<b>Fuel</b>	<b>Time/Fuel</b>	
Route distance:	[nm]	3264	3310	3272	3272	3225
		100%	101.4%	100.2%	100.2%	98.8%
Sailing time:	[days]	17.00	11.49	15.17	15.16	16.80
		100%	67.6%	89.3%	89.2%	98.8%
Average speed:	[knots]	8.00	12.00	8.98	8.99	8.00
Fuel consumption:	[mton]	1250.8	2080.8	1433.5	1433.7	1235.9
		100%	166.4%	114.6%	114.6%	98.8%
Runtime:	[ms]	359	182	265	208	1881756
Open set:	[-]	2799	82	177	104	-
Explored nodes:	[-]	4006	268	754	252	87725
<b>3C: Simulation in weather</b>						
Simulated sailing time:	[days]	12.77	10.07	11.99	11.99	
rel. to expected time		75.1%	87.6%	79.0%	79.1%	
Simulated sailing speed:	[knots]	10.65	13.70	11.36	11.37	
rel. to expected speed		133.1%	114.1%	126.5%	126.4%	
Simulated fuel consumption:	[mton]	939.9	1823.6	1134.4	1135.3	
rel. to expected fuel		75.1%	87.6%	79.1%	79.2%	

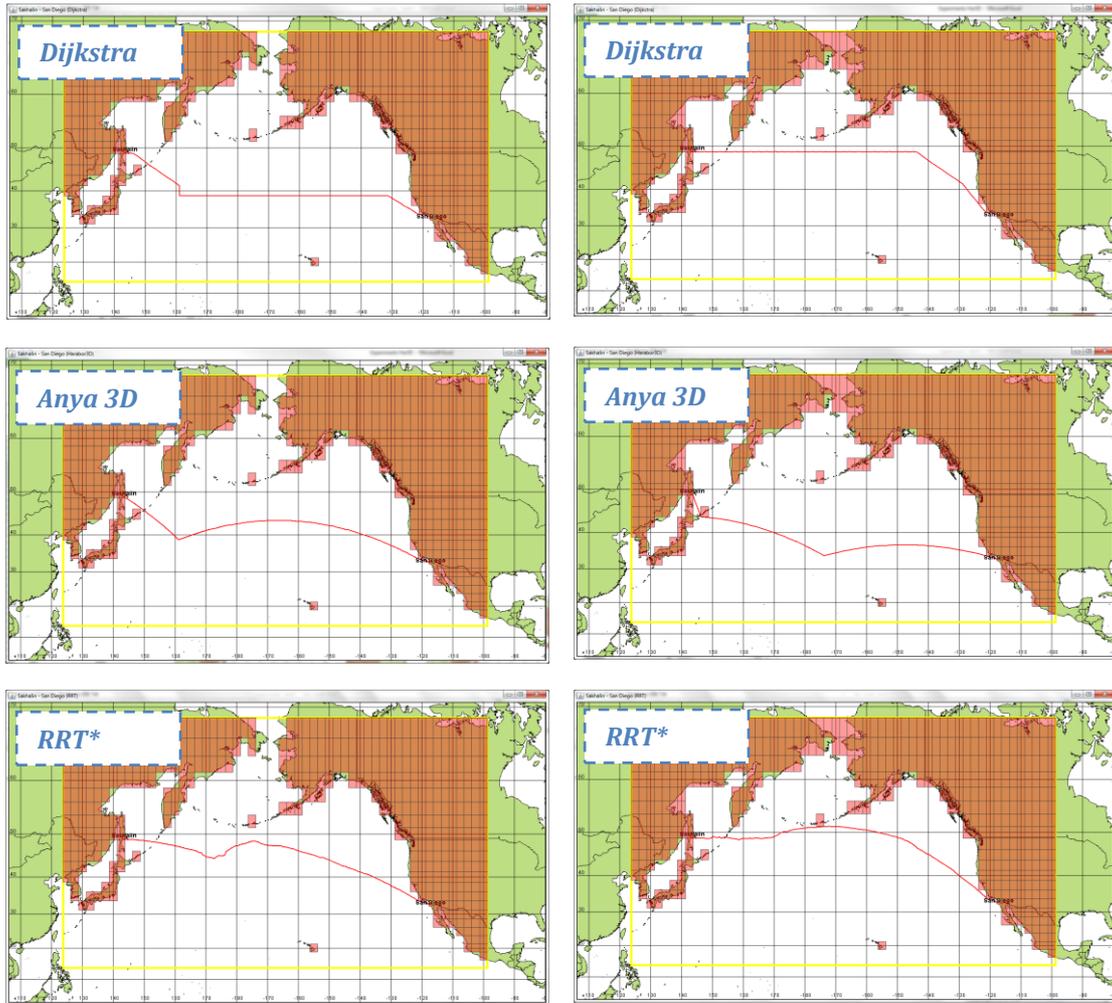
**Table 5-14.** Weather routing results from Rio de Janeiro to Offshore Angola. Top: departure 24 May 1996; Bottom: departure 09 September 2000.



**Figure 5-10.** Weather routing results from Rio de Janeiro to Offshore Angola. Top: departure 24 May 1996; Bottom: departure 09 September 2000.

EXPERIMENTS 2D + 3D		Route generation and simulation with weather criteria				
Departure: 10 Nov 2000		Route: Sakhalin - San Diego				
		Criteria: 1) Hs ≤ 5.5 [m]				
		2) V <sub>wind</sub> ≤ 45 [knots]				
2D: Route generation						
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	Time/Fuel
Route distance:	[nm]	5003	4847	4847	4847	4592
		100%	96.9%	96.9%	96.9%	91.8%
Sailing time:	[days]	10.97	10.36	12.55	10.44	10.07
		100%	94.4%	114.4%	95.2%	91.8%
Average speed:	[knots]	19.00	19.49	16.09	19.34	19.00
Fuel consumption:	[mton]	1184.8	1230.5	1083.8	1211.0	1087.2
		100%	103.9%	91.5%	102.2%	91.8%
Runtime:	[ms]	580	447	168	277	1709396
Open set:	[-]	4056	329	170	190	-
Explored nodes:	[-]	6835	1279	894	890	69616
3D: Simulation in weather						
Simulated sailing time:	[days]	11.19	10.74	13.42	10.83	
rel. to expected time		102.0%	103.6%	106.9%	103.7%	
Simulated sailing speed:	[knots]	18.62	18.81	15.05	18.65	
rel. to expected speed		98.0%	96.5%	93.5%	96.4%	
Simulated fuel consumption:	[mton]	1214.5	1274.3	1158.2	1255.0	
rel. to expected fuel		102.5%	103.6%	106.9%	103.6%	
EXPERIMENTS 2D + 3D		Route generation and simulation with weather criteria				
Departure: 08 Dec 2002		Route: Sakhalin - San Diego				
		Criteria: 1) Hs ≤ 5.5 [m]				
		2) V <sub>wind</sub> ≤ 45 [knots]				
2D: Route generation						
Algorithm:		Dijkstra	Anya 3D			RRT*
Optimization:			Time	Fuel	Time/Fuel	Time/Fuel
Route distance:	[nm]	4469	5087	5087	5087	4447
		100%	113.8%	113.8%	113.8%	99.5%
Sailing time:	[days]	9.80	10.43	11.12	11.13	9.75
		100%	106.4%	113.5%	113.6%	99.5%
Average speed:	[knots]	19.00	20.33	19.06	19.04	19.00
Fuel consumption:	[mton]	1058.3	1318.7	1207.8	1210.3	1052.9
		100%	124.6%	114.1%	114.4%	99.5%
Runtime:	[ms]	643	2168	88833	6555	1653311
Open set:	[-]	4208	2589	15843	1757	-
Explored nodes:	[-]	6683	9111	66930	4104	74657
3D: Simulation in weather						
Simulated sailing time:	[days]	10.09	10.86	11.62	11.63	
rel. to expected time		103.0%	104.1%	104.5%	104.5%	
Simulated sailing speed:	[knots]	18.46	19.53	18.24	18.22	
rel. to expected speed		97.2%	96.0%	95.7%	95.7%	
Simulated fuel consumption:	[mton]	1110.3	1373.3	1264.4	1262.1	
rel. to expected fuel		104.9%	104.1%	104.7%	104.3%	

**Table 5-15.** Weather routing results from Sakhalin to San Diego. Top: departure 10 November 2000; Bottom: departure 08 December 2002.



**Figure 5-11.** Weather routing results from Sakhalin to San Diego. Top: departure 10 November 2000; Bottom: departure 08 December 2002.

## 6. CONCLUSIONS & RECOMMENDATIONS

### 6.1. Conclusions

In this study the objective was to design a re-route algorithm for SafeTrans Monte Carlo Simulations that returns the optimal sailing route for a given weather forecast, while meeting user-defined transport criteria. In addition, the returned route should avoid land crossings and the search queries should have reasonable runtime compared to the actual *Dijkstra*-based re-route algorithm. Optimality of the route should be within an admissible range of engine settings and based on a cost function that includes sailing time, fuel consumption and late arrival penalty. The balance between these objectives should be set by the user.

SafeTrans contains a weather database subdivided by a grid of latitudes and longitudes that form a set of non-overlapping land and sea grid cells. Depending on the applicable transport criteria, sea grid cells are either traversable or non-traversable in time. When aiming for optimal routes, this problem transforms into an any-angle pathfinding problem in a continuous 3D search space with grid-based obstacle information. Harabor and Grastien have published *Anya*, a sketch of an optimal *A\**-based algorithm for such problems in 2D environments. It explores the space by building a search tree of grid edges visible from root points. Each edge is evaluated based on a lower bound of the path distance from start to goal.

In this study *Anya 3D* is proposed, which is a 3D extension of *Anya* and meets all study objectives. This algorithm searches the space by exploring grid faces and edges that are visible from root edges. Search nodes are evaluated based on the cost of the taut path from start to goal. The cost depends on sailing time, fuel consumption and late arrival penalty and uses a Great circle distance at economical speed heuristic to obtain a lower bound on the final path cost estimation.

1. For routes without considering weather, the proposed *Anya 3D* algorithm returned optimal routes for all tested problem instants. These routes are between 1 and 2% shorter than the actual *Dijkstra* algorithm.
2. Runtime of route generation with *Anya 3D* without considering weather was less than the actual *Dijkstra* algorithm in all cases.
3. Land crossings can be avoided by making a subdivision of land-containing grid cells into smaller convex polygon sections that are either land or sea. This will increase runtime of search.
4. Route generation with weather criteria showed different route solutions for several problem instants compared to the actual *Dijkstra* algorithm. Some solutions were more costly in terms of time and fuel. It is to be noted that in the experiments *Anya 3D* considered only the free space for route generation, while *Dijkstra* does not check edges on obstacle blocking.
5. The tested implementation of *Anya 3D* has exponential growth of the search space with the number of obstacles. For environments that contain



many obstacles this might result in large runtime or no algorithm solution.

6. Simulations have shown that performance of route plans by *Anya 3D* in developing weather are less reliable for lower sailing speed. This is as expected because at slower speed the sailing time increases and consequently the reliability of weather forecast decreases.

## 6.2. Recommendations

Based on the simulation experiments the following recommendations are made for further investigation.

1. The problem of exponential growth of the search space with increasing obstacles could be improved by discretizing the search space. This is typically seen in the weather routing problems of the tug-tow combination from Rio de Janeiro to offshore Angola. In this discretized space only the shortest path to each edge or face should be stored. The risk of discretization is loss of optimality, since pruning the search tree might cut off branches that contain the optimal solution. Therefore discretization should be carefully investigated for effective implementation of weather routing in SafeTrans.
2. In order to increase performance of route plans in weather the severity of forecasted weather should be included in the cost. Instead of traversing a *black/white* space, the environment could be coloured according to a *greyscale* based on the weather conditions that apply at that particular moment in time. In such a grid the colour is associated with travel cost.
3. Uncertainty of weather forecasts should be implemented in the route plan. This could for example be accomplished by running multiple search queries for randomly changed weather forecasts in a Monte Carlo simulation. A heuristic method could be used to select the most reliable route from the set of generated routes. As an alternative to incorporating uncertainty in the forecast, the sailing route can be updated daily.
4. Subdivision of land-containing grids into convex land and sea grid cells solves the actual problem of land crossings in route solutions. But this subdivision could be done in various ways, e.g. in rectangular shaped areas or by following land contours. The method of subdivision might have large consequences for complexity of algorithm coding and runtime of route generation. It should be noted that SafeTrans is primarily used for predicting ship motion behaviour along a route and that such land crossings do not influence the predicted motion behaviour.

In general, the following recommendation can be made with respect to the cost function.

5. In terms of profit optimization not only transportation costs should be included in the cost function. Transport revenues are also time dependent and could influence economical speed. Therefore it is advised to include revenues in the cost function as well.



## REFERENCES

Aalbers, A.B. et al. (2001), *SafeTrans: A new software system for safer rig moves*. Proceedings of the City University jack-up conference, London, September 2001.

Abramowski, P. et al. (2006), *Formal solution of ship weather routing problem via Pontryagin's maximum principle*. Sintesis Tecnologica, Vol.3, No.1, pp 27-31.

Allsopp, T. et al. (2000), *Optimal sailing routes with uncertain weather*. 35<sup>th</sup> Annual conference of the Operational Research Society of New Zealand. December 2000.

Bijlsma, S.J. (1975), *On minimal-time ship routing*. PhD thesis, Delft University of Technology.

Bijlsma, S.J. (2006), *On the computation of ship routes with minimal fuel consumption*. European Journal of Navigation, Vol.4, No.2, May 2006.

A. Botea, M. Müller and J. Schaeffer (2004). *Near Optimal Hierarchical Path-finding*. Journal of Game Development, Vol.1, pp 7-28.

Böttner, C-U. (2007), *Weather routing for ships in degraded condition*. International Symposium of Maritime Safety, Security and Environmental Protection, Athens, Greece, September 2007.

Chen, H.T. (1978), *A dynamic program for minimum cost ship routing under uncertainty*. PhD thesis, Massachusetts Institute of Technology.

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

Ferguson, D., Stenz, A. (2005), *Field D\*: An interpolation-based path planner and replanner*. Proceedings of the International Symposium on Robotics Research, pp 239 - 253.

Grin, R., Aalbers, A.B, Verwey, R., Ofosu-Apeasah, E., Quinn, S. (2012), *Enhancements in Monte Carlo simulation in SafeTrans 5*. Marine Heavy Transport & Lift III, RINA London, October 2012.

Hagiwara, H. (1989), *Weather routing of (sail-assisted) motor vessels*. PhD thesis, Delft University of Technology.

Haltiner, G.J. et al. (1962), *Minimal-time ship routing*. Journal of Applied Meteorology, Vol.1, No.1, March 1962.

Harabor D., Grastien A. (2011), *Online Graph Pruning for Pathfinding on Grid Maps*. Proceedings of the National Conference on Artificial Intelligences.

Harabor, D. (2012), *Fast Pathfinding via Symmetry Breaking*. www.aigamedev.com, February 2012.

Harabor, D., Grastien, A. (2013), *An optimal any-angle pathfinding algorithm*. Proceedings of the 23<sup>rd</sup> International Conference on Automated Planning and Scheduling.

Hart P., Nilsson N., Raphael B. (1968). *A formal basis for the heuristic determination of minimum cost paths*. IEEE Transactions on Systems Science and Cybernetics, volume 4, issue 2, pp 100–107.

Hinnenthal, J. (2008), *Robust Pareto – Optimum routing of ships utilizing deterministic and ensemble weather forecasts*. PhD thesis, Technische Universät Berlin.

Holte R.C., Perez M.B., Zimmer R.M., and MacDonald A.J. (1996). *Hierarchical A\*: Searching Abstraction Hierarchies Efficiently*. Proceedings of AAAI-96, pp 530–535.

James, R.W. (1957), *Application of wave forecasts to marine navigation*. U.S. Navy Hydrographic Office.

Karaman S., Frazzoli E. (2011). *Sampling-based algorithms for optimal motion planning*. International Journal of Robotics Research, Vol. 30, No. 7, pp 846-894.

Klompstra, M.B., Olsder, G.J., van Brunschot, P.K.G.M. (1992), *The isopone method in optimal control*. Dynamics and Control, Vol.2, No. 3, pp 281-301.

LaValle, S.M., Kuffner, J. (2011), *RRT-Connect: An efficient approach to single-query path planning*. Proceedings of the IEEE International conference on Robotics and Automation, Vol.2, pp 995-1001.

LaValle, S.M., Kuffner, J. (2001). *Randomized kinodynamic planning*. International Journal of Robotics Research, volume 20, Issue 5, pp 378-400.

LaValle, S.M. (2006), *Planning Algorithms*. Cambridge University Press, New York.

Law, A.M. (2007), *Simulation modeling & analysis*. McGraw-Hill. Fourth Edition.

Marie, S., Courteille, E. (2009), *Multi-objective optimization of motor vessel route*. TransNav, International Journal on Marine Navigation and Safety of Sea Transportation, Vol.3, No.2.

NIMA (1997). *Department of Defense World Geodetic System 1984, Its Definition and Relationships With Local Geodetic Systems*. Technical Report TR8350.2, 3<sup>rd</sup> Edition, 4 July 1997.

NIMA (2002), *The American Practical Navigator, Bowditch. Ch. 37 Weather routing*. [www.en.wikisource.org/wiki/The\\_American\\_Practical\\_Navigator](http://www.en.wikisource.org/wiki/The_American_Practical_Navigator).

Park, G. et al. (2008), *Development and evaluation of optimal routing system*. OMAE 2008, Estoril, Portugal.

MARIN (2011), *SafeTrans 5 Theory manual*. Report No. 21891-14-CPS, June 2011.

Nash, A., Daniel, K., Koenig, S., Felner, A. (2007), *Theta\*: Any-angle path planning on grids*. Proceedings of the AAAI Conference on Artificial Intelligence pp 1177 – 1183.

Spaans, J.A., Stoter, P. (2000), *Shipboard weather routing*. Proceedings of the IAIN World Congress and the 56<sup>th</sup> Annual meeting of the Institute of Navigation, San Diego, CA.

Sturtevant, N.R., Buro, M. (2005). *Partial pathfinding using map abstraction and refinement*. Proceedings of AAAI-05, pp 47–52.

Szłapczyńska, J. (2007), *Multi-objective approach to weather routing*. TransNav, International Journal on Marine Navigation and Safety of Sea Transportation. Vol.1, No.3.

Szłapczyńska, J. (2008), *Adopted isochrone method improving ship safety in weather routing with evolutionary approach*. R&RATA, Vol.1, No. 2.

Szłapczyńska, J. (2013), *Multi-criteria evolutionary weather routing algorithm in practice*. TransNav, International Journal on Marine Navigation and Safety of Sea Transportation. Vol.7, No.1.

de Wit, C (1968), *Mathematical treatment of optimal ocean ship routing*. PhD thesis, Delft University of Technology.

Yap, P., Burch, N., Holte, R. C., Schaeffer, J. (2011), *Block A\*: Database-driven search with applications in any-angle path-planning*. Proceedings of the 25<sup>th</sup> AAAI Conference on Artificial Intelligence.

Zappoli, R. (1972), *Minimum-time routing as an N-stage decision process*. Journal of Applied Meteorology. Vol.11, No.3, pp 429 – 435.