# A bidirectional path-finding algorithm and data structure for maritime routing

**Article** *in* International Journal of Geographical Information Science · July 2014

DOI: 10.1080/13658816.2014.887087



**3 authors**, including:

Dieudonne Tsatcha
Docapost
**4** PUBLICATIONS **15** CITATIONS
SEE PROFILE

Christophe Claramunt
Naval Academy Research Institute
**289** PUBLICATIONS **3,372** CITATIONS
SEE PROFILE

*International Journal of Geographical Information Science*
Vol. 00, No. 00, Month 2013, 1–21

**RESEARCH ARTICLE**

# *A Bidirectional Path-Finding Algorithm and Data Structure for Maritime Routing*

(*Received 00 Month 200x; final version received 00 Month 200x*)

Route planning is an important problem for many real time applications in open and complex environments. The maritime domain is a relevant example of such environments where dynamic phenomena and navigation constraints generate difficult route finding problems. This paper develops a spatial data structure that supports the search for an optimal route between two locations while minimizing a cost function. Although various search algorithms have been so far proposed (e.g. breadth-first search, bidirectional breadth-first search, Dijkstra's algorithm, A*, etc.), this approach provides a bidirectional dynamic routing algorithm which is based on hexagonal meshes and an Iterative Deepening A* algorithm ($IDA^*$), and a front to front strategy using a dynamic graph that facilitates data accessibility. The whole approach is applied to the context of maritime navigation, taking into account navigation hazards and restricted areas. The algorithm developed searches for optimal routes while minimizing distance and computational time.

**Keywords:** Maritime routing, computational geometry, artificial intelligence, geographic information science, navigation aids.

## 1. Introduction

Route planning between two geographical positions in a dynamic and open environment is often a frustrating problem for maritime navigation. As the number of entities and rules in these environments increase, path-finding problems are often non straightforward. Several successful algorithms developed for Intelligent Transportation Systems (ITS) have been applied to terrestrial environment where path-finding algorithms are network constrained. This is exemplified by in-vehicle Route Guidance System (RGS) and real time Automated Vehicle Dispatching System (AVDS) where routing strategies rely on road networks. Maritime environment also needs efficient algorithms to find the shortest path from an origin to a destination. But while predefined tracks exist for tankers or large ships, the maritime environment remains an open space where navigation in any direction is allowed, this making the path-finding problem a rather different issue than in network environments.

The challenge considered in this paper is to perform a real-time path-search where a given path is continuously constrained by the objects and the phenomena that are likely to modify the environment and the navigation. The real-time decision process to develop should be computed in a few seconds, this being sufficient to take into account possible environment changes (e.g. modification of visibility (fog, night), perception of ships or drifting objects) and to sup-

ISSN: 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/1365881YYxxxxxxxx
http://www.informaworld.com

port collision avoidance. Such a path-finding search should also take into account the specific properties of the maritime environment. Table 1 summarizes the main differences between maritime and network-based routing. Navigation in a maritime environment is constrained by several properties such as rules (e.g. traffic regulation, collision regulations (U.S. Coast Guard 1999)), restricted areas (e.g. traffic separation zones or lines), hazard areas (e.g. shoals), mobile objects (e.g. ships) and change dynamically with tide, current and wind. Moreover, visibility which is not constant (e.g., fog, day vs. night) can influence the perception of entities and actions to perform (Tsatcha *et al.* 2013). All these constraints have an impact on the resources, both on computational time and memory space, and penalize real time path-finding.

Nowadays, and to the best of our knowledge, Electronic Chart Display and Information Systems (ECDIS) used for maritime navigation integrate some path-finding algorithms as mentioned by Xu *et al.* (1994) and Yu *et al.* (2003). However, none of them take into account all the constraints previously described. Current maritime routing algorithms are usually applied to ship races and take into account meteorologic phenomena (wind, current) on top of raster data (Ahmadi *et al.* 2008). Figure 1 shows an example of a route proposed by MaxSea[1] software where the two most important factors involved in achieving optimum courses are the weather forecast and the speed of the boat. However, this approach does not preserve security and cannot be considered as real time routing.

Table 1. Main differences between maritime and network-based routing.

| | **Space** | **Entities** | **Time** | **Route planning** |
|---|---|---|---|---|
| Maritime routing | Open | Dynamic | High influence | Mainly based on landmarks |
| Network-based routing | Constrained | Dynamic | Low influence | Mainly network-constrained |

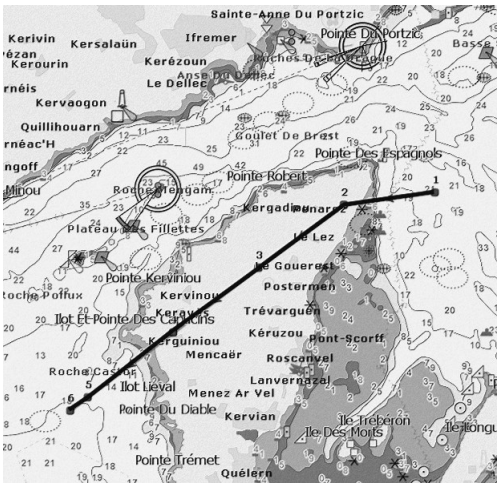



Figure 1. Example of a maritime route (black straight line) derived by MaxSea software that crosses a land area.

The approach presented in this paper is oriented to a navigation decision-aid system that should facilitate the decision-process of a captain in high stress situations by proposing an efficient path to follow. Let us first restrict the study to a real time path-finding problem applied to a two-dimensional maritime environment (i.e., 2D map) by taking into account both the length of the trajectory and the static objects composing a region. The choice of a planar

[1] http://www.maxsea.com

representation rather than an ellipsoidal representation is motivated by the area of navigation that is rather of the size of a region (i.e. coastal navigation) than of the size of a part of the Earth (i.e. off-shore navigation, less constrained by the environment). The maritime environment is modelled by S57 vector data[1] (format defined by the International Hydrographic Organization (IHO)). In order to achieve planning tasks, most of current maritime transport systems superimpose a grid over a region, and a graph is used to find the best paths. Usually, the terrain is modelled by regular squares (or tiles), hexes or octiles defining the mesh (Yap 2002). However, the number of cells in the path, as well as computational time, can be reduced in the search process using an iterative approach. In order to do so, let us introduce an iterative bidirectional path-finding algorithm based on an optimal graph structuring a hexagonal mesh. The search algorithm is founded on an Iterative Deepening A* algorithm $(IDA^*)$ .

The remainder of the paper is organized as follows. Section 2 introduces the main approaches or algorithms used in path-finding strategies and suggests the more relevant type of mesh for maritime routing. Section 3 introduces the background theory used to built two complementary meshes associated to the bidirectional path and proposes an optimized data structure for storing the hexagonal cells. Section 4 develops the approach retained to merge the $IDA^*$ algorithm and the bidirectional strategy for hexes and presents a case study. Finally, section 5 draws some conclusions and highlights future work.

## 2. Path-Finding Algorithms

Two classes of algorithms are usually applied to find an optimal path connecting two points denoted $s$ (starting point) and $t$ (terminal point): unidirectional and bidirectional algorithms. Each of them has many variants but the most useful algorithms are admissible algorithms. A search algorithm is admissible if it guarantees to find a valid solution that satisfies a minimal path among the set of existent solutions.

Moore (1957) and Dijsktra (1959) algorithms are commonly considered as the first admissible algorithms. Let us consider a mesh (i.e. a partition of the search area into cells) related to a directed graph $G(X, U)$ where $X$ is a collection of nodes (i.e. cells) and $U$ is a collection of arcs. Each arc of $U$ can be considered as an ordered pair of nodes of $X$. Starting with a node $n_0$ including initial point $s$ and for each iteration, the algorithm explores some parts of the graph[2] and marks the best successor of the current node until to reach the ending node $n_N$ that includes terminal point $t$. A path can be defined by a sub-graph of $G$ where the selected nodes of $X$ can be arranged in increasing number and where the initial node $n_0$ is assumed to be the root node. A path $u$ starting from root node $n_0$ to an ending node $n_N$ is denoted as a sequence of arcs, such as:

$$u = (e_{n_0n_1}, e_{n_1n_2}, e_{n_2n_3}, e_{n_3n_4}, e_{n_4n_5}, \ldots, e_{n_{N-1}n_N})$$

where $s \in n_0$, $t \in n_N$ and $e_{n_{k-1}n_k}$ is the arc formed by the nodes $n_{k-1}$ and $n_k$. Alternatively, one can write:

$$u = (n_0, n_1, n_2, n_3, n_4, ..., n_N).$$

The length of the path is computed using the distance $l$, i.e. :

$$l(u) = \sum_{i=0}^{N-1} ||e_{n_in_{i+1}}|| \text{ , where } ||.|| \text{ is the } L_2 \text{ norm.}$$

[1] http://www.iho.int/iho_pubs/IHO_Download.htm

[2] In the conceptualization developed, a graph is related to a mesh. A directed graph $G(X, U)$ where $X$ is a collection of nodes and $U$ is a collection of arcs. Each arc of $U$ may be considered as an ordered pair of nodes of $X$.

The best successor $n_j$ of current node $n_i$ minimizes the distance $l(n_i, n_j) + \lambda_{n_0,n_i}$ where $l(n_i, n_j)$ is the cost between nodes $n_i$ and $n_j$, and $\lambda_{n_0,n_i}$ is the minimal cost to reach node $n_i$ starting from $n_0$. Dijkstra's algorithm has been extended into a more efficient algorithm denoted $A^*$ by including the estimated remaining cost to reach the final node $n_N$. $A^*$ estimates the optimal cost of a current node $n_i$ with an evaluation function having the form $f(n_i) = g(n_i) + h(n_i)$, where $g(n_i)$ is the cost of the finding path from $n_0$ to $n_i$, and $h(n_i)$ is a heuristic estimating the cost of the remaining path to reach $n_N$ from $n_i$. $A^*$ is admissible and never overestimates the cost function $f$ at each iteration in comparison with algorithms which estimate the cost function at the end of the process. It has an iterative variant method denoted "Iterative Deepening $A^*$" $(IDA^*)$ proposed by Korf (1985) where the search is performed by a depth analysis in the graph.

All admissible algorithms can be applied by a bidirectional approach. The bidirectional algorithms belong to the second class of algorithms. The aim is to find the optimal path using a heuristic in both directions denoted "forward front" and "backward front" originated from starting node $n_0$ and ending node $n_N$. A Bidirectional Heuristic Pohl $A^*$ search $(BHPA^*)$ was proposed by Pohl (1971). It was later improved by Kwa (1989) who proposed a Bidirectional Search $A^*$ $(BSA^*)$ by adding several specific operations (nipping, pruning, trimming (removing) and screening (not placing)). The interest of these operations is to avoid unnecessary explorations and prevent repeated expansions in both fronts (Pulido *et al.* 2012). However, these two heuristics have a crucial drawback called missile metaphor (where the two searches pass without touching each other) due to the fact that they perform a heuristic with the principle front-to-end[1]. In comparison with $BHPA^*$ and $BSA^*$, Sint and de Champeaux (1977) and de Champeaux (1983) proposed a front-to-front[2] principle. Kaindl and Kainz (1997) show that the front-to-front evaluations are more efficient than the front-to-end evaluations. In the 90's, Dillenburg and Nelson (1994) and Manzini (1995) introduced an approach in comparison with the bidirectional search called perimeter search. This approach is based on a front-to-front strategy using a breadth-first search and generates nodes around the evaluated node and stores all of them located within a research area limited by a boundary. Later, Kaindl and Kainz (1997) devised an algorithm computationally cheaper than the previous ones that relies on dynamic changes called difference approaches. This approach is grounded on a known cost $g(n_i)$ and its heuristic estimates the cost from a given evaluation function $h(n_i)$ which minimizes the estimations obtained with other heuristics.

Cui and Shi (2011) show that the sort of grid used can have an impact on the performance of path-finding algorithms. Sahr *et al.* (2003) and Tong *et al.* (2013) introduce relevant discrete global grid systems (DGGSs) which are considered to be a promising structure for global geospatial information representation based on hexagonal grids. Having the objectives to find an optimal path in real-time in a maritime environment, we propose a routing algorithm based on a front-to-front Bidirectional Iterative Depth $A^*$ algorithm $(BIDA^*)$. The bidirectional algorithm is dynamic and develops a non overlapping mesh paving part of the maritime environment.

Hexagonal mesh performs the three main factors usually used to evaluate a graph search on a grid:

(1) It minimizes the branch-factor value[3] defined by Korf (1985). This criterion is illustrated in figures 2(a), 2(b), 2(c), 2(d) and summarized in table 2.

[1]Evaluations estimate the minimal cost of some paths from an evaluated node $n_i$ to the terminal node $n_N$.
[2]Evaluations estimate the minimal cost of some paths from an evaluated node $n_i$ in the forward front to node $n_j$ of the backward front.
[3]The branch-factor value is the number of possible adjacent nodes in order to avoid backtrack.

(2) It minimizes the number of nodes contained in the solution path. According to Yap (2002), this path minimizes the number of changes in direction.
(3) It proposes an optimal cluster movement (Yousefi and Donohue 2004). This criterion enables to determine the longest node sequence in a direction while minimizing both the direction changes and the total number of nodes composing the path (Figure 3).

Table 2. Number of adjacent nodes and branch-factor value for different grids.

| Grid | Tiles | Hexes | Octiles |
|---|---|---|---|
| Adjacent nodes | 4 | 6 | 8 for diagonal or 5 for non-diagonal cells |
| Branch-factor | 3 | 3 | 5 for diagonal cells and 3 for non-diagonal cells |

These facts have made hexagonal mesh as the optimal method which offers several advantages for a grid solution for the navigation planning problem. The next section focuses on the data structure used to implement the $BIDA^*$ algorithm with a hexagonal mesh. The $BIDA^*$ algorithm is then detailed in section 4.

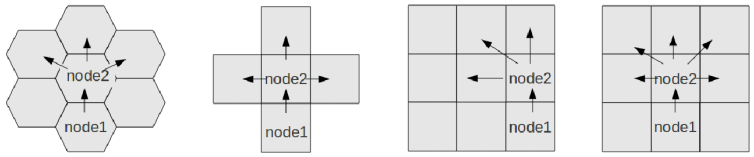



Figure 2. Branch-factor for (a) a hexagonal grid, (b) a square tile grid, (c) an octagonal grid and a non-diagonal cell, (d) an octagonal grid and a diagonal cell.

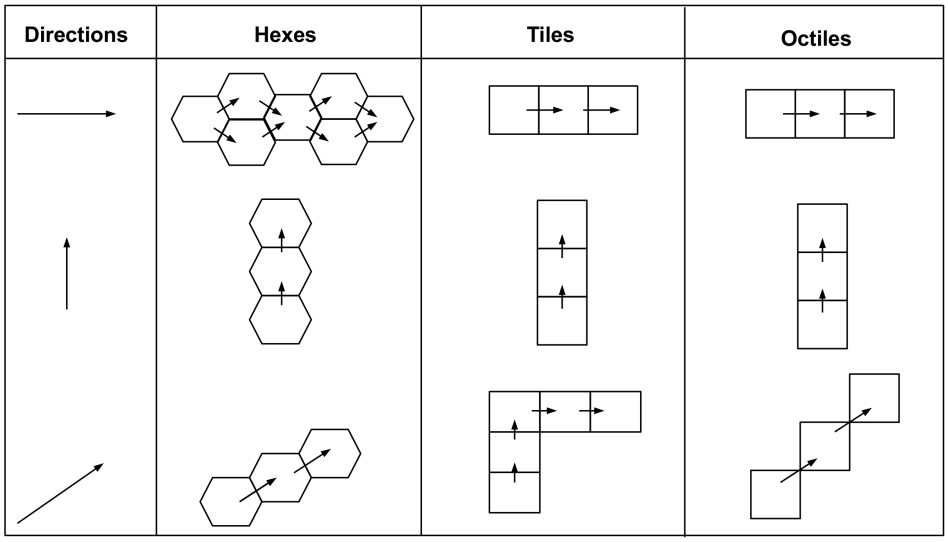



Figure 3. Optimal cluster movement on different grids.

## 3. Hexagonal Mesh Path and Data Structure

The aim of this section is to find the directions of propagation in the bidirectional search while producing a complementary mesh for the solution path. This is achieved by a two-step approach. The first step presented in section 3.1 determines the modelling process for a hexagonal mesh. This implies that the hexagonal cells should not overlap when the two paths meet. The second step, detailed in section 3.2, selects the more efficient direction of propagation that minimizes the constraints required by the application. Finally, section 3.3 proposes a data structure to store the hexagonal cells of the solution path.

### 3.1. *Complementary Hexagonal Mesh*

The main principle of this process is based on mesh parallelism applied on the planar faces. According to Pottmann *et al.* (2007a), parallelism can be used to produce a unified view or complementary mesh by keeping optimal nodes and the offset of geometrical properties. Let us consider two meshes $M$ and $M'$ displayed in figure 4. These authors prove that these meshes are combinatorially equivalent if and only if there is a 1-1 correspondence between vertices and edges. It follows that two meshes $M$ and $M'$ are considered as parallel since the corresponding edges are parallel (Pottmann *et al.* 2007b).

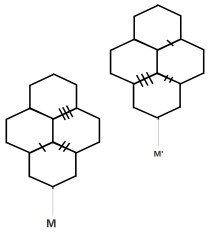
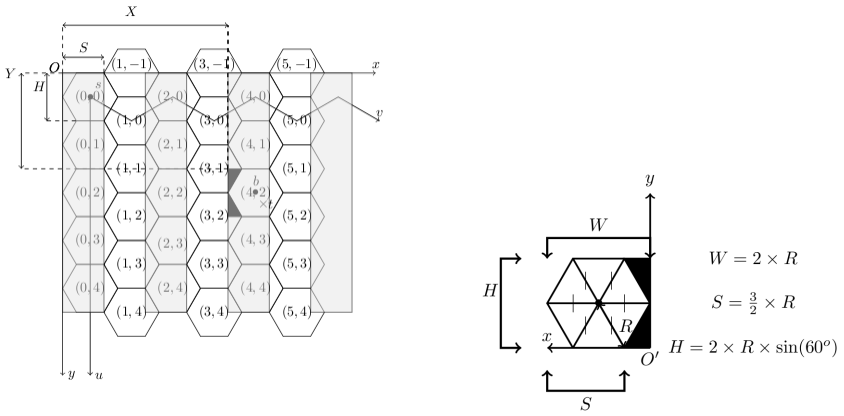



Figure 4. Example of parallel meshes $M$ and $M'$ with planar faces. Meshes are combinatory equivalent and corresponding edges are parallel.

Let us consider two points $s$ and $t$ in a two-dimensional space corresponding to the starting and terminal points. There always exists a propagation initiated by $s$ that covers the terminal point $t$ with a hexagon. Based on the proof of parallelism between meshes, there exists one and only one hexagon (node) initiated by a centre $b$ that contains the point $t$ and whose mesh initiated by itself meets exactly the mesh initiated by the centre $s$. One of the complexities of this algorithm is to compute the coordinates of the centre of the hexagon which covers point $t$ starting from point $s$. The computation of the centre $b$ is realized assuming the hexagonal mesh to be represented by a two-dimensional fixed matrix where indexes $i$ and $j$ represent the indexes of the columns and rows respectively using zigzag axes (Figure 5(a)). Finding the hexagonal cell that contains $t$ is equivalent to find the indexes $(i, j)$ of the hexagonal cell which covers $t$. The coordinates of this centre are derived from the formulae detailed as follows, where we consider $\Gamma(O, \overrightarrow{i}, \overrightarrow{j})$ being the initial coordinate system of figure 5.

The strategy to compute the centre $b = (b_x, b_y)$ of the hexagon that contains terminal node $t = (t_x, t_y)$ is realised by a two-step approach. The first step subdivides the space with rectangular tiles corresponding to a two-dimensional matrix. A tile is identified by its indexes $(i_{tile}, j_{tile})$ and the coordinates of its left-top corner. In figure 5, the widths of the rows and columns are respectively equal to $H$ and $S$. The second step is to use the indexes of the tile containing the terminal point $t$ for determining the indexes $(i_{hex}, j_{hex})$ of the final hexagon and the coordinates of its centre.




(a) (b)

Figure 5. Schema of a hexagonal mesh (a) in a coordinate system $\Gamma(O, \overrightarrow{i}, \overrightarrow{j})$ and (b) in a local coordinate system $\gamma(O', \overrightarrow{u}, \overrightarrow{v})$.

The indexes $(i_{tile}, j_{tile})$ of the tile that contains $t$ are computed by equation 1:

$$\begin{aligned} i_{tile} &= \left\lfloor \frac{t_x}{S} \right\rfloor \\ j_{tile} &= \begin{cases} \left\lfloor \frac{t_y}{H} \right\rfloor & \text{when } i_{tile} \text{ is odd} \\ \left\lfloor \frac{t_y - \frac{H}{2}}{H} \right\rfloor & \text{when } i_{tile} \text{ is even} \end{cases} \\ &= \left\lfloor \frac{y_{ts}}{H} \right\rfloor \text{ with } y_{ts} = t_y - i_{tile} \bmod 2 \times \frac{H}{2} \end{aligned} \tag{1}$$

In addition, the point $t$ can be defined in the local coordinate system $\gamma(O', \overrightarrow{u}, \overrightarrow{v})$ associated to the tile that contains $t$ (Figure 5(b)). Its local coordinates $(t_{x\_tile}, t_{y\_tile})$ are defined by equation 2:

$$t_{x\_tile} = t_x - i_{tile} \times S \quad \text{and} \quad t_{y\_tile} = y_{ts} - j_{tile} \times H \tag{2}$$

The next step concerns the identification of the hexagonal cell (among the three possible hexagonal cells that intersect the tile $(i_{tile}, j_{tile})$) containing the terminal point $t$. This point belongs either to the light black area or the dark black one in figure 5(a). The boundary of these two areas is defined by the following equation:

$$x\_tile = R \times \left| \frac{1}{2} - \frac{y\_tile}{H} \right| \tag{3}$$

It results that the indexes $(i_{hex}, j_{hex})$ of the hexagon that contains point $t(t_x, t_y)$ are:

$$\begin{aligned} i_{hex} &= \begin{cases} i_{tile} & \text{when } t_{x\_tile} > R \times \left| \frac{1}{2} - \frac{t_{y\_tile}}{H} \right| \\ i_{tile-1} & \text{otherwise} \end{cases} \\ j_{hex} &= \begin{cases} j_{tile} & \text{when } t_{x\_tile} > R \times \left| \frac{1}{2} - \frac{t_{y\_tile}}{H} \right| \\ j_{tile} - i_{tile} \bmod 2 + d & \text{otherwise} \end{cases} \\ \text{with } d &= \begin{cases} 1 & \text{when } t_{y\_tile} > \frac{H}{2} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

The parameter $d$ differentiates the case where $t$ belongs to the dark black top ($d = 0$) or dark black bottom ($d = 1$) hexagon.

Finally, the coordinates of the centre of the hexagon that contains point $t$ are:

$$b_x = i_{hex} \times S + R \quad \text{and} \quad b_y = j_{hex} \times H + \frac{H}{2} \tag{5}$$

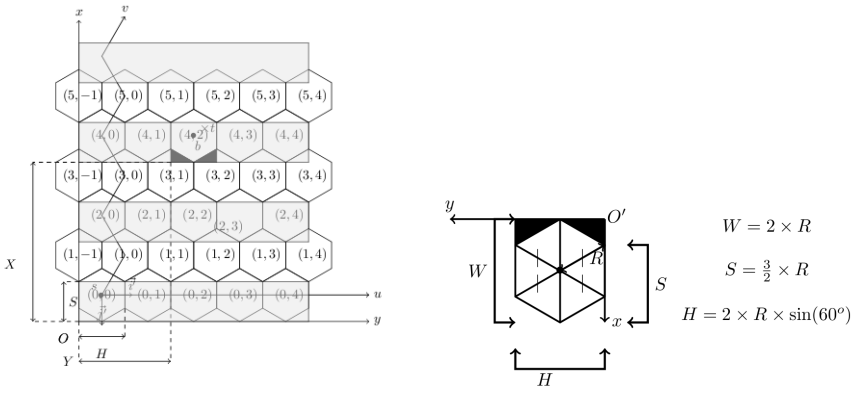
Centering the reference system $\Gamma$ to starting point $s$ by a translation of vector $(R, \frac{H}{2})$ leads to the new coordinates:

$$b_x = i_{hex} \times S \quad \text{and} \quad b_y = j_{hex} \times H \tag{6}$$

Two kinds of mesh can be distinguished for the paving. One can have a flat (Figure 5(b)) or pointy orientation (Figure 6(b)). The second orientation is derived from the first one by changing the coordinate system. We arbitrary chose this orientation in the following sections. Let $\Gamma'(s, \overrightarrow{i'}, \overrightarrow{j'})$ be this new reference system displayed in figure 6. $\Gamma$ is derived from $\Gamma'$ by applying a rotation of angle $\frac{\pi}{2}$ (Equation 7). In this new coordinate system, the point $b = (b_x, b_y)$ is defined by equation 8.

$$\begin{pmatrix} \overrightarrow{i} \\ \overrightarrow{j} \end{pmatrix} = \begin{pmatrix} cos(\frac{\pi}{2}) & -sin(\frac{\pi}{2}) \\ sin(\frac{\pi}{2}) & cos(\frac{\pi}{2}) \end{pmatrix} \times \begin{pmatrix} \overrightarrow{i'} \\ \overrightarrow{j'} \end{pmatrix} = \begin{pmatrix} \overrightarrow{-j'} \\ \overrightarrow{i'} \end{pmatrix} \tag{7}$$

$$\begin{aligned} \begin{pmatrix} b_x \\ b_y \end{pmatrix}_{\Gamma} &= (b_x \quad b_y) \times \begin{pmatrix} \overrightarrow{i} \\ \overrightarrow{j} \end{pmatrix} = (b_x \quad b_y) \times \begin{pmatrix} \overrightarrow{-j'} \\ \overrightarrow{i'} \end{pmatrix} = -b_x \overrightarrow{j'} + b_y \overrightarrow{i'} \\ &= (b_y \quad -b_x) \times \begin{pmatrix} \overrightarrow{i'} \\ \overrightarrow{j'} \end{pmatrix} = \begin{pmatrix} b_y \\ -b_x \end{pmatrix}_{\Gamma'} \end{aligned} \tag{8}$$




(a) (b)

Figure 6. Schema of a hexagonal mesh in a coordinate system $\Gamma'(s, i', j')$.

Finally, starting from a hexagonal cell centred on initial point $s$, this modelling process enables to compute the centre $b$ of the hex-cell containing the final point $t$ to be joined while

producing a complementary mesh for the routing problem. The next section focuses on the choices of directions of propagation for the forward and backward fronts in the bidirectional search.

### 3.2. *Directions of Propagation in the Bidirectional Search*

This section introduces a real-time dynamic direction finding while avoiding backtracks. A direction of exploration is hereafter denoted as a beam. Assuming a hexagonal mesh, the objective is to determine the set of hexes contained in a beam corresponding to a next possible direction (i.e. the search area) on which the $BIDA^*$ algorithm will be applied. Each searching area is defined with three adjacent hexes whose number corresponds to the value of the branch-factor (see section 2). It results that there exists six beams (three beams and their symmetric) initiated by a hexagonal cell (Figure 7).

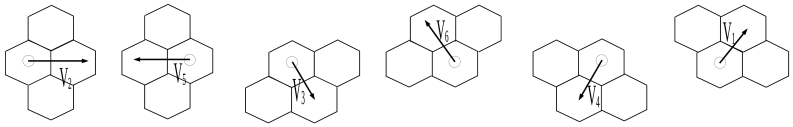



Figure 7. Symmetry in the direction of beams.

Assuming an initial hexagon of centre $c$, space is partitioned into conic sections $(Cone_i)_{i=1}^{6}$ centred on this hexagon. Each conic section, obtained by a rotation of $\frac{\pi}{3}$ around $c$, contains three adjacent hexes that can be chosen in the path-finding. A conic section is identified by its direction of propagation denoted $V_i$ being the interior bisector.

The solution beams result from a minimisation problem. One has to determine the efficient beams that minimize the criteria required by the application. The navigation or displacements imply to take into account two fundamental types of constraints: spatial and temporal. Each of them can be consistency or requirement constraints. On the one hand, consistency constraints are generic and do not depend on a project or application. They are always related to the intrinsic properties of the entities located in the environment. On the other hand, requirement constraints represent as an example building regulations, best-practise construction rules, or client requirements, which may vary from the applications (Borrmann *et al.* 2009). Therefore, let us consider $(c_s, c_t)_i$ the pair of spatial and temporal constraints restricted to the beam $i$. The challenge of the computational algorithm is to find the beam which satisfies the set of constraints or minimizes the functions modelling these constraints. Regarding the temporal constraint satisfaction problem, Schwalb and Vila (1998) define it like a computational solution to represent and perform queries on temporal occurrences and relations.

Borrmann *et al.* (2009) distinguishes three different types of spatial constraints: distance constraint (distance, closerThan, fartherThan and maxDist), directional constraints (above, below, northOf, southOf, eastOf, westOf) and topological constraints (disjoint, meet, overlap, cover, coveredBy, contain, equal and inside). This section explores directional constraints and the next section 4 dedicated to $BIDA^*$ satisfies the distance and topological constraints. The directional constraint is expressed as to be as close as possible to the direction $D = \overrightarrow{b_i b_j}$ defined from the straight line joining the centres $b_i$ and $b_j$ of the cells associated to the forward and backward fronts (see figure 8). The bidirectional process leads to use symmetrical beams on both parts of the propagation. Thus, optimal direction $V_{\bar{i}}$ minimizes the criterion defined by equation 9:

$$\widehat{(D, V_{\bar{i}})} = \text{Min}_{i=1\ldots6}\widehat{(D, V_i)} \tag{9}$$

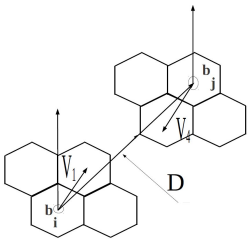



Figure 8. Symmetrical propagation using complementary beams.

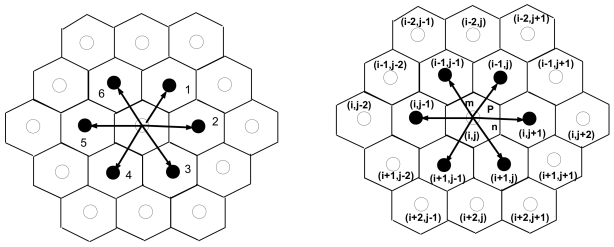
### 3.3. *Data Structure*

This section develops an optimized data structure to implement the $BIDA^*$ algorithm taking into account the issue of memory consumption. This implies that the data structure should enable to store the two oriented beams of the bidirectional search. This latter should be dynamic, including relationships linking a parent hexagon to a sequence a neighbouring child cells. The process should also be optimized with the objective that the coordinates of each hexagon and corresponding vertices must be computed only once. The proposed data structure is modelled as two separated dynamic oriented and complementary graphs where the root nodes are the hexagons that contain initial points $s$ and $t$ and whose the centre coordinates are defined by section 3.1. Two nodes are joined by one direction or arc (parent-child). We derive the following notations regarding the relations between two neighbouring nodes:

$$\begin{array}{l} R_{pc} : \text{parent-child relation} \\ R_{cp} : \text{child-parent relation} \\ R^{p}_{bb} : \text{brother-brother relation having parent } p \end{array} \tag{10}$$

The data structure allows to build and store hexagons by a dynamic process. Figure 9 shows that the hexagonal mesh can be organised from global or local points of view. According to an absolute reference system, a hexagon is identified with its row and column indexes $(i, j)$ as explained in section 3.1. As regards a local reference system centred on a current hexagon, six child hexagons are placed on different directions $(NE, E, SE, SO, O, NO)$ identified as $V_i$ in the previous section: $NE$ is denoted by 1, $E$ by 2, $SE$ by 3, $SO$ by 4, $O$ by 5 and $NO$ by 6. In the absolute reference system, relations between indexes (row, column) of two neighbouring nodes exist. Assuming the variation $(\delta_i,\delta_j)_d$ in direction $d$ from a node $(i, j)$ to a node $(i', j')$, indexes can be computed using equation 11. Table 3 identifies the variation coefficients associated with the possible directions.

$$(i', j') = [-1]^i(\delta_i, \delta_j)_d + (i, j) \tag{11}$$




(a) (b)

Figure 9. Modelling of neighbouring cells in a local (left) or global (right) reference system.

Table 3. Variation coefficients between parent and child.

| $d$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $(\delta_i,\delta_j)_d$ | $(-1,0)$ | $(-1,1)$ | $(0,1)$ | $(1,1)$ | $(1,0)$ | $(0,-1)$ |

A hexagonal cell $(i,j)$ is defined in the graph from the following notation:

$$
{}^{m}_{n}[Hex(i,j)]^{d_{pc}}_{S}
\begin{cases}
d_{pc}\text{: direction parent-child of the current cell from} \\
\quad\text{the parent point of view} \\
(i,j)\text{: indexes of the current hexagon (row, column)} \\
m,n\text{: left and right directions bordering direction } d_{pc} \\
S\text{: set of directions identifying the children}
\end{cases}
\tag{12}
$$

In figure 9(b), the graph node related to the cell $(i-1,j)$ is denoted as ${}^{6}_{2}[Hex(i-1,j)]^{1}_{6,1,2}$. Directions $m=6$, $d_{pc}=1$ and $n=2$ are defined from the indexes $(i,j)$ of the parent cell and correspond to a beam having a direction $V_1$ illustrated by figure 7(c) (right configuration). Directions in set $S=\{6,1,2\}$ are defined from the indexes $(i-1,j)$ of the current cell and identify the possible neighbouring cells $(i-2,j)$, $(i-2,j+1)$ and $(i-1,j+1)$ extending the structure. Considering the definition of a hexagonal mesh in any directions and at the same resolution, the modelling is realized by a growing process. The first step is the modelling of the root cell $(i,j)$ (Figure 10). To start with, the cell $(i,j)$ is assimilated to a point. It can be either the starting point $s$ or the centre of the hexagon containing the terminal point $t$. Rotating around the centre of the cell enables to update the set of children stored in set S. At the end of this process the data structure is restricted to a single root cell denoted as ${}^{0}_{0}[Hex(i,j)]^{0}_{1,2,3,4,5,6}$. By convention, a null direction ($m=0$, $d_{pc}=0$ or $n=0$) implies that the cell has no parent.

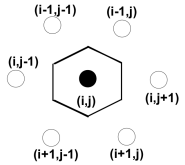



Figure 10. Illustration of the root cell modelling process.

The second step consists in the creation of the six child cells of the initial or root cell. This step defines the first ring around an initial hexagon (Figure 11). In order to optimize the data structure, the brother-brother relations $R^{p}_{bb}$ between the children are updated. The graph displayed in figure 12 shows the data structure associated to the first ring of cells where the cell $(i,j)$ is the root node. Each arc is oriented from a child to its parent and labelled with the direction $d_{cp}$. The direction $d_{cp}$ is not only used in order to label the arcs in the graphs of figures

12 and 14 but also to update the child nodes in S of the parent cell. It can be derived that if a child node "looks" its parent in direction $d_{cp}$ then the parent "looks" its child in the direction $d_{pc}$ defined by equation 13:

$$d_{pc} = (d_{cp} + 2) \bmod 6 + 1 \tag{13}$$

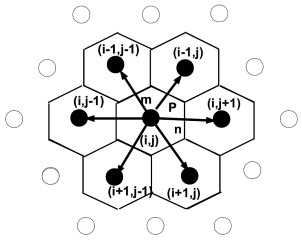



Figure 11. Illustration of the first ring modelling process.

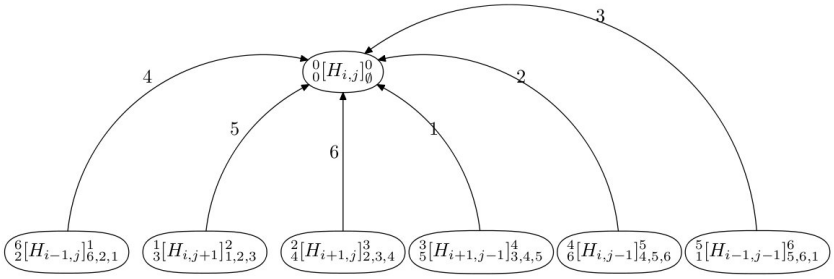



Figure 12. Graph corresponding to the structure of figure 11.

Applying the previous process to the following child cells leads to the second ring modelling process illustrated by figure 13 and corresponding graph presented in figure 14. Starting from the first child cell ${}^{6}_{2}[H_{i-1,j}]^{1}_{6,2,1}$ three child cells are built in directions 6, 1 and 2 (Figure 13(a)). This leads to the definition of cells ${}^{5}_{1}[H(i-2,j)]^{6}_{5,6,1}$, ${}^{6}_{2}[H(i-2,j+1)]^{1}_{6,1,2}$ and ${}^{1}_{3}[H(i-1,j+1)]^{2}_{1,2,3}$ (Figure 14). These new cells are considered as the children of the current cell $(i-1,j)$ that implies to update the brother relations $R^{p}_{bb}$ between the new three cells and the older ones (i.e. to update the relations between brothers $(i,j+1)$, $(i-1,j+1)$ and brothers $(i-1,j-1)$, $(i-2,j)$). The bordering new brothers to update are indicated by the indexes $m$ and $n$ in the cell notation. This implies to create a link between the brothers identified by directions $m$ and $(m+4)$ mod $6+1$ (according to the current cell $(i-1,j)$) as well as the brothers identified by directions $n$ and $n$ mod $6+1$. In the previous example, set $S$ of cell $(i,j+1)$ is modified to include its new brother in its direction 1. Thus, number 1 is removed from S and cell $(i,j+1)$ becomes ${}^{1}_{3}[H(i,j+1)]^{2}_{2,3}$ (Figure 14). The same principle is applied to cell $(i-1,j-1)$ where $S$ is modified to include its new brother in its direction 1. Thus, number 1 is removed from S and cell $(i-1,j-1)$ becomes ${}^{5}_{1}[H(i-1,j-1)]^{6}_{5,6}$ (Figure 14). This process is repeated on the next child cells of the root cell (see figures 13(b),13(c),13(d),13(e),13(f)) and ends the second ring. Next iterations are based on the same principle. The resulting data structure is optimized in the sense that there is no graph node duplication and that node links are computed once.

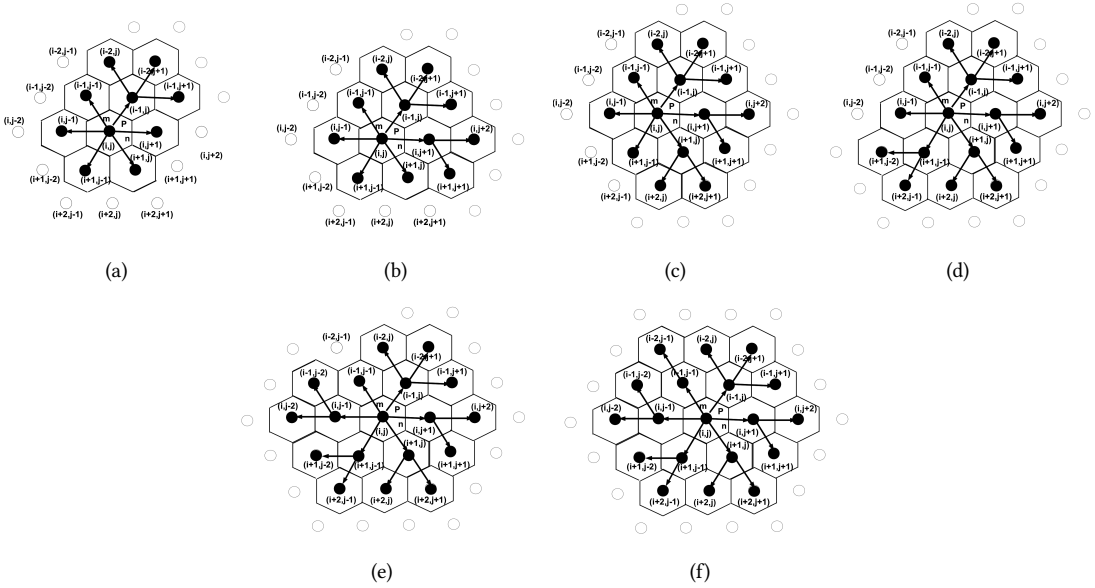


Figure 13. Illustration of the second ring modelling process.

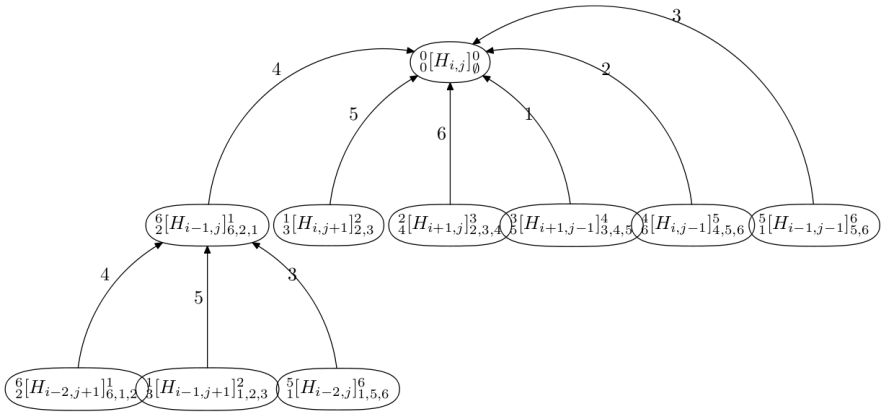
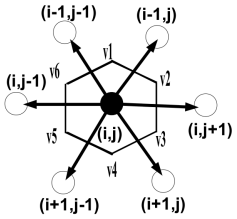


Figure 14. Graph corresponding to the structure of figure 13(a).

The previous data structure has been presented from a general context and for a whole mesh. Using a direction of propagation like defined in figure 7 of section 3.2 limits the number of computation and produces a sub-graph of the proposed one. This reduces the computation cost and provides an optimal data structure to apply the bidirectional iterative path-finding algorithm.

From a computational and visualisation point of view, two neighbouring cells are joined by two common vertices. Assuming the vertices $(v_i)_{i=1}^{6}$ of a hexagon (where $v_1$ is the vertex at the top of a hex-cell) and in order to avoid repeated vertex calculations, each direction $V_i$ is linked to the vertex $v_i$ in the hexagonal cell. In figure 15, the rotation around the centre of the hexagon $(i, j)$ in direction $(V_i)_{i=1}^{6}$ leads to the creation of the six vertices $v_i$. Each of these vertices are common to the parent cell $(i, j)$ and two of its child cells. For each hexagonal cell, a hash table stores all the vertices previously computed avoiding redundant vertex computations in the mesh and optimizing the visualisation process. The hash table is updated taking into account the mesh generation process. As an example and after the creation of the root cell $(i, j)$ (Figure 15), the vertex $v_2$ belongs to the three cells $(i, j)$, $(i - 1, j)$ and $(i, j + 1)$ and a neighbouring

cell $(i-1, j)$ has its two vertices $v_4$ and $v_5$ already built (hash table view per line). Centred on a cell and analysing the hash table, one deduces the vertices not yet defined and their direction of construction.




| Cells / $v_i$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ |
|---|---|---|---|---|---|---|
| (i-1,j-1) | | | × | × | | |
| (i,j-1) | | × | × | | | |
| (i+1,j-1) | × | × | | | | |
| (i+1,j) | × | | | | | × |
| (i,j+1) | | | | | × | × |
| (i-1,j) | | | | × | × | |
| (i,j) | × | × | × | × | × | × |

Figure 15. Hash table illustrating the vertex computations of the hexagon during the creation of the root cell $(i, j)$.

## 4. $BIDA^*$ on Hexes

### 4.1. *Bidirectional Path-Finding based on $IDA^*$*

The complexity of search algorithms is usually considered in terms of time, space and cost of the solution path. $IDA^*$ is known to be asymptotically optimal in these three dimensions for exponential tree searches (Korf 1985). This latter can be efficiently combined with bidirectional search and $IDA^*$ is the only known algorithm that is able to find an optimal solution when practical resources are limited. Dantzig (1960) initially proposed the bidirectional algorithm and an interpretation as follows: "*If the problem is to determine the shortest path from a given origin to a given destination, the number of comparisons can often be reduced in practice by fanning out from both the origin and the destination, as if they were two separate independent problems.... However, once the shortest path between a node and the origin or the destination is found in one problem, the path is conceptually replaced by a single arc in the other problem.*" Nicholson (1966) and Dreyfus (1969) proposed many approaches to improve this previous description. From this interpretation, Korf (1985) derives that bidirectional search optimizes space and time by searching simultaneously forward from the initial state and backward from the goal state, and storing the states generated until a common one is found on both search frontiers. Table 4 summarizes the cost of the $IDA^*$ algorithm with different grids (Yap 2002). The hexagonal mesh offers a good compromise between the length of the path and the complexity. Assuming that a hashing schema and hexagonal mesh are used, it results that the complexity of bidirectional algorithm in time and space is $O(2.42^{\frac{D-H}{2}})$.

Table 4. Complexity of $IDA^*$ according to different grids (Yap 2002). $D = l(u)$ and $H$ corresponds to the effect of the heuristic on the algorithm.

| **Grid** | **Tiles** | **Hexes** | **Octiles** |
|---|---|---|---|
| Average depth | $1.00D$ | $0.81D$ | $0.71D$ |
| $IDA^*$ complexity | $0(3.00^{D-H})$ | $0(2.42^{D-H})$ | $0(2.77^{D-H})$ |

The principle of $IDA^*$ is as follows. For each iteration, a Depth-First Search $(DFS)$ is realized to find the next node in the graph. The $DFS$ explores the three possible nodes in a beam

cutting the branches when the value of the cost function $f(n_i) = g(n_i) + h(n_i)$ exceeds a threshold value. The cost function $f(n_i)$ considers that $g(n_i)$ is the cost of the finding path from $s$ to $n_i$, and $h(n_i)$ is a heuristic estimating the cost of the remaining path from $n_i$ to $t$. The threshold value used for an iteration is the minimum cost value among all the possible cost values at the previous iteration. The idea of this algorithm is not to minimize the depth search, but rather to minimize the total cost of the path (i.e. function $f$). An example of node selection is illustrated by figures 16, 17 and 18. Our algorithm restricted to a search direction constrained by beams does not counteract the admissibility property of $IDA^*$ algorithm.

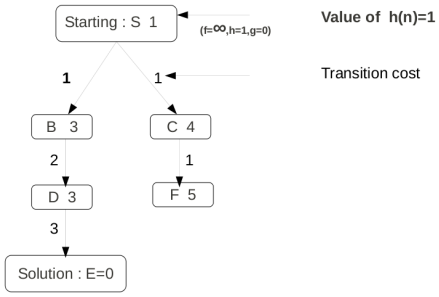



Figure 16. An initial graph with cost values.

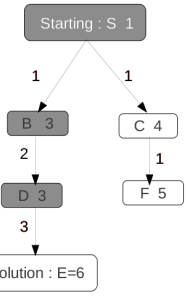



Figure 17. The solution path.

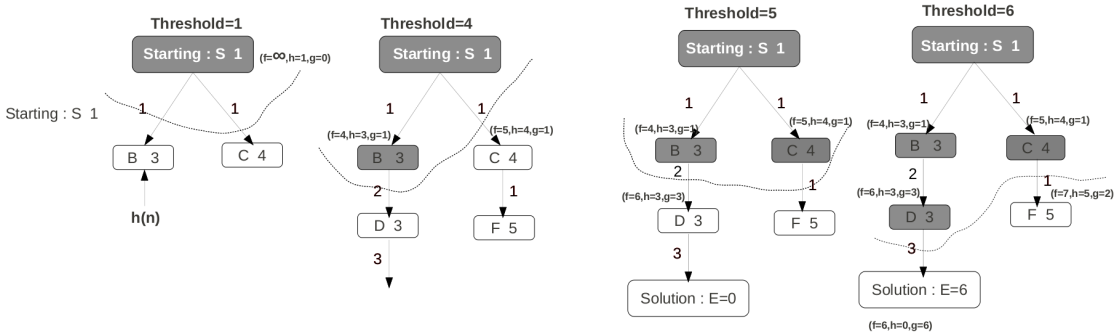



Figure 18. Node selection (in dark grey) using $IDA^*$.

The algorithm presented in appendix A provides the main concepts used in the implementation of $BIDA^*$ for the case studies presented in section 4.2. $Map$ represents the set of geographical features used for the maritime environment. Two threads $T1$ and $T2$ have been applied in parallel for the bidirectional search. $(tip)$ causes the current thread to suspend its execution for a specified period avoiding concurrent processing. Each thread is related to a dynamic data structure of search denoted as $Mesh^i$ $(i = 1, 2)$ where a mesh is a set of hexagonal cells as defined by equation 12. At each step of the hexagon cell creation, a node $N^{(i,j)}$ corresponds to a vertex and can be assimilated to a row of the hash table (see figure15). In order to avoid a redundant process, each mesh is managed by two stacks of nodes. The first stack of nodes is a set of promising nodes to explore and evaluate using a function $f = g + h$, denoted as open stack ($Stack_1$). $Stack_1$ has a LIFO[1] architecture. The second is a set of visited nodes that are not selected to reach the goal and is denoted as closed stack of nodes ($Stack_2$). The

[1]Last-in, First-out

radius of the hexagonal cells is linked to parameter $r$. It ensures either a fine or large resolution of the solution path. Two approaches can be taken into account. On the one hand, $r$ can be invariant during the path-finding algorithm as presented in figure 20 ensuring a single level of resolution. On the other hand, a multi-resolution approach can be used where a discrete grid system is represented by a group (aperture) of 3, 4 or 7 hexagons (Sahr 2011). All these apertures preserve the centre of the current cell but its radius can change allowing to refine or enlarge the resolution dynamically. Three parameters enable the change of resolution of hexagonal cells: i) centre $c$ ii) radius $r$ and iii) deviation angle $\alpha$ computed between the symmetrical axes of two consecutive resolutions. Thus, one can determine the properties (i.e. $r'$, $c'$ and $\alpha'$) of a hexagonal grid at a lower resolution from the parameters at a higher one (i.e. $r$, $c$ and $\alpha$). In aperture 4, the parameters are computed as follows: $r' = 2 * r$, $c' = c$ and $\alpha = 0$. Figure 19 is an example of $BIDA^*$ algorithm with such a multi-resolution approach applied on an aperture 4. The main drawback encountered using a multi-resolution strategy is that it counteracts the complementarity conditions (see section 3.1) between the two directional paths, i.e. the two last cells of each directional path intersect.

## 4.2. $BIDA^*$ for Maritime Routing

The $BIDA^*$ algorithm has been applied to maritime data to find the optimal route joining two positions corresponding to the departure and arrival points of a ship trajectory. In order to find an optimal route, one should take into account two constraints. The first one is to avoid hazard areas during routing while taking into account meteorological and environmental information such as wind and current. The second one is to consider the semantic information generated by a region or an object (e.g. cardinal buoy, alignment) that can modify the trajectory ship. In order to avoid hazard areas and for each iteration of the $BIDA^*$ algorithm, let us compute the spatial relationships (equal, disjoint, intersect, touch, cross, within, contain, overlap (Freeman 1975)) between a cell $(i, j)$ in the mesh and the different objects forming the maritime environment. If an intersection exists, the search tries to find an other solution path among the brother cells (see section 3.3). In the case where no solution is found among the "brothers", the strategy is to have a backward process. One comes back to the "father" cell in order to find a solution among its brothers. In the case study, the cost function $g$ (respectively $h$) evaluates the Manhattan distance between the starting (respectively ending[1]) and the current point.

The approach has been applied to maritime data extracted from Electronic Navigational Charts (ENC). These charts are defined from vector files where geographic objects are modelled using geometrical primitives like point, line, polygon defined in the European Datum 1950 (ED50) geodetic datum. The S57[2] format describes the concepts of the maritime environment with their attributes. The proposed example uses electronic chart number 7066 provided by the SHOM[3] ("Service Hydrographique et Océanographique de la Marine", Brest, France) and represents a maritime area in the surrounding area of Brest city in France.

The user first submits departure point $s$, destination point $t$ and the radius of the hexagonal cells. The cell radius can be arbitrary set for computational time purpose (large radius reduces computational time) or having a meaning for the application like being a safety or security area around the ship related to a minimal distance and defined according to the speed and rate of turn[4] of the ship.

[1] In the bidirectional algorithm, the ending point does not correspond to the terminal point $t$ but to the centre of the front node in the symmetrical path.
[2] http://www.iho.int/iho_pubs/IHO_Download.htm
[3] http://www.shom.fr
[4] ROT attribute in the Automated Identification System (AIS) format.

Figures 19 and 20 respectively illustrate the results using a multi-resolution and a single resolution hexagonal mesh. In our case study, the relevant cells of the bidirectional path are displayed satisfying the heuristic function while avoiding the hazard areas like land areas (light grey areas), buoys (light grey circles) and restricted areas (dark grey areas). The sea area is represented by a white colour. The prototype was implemented in Java language using the Java topology suite (JTS[5]) for representing the spatial relationships. The black line illustrates the proposed route and the hexagonal cells correspond to the graph nodes deployed for this solution. Each node of the graph builds exactly three children. The three black nodes in figure 19 are the children of the current nodes in the forward and backward fronts. Table 5 proposes some results to evaluate the performance of the strategy. The criteria hereafter used are the ratio $\gamma$ quantifying the gain of nodes in the ordinary path-finding problem and the computational time. The ratio $\gamma$ evaluates the number of cells required in the proposed algorithm in comparison with the number of cells needed for covering the whole search space ($Nb_{hexagon} = \frac{\text{Surface of the search space}}{\text{Surface of a hexagon}}$). As an example, the ratio $\gamma_1$ of the first use case in table 5 is equal to $\frac{142177-1014}{142177} = 0.99$. This means that only $1\%$ of the cells of a coverage grid is needed to compute the optimal solution. Computational time presented in the last column of table 5 has been obtained by a computer with an Intel Celeron Processor E3200 (2.40 GHz). One can remark that this strategy minimizes the processing time by optimizing the storage of nodes. This is especially true in large and open environnments.

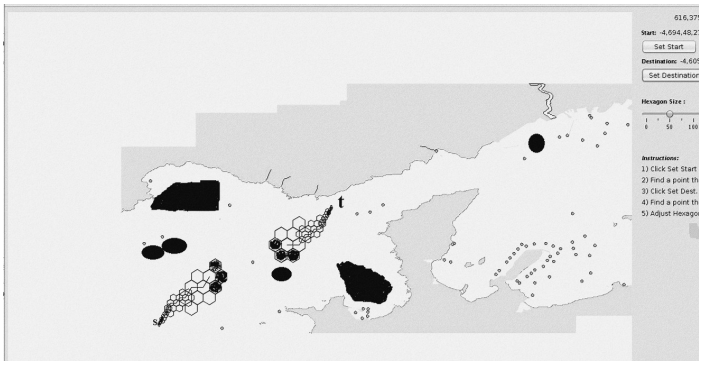



Figure 19. Example of iterative maritime routing using the $BIDA^*$ algorithm on a dynamic data structure. Case 1: multi-resolution hexagonal meshes with an aperture 4.

Table 5. Evaluation of the proposed strategy in comparison with a usual strategy based on a coverage grid.

| Departure point $s$ (km) | Destination point $t$ (km) | Radius of a hexagon (m) | Number of cells in the graph | Number of cells meshing the environment | Number of cells in the solution path | $\gamma$ | Computational time (ms) |
|---|---|---|---|---|---|---|---|
| -524282.64 5379318.29 | -497390.83 5382653.57 | 50 | 1014 | 142177 | 338 | 0.99 | 21350 |
| -524282.64 5379318.29 | -510401.87 5377558.01 | 50 | 576 | 142177 | 172 | 0.99 | 8207 |
| -524313.63 5379102.12 | -497019.12 5382653.57 | 96 | 567 | 38568 | 185 | 0.98 | 5309 |
| -524313.63 5379102.12 | -497019.12 5382653.57 | 122 | 432 | 23881 | 144 | 0.98 | 5332 |
| -524313.63 5379102.12 | -497297.90 5385000.62 | 150 | 558 | 15798 | 159 | 0.96 | 11477 |
| -505228.18 5375396.25 | -497297.90 5385000.62 | 103 | 240 | 15798 | 80 | 0.98 | 2969 |

[5]http://www.vividsolutions.com/jts/jtshome.htm

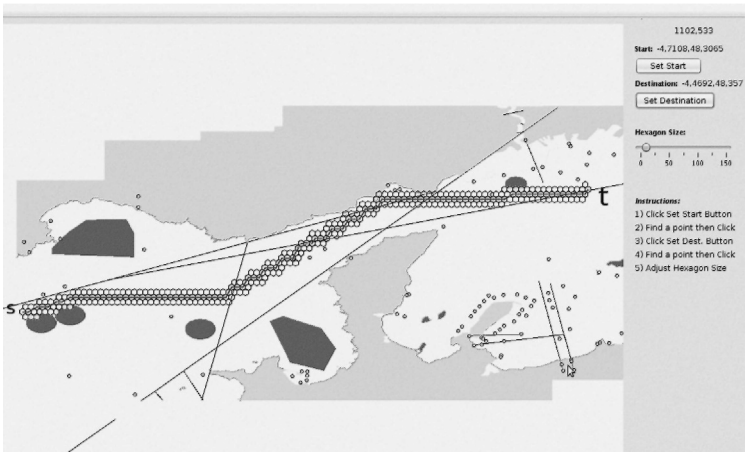



Figure 20. Example of maritime routing solution (black line) using the $BIDA^*$ algorithm on a dynamic data structure. Case 2: single resolution hexagonal mesh.

## 5. Conclusion

This paper introduces a spatial data structure applied to a real-time path-finding algorithm in open maritime environments. It is based on a hexagonal mesh and a dynamic data structure. This data structure is applied to a $BIDA^*$ algorithm in order to derive an optimal track in terms of distance and complexity. The proposed data structure has several properties that (i) determine a complementary hexagonal mesh producing a complete paving for the solution path and (ii) optimize the direction of propagation by following one of the three directions required by the branch-factor. This latter is implemented on a dynamic and hierarchical graph avoiding the duplication of graph nodes and the redundancy of information in the nodes. Furthermore, this data structure reduces computational time and memory space costs of the solution path that is relevant for real-time path-finding. In the context of real-time maritime navigation, one can use a bidirectional path-finding to have real-time foresights of the route to follow. Accordingly, the real-time track process can facilitate captain decision making during high stress or workload situations. A direction for future research is to integrate additional maritime navigation constraints like weather, meteorological information and the semantics generated by entities or other mobile objects around the ship (Tsatcha *et al.* 2013). All these constraints could further improve the proposed safety track.

## Acknowledgements

The authors acknowledge the three anonymous referees for their detailed and valuable comments which have improved this paper. We are also grateful to the SHOM (“Service Hydrographique et Océanographique de la Marine”, Brest, France) for their maritime data and support in this research.

## References

Ahmadi, S., Ebadi, H., and Valadan, Z., 2008. A new method for path finding of power transmission lines in geospatial information system using raster networks and minimum of mean algorithm. *World Applied Sciences Journal*, 3 (2), 269–277.

Borrmann, A., Hyvärinen, J., and Rank, E., 2009. Spatial Constraints in Collaborative Design Processes. *In*: *Proceedings of the International Conference on Intelligent Computing in Engineering (ICE'09). Berlin, Germany*, Berlin, Germany.

Cui, X. and Shi, H., 2011. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security (IJCSNS)*, 11 (1), 125–130.

Dantzig, G.B., 1960. On the shortest route through a network. *Management Science*, 6 (2), 187–190.

de Champeaux, D., 1983. Bidirectional heuristic search again. *Journal of the ACM (JACM)*, 30 (1), 22–32.

Dijsktra, E.W., 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1), 269–271.

Dillenburg, J.F. and Nelson, P.C., 1994. Perimeter search. *Artificial Intelligence*, 65 (1), 165–178.

Dreyfus, D., 1969. An appraisal of some shortest-path algorithms. *Operations research*, 17 (3), 395–412.

Freeman, J., 1975. The modelling of spatial relations. *Computer Graphics and Image Processing*, 4 (2), 156–171.

Kaindl, H. and Kainz, G., 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7, 283–317.

Korf, R.E., 1985. Iterative-deepening-A: an optimal admissible tree search. *In*: *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, Vol. 2 Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1034–1036.

Manzini, G., 1995. BIDA*: an improved perimeter search algorithm. *Artificial Intelligence*, 75 (2), 347–360.

Moore, E., 1957. The shortest path through a maze. *In*: *Proceedings of the International Symposium on Theory of Switching Cambridge* Harvard University Press.

Nicholson, T.A.J., 1966. Finding the shortest route between two points in a network. *The Computer Journal*, 9 (3), 275–280.

Pottmann, H., Brell-Cokcan, S., and Wallner, J., 2007a. Discrete surfaces for architectural design. *In*: P. Chenin, T. Lyche and L.L. Schumaker, eds. *Curves and Surfaces: Avignon 2006*. Nashboro Press.

Pottmann, H., *et al.*, 2007b. Geometry of multi-layer freeform structures for architecture. *ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2007*, 26 (3), article No. 65.

Pulido, F.J., Mandow, L., and Pérez de la Cruz, J.L., 2012. A two-phase bidirectionnal heuristic search algorithm. *In*: *Proceedings of the Sixth Starting AI Researcher's Symposium (STAIRS)*, 240–251.

Sahr, K., 2011. Hexagonal discrete global grid systems for geospatial computing. *Archives of Photogrammetry, Cartography and Remote Sensing*, 22, 363–376.

Sahr, K., White, D., and Kimerling, J.A., 2003. Geodesic discrete global grid systems. *Cartography and Geographic Information Science*, 30 (2), 121–134.

Schwalb, E. and Vila, L., 1998. Temporal constraints: a survey. *Constraints*, 3 (2-3), 129–149.

Sint, L. and de Champeaux, D., April 1977. An improved bidirectional heuristic search algorithm. *Journal of the ACM (JACM)*, 24 (2), 177–191.

Tong, X., *et al.*, 2013. Efficient encoding and spatial operation scheme for aperture 4 hexagonal discrete global grid system. *International Journal of Geographical Information Science*, 27 (5), 898–921.

Tsatcha, D., Saux, E., and Claramunt, C., 2013. A modeling approach for the extraction of semantic information from a maritime corpus. Advances in Geographic Information Science, *In*: S. Timpf and P. Laube, eds. *Advances in Spatial Data Handling*. Springer Berlin Heidelberg, 175–191.

U.S. Coast Guard, 1999. *Navigation rules: international-inland.* Technical report, U.S. Department of Transportation, United States Coast Guard, 2100 Second St., SW Washington, DC 20593-0001.

Xu, J., Lathrop, J., and G., R., 1994. Improving cost-path tracing in a raster data format. *Computers & Geosciences.*, 20 (10), 1455–1465.

Yap, P., 2002. Grid-based path-finding. *In*: *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, 44–45.

Yousefi, A. and Donohue, G.L., 2004. Temporal and spatial distribution of airspace complexity for air traffic controller workload-based sectorization. *In*: *Proceedings of the 4th American Institute of Aeronautics and Astronautics (AIAA) Aviation Technology, Integration and Operations Forum, Chicago, IL.*

Yu, C., Lee, J., and Munro-Stasuik, M.J., 2003. Extensions to least-cost path algorithms for roadway planning. *International Journal of Geographical Information Science*, 17 (4), 361–376.

## Appendix A. Maritime Path-Finding Algorithm based on $BIDA^*$

**Algorithm** *A.1*:

**Input** Starting point $s$ and terminal point $t$: two geographical positions.
$r$: the radius of a hexagonal cell.
$Map$: the set of geographical features representing the environment.
**Output** $Path$: the set of nodes (i.e. vertex and its indexes) defining the solution path.
**function** $BIDA^*(s, t, r, Map)$ ▷ The main bidirectional path-finding function based on two threads.
  $Mesh^1, Mesh^2, Stack1_1, Stack1_2, Stack2_1, Stack2_2, Path_1, Path_2 \leftarrow \varnothing$
  $Cell, Pnode \leftarrow \varnothing$
  $ObstacleArea \leftarrow selectFeaturesFromMap(Map)$ ▷ This procedure selects the features/objects to avoid during the navigation (e.g. restricted areas, hazard areas, buoys, etc.).
  **while** $joinMeshes(Mesh^1, Mesh^2) \neq true$ **do**
    **if** $currentThread() = T_1$ **then**
      $Cell \leftarrow updateMesh(T_1, s, t, r, Mesh^1, Stack1_1, Stack1_2, Mesh^2)$
      $Pnode \leftarrow parentNode(Cell)$
      $Path_1 \leftarrow Path_1 \cup Pnode$
      $wait(tip)$
    **else if** $currentThread() = T_2$ **then**
      $Cell \leftarrow updateMesh(T_2, s, t, r, Mesh^2, Stack2_1, Stack2_2, Mesh^1)$
      $Pnode \leftarrow parentNode(Cell)$ ▷ Update the indexes.
      $Path_2 \leftarrow Path_2 \cup Pnode$
      $wait(tip)$
    **end if**
  **end while**
  $Path \leftarrow mergingPath(Path_1, Path_2)$
  **return** $Path$ ▷ Return the optimal path.
**end function**
**Input** $T_j$: the thread to process.
$s$ and $t$: two geographical positions.
$r$: the radius of a hexagonal cell.
**Input** and
**Output** $Mesh^j$: a directional mesh.
$Stack_1$: the stack of selected nodes for the solution path.
$Stack_2$: the stack of visited nodes and evaluated using function $f = g + h$. The best promising node is at the top of the stack.
**function** $updateMesh(T_j, s, t, r, Mesh^j, Stack_1, Stack_2, Mesh^i)$ ▷ This procedure returns the best cell in $Mesh^j$ in the path-finding search and updates the outputs.

$Cell \leftarrow \varnothing$
**if** $Mesh^j = \varnothing$ **then**
**if** $currentThread() = T_1$ **then**
$N^{(0,0)} \leftarrow buildFirstNode(s, r)$ ▷ $s$ is the centre of the first cell and $N^{(0,0)}$ is the first node of the forward front (Figure 11).
**else**
$N^{(0,0)} \leftarrow buildLastNode(s, t, r)$ ▷ The function computes the centre $b$ of the last node using the formula presented in section 3.1 and generates the first node of the backward front.
**end if**
$pushStack(N^{(0,0)}, Stack_2)$ ▷ Put the first node at the top of the stack of the visited nodes.
**else**
**if** $Stack_2 \neq \varnothing$ **then**
$N^{(i,j)} \leftarrow popStack(Stack_2)$ ▷ This function selects the best node at the top of the stack.
**if** $positionSelected(N^{(i,j)}, Stack_1) = true)$ **then** ▷ This function checks if node $N^{(i,j)}$ corresponds to a geographical position or space that has already been selected and stored in $Stack_1$
**return** $\varnothing$
**end if**
**if** $(meetObstacle(N^{(i,j)}, ObstacleArea) = true)$ **then**
**return** $\varnothing$
**else**
$PartialCell \leftarrow extractRelationsFromMesh(Mesh^j, N^{(i,j)})$ ▷ Centred on $N^{(i,j)}$, the function extracts the set of vertices (and their relations) already built in $Mesh^j$. For the first node $N^{(0,0)}$, this function only computes the first hexagonal cell ${}^{0}_{0}[Hex(0,0)]^{0}_{1,2,3,4,5,6}$.
$Cell \leftarrow buildCell(PartialCell, N^{(i,j)}, Mesh^j)$ ▷ Computes the cell associated to $N^{(i,j)}$ from PartialCell and its complementary (i.e. the vertices of $N^{(i,j)}$ not yet defined) (Figure 15).
$storeCell(Cell, Mesh^j)$
$V_{\vec{i}} \leftarrow optimalDirection(Mesh^j, Mesh^i)$
$Beam \leftarrow generateThreeBestNodes(Cell, V_{\vec{i}})$
$pushStack(sort(Beam), Stack_2)$
$pushStack(N^{(i,j)}, Stack_1)$
**return** $Cell$
**end if**
**end if**
**end if**
**end function**