THE JOURNAL OF NAVIGATION (2003), **56**, 371–384. © The Royal Institute of Navigation
DOI: 10.1017/S0373463303002418 Printed in the United Kingdom

A Method for Searching Optimal Routes with Collision Avoidance on Raster Charts

Ki-Yin Chang and Gene Eu Jan
(*National Taiwan Ocean University*, *Taiwan*)
Ian Parberry
(*University of North Texas*, *USA*)

Collision avoidance is an intensive discussion issue for navigation safety. This article introduces a new routing algorithm for finding optimal routes with collision detection and avoidance on raster charts or planes. After the required data structure of the raster chart is initialized, the maze routing algorithm is applied to obtain the particular route of each ship. Those ships that have potential to collide will be detected by simulating the particular routes with ship domains. The collision avoidance scheme can be achieved by using the collision-area-marking method with collision avoidance rules at sea. The algorithm has linear time and space complexities, and is sufficiently fast to perform real-time routing on the raster charts.

KEY WORDS

1. Charts. 2. Raster charts. 3. Maze routing. 4. Optimal route.

1. INTRODUCTION. The technology of Electronic Navigational Charts (ENC) and digital electronic maps (DEM) has been used increasingly in marine navigation, GPS applications and geographic information systems. Reduced manning levels, the lack of skilled maritime labour and new satellite navigation systems have created a need for electronic navigational charts in merchant ships (Beattie, 1995). In the ENC system, the most common application is to determine an optimal or a most economical route (path) leading from any source (start) cell (point, port) to the destination cell without crossing any landmass (also called obstacle or barrier) including shoals. There are two kinds of ENC formats; those are raster and vector data. Since the raster charts are cheaper and simpler to produce and update than official vector charts, officially produced raster charts now cover most of paper charts (Dawson, 1997). The major drawback of the raster charts is that due to lack of adequate information, their emulation and implementation for ocean shipping is limited. To enhance the capability of the raster chart, some vector functions are emulated and implemented in the raster chart systems. But, the vector emulation is not sufficient for navigating in a sophisticated landmass and shoal areas. It is also clear that in the future ENC data with Automatic Identification Systems (AIS) will become widely available for vessels. Therefore, the more highly intelligent path

searching and collision avoidance systems on the charts are imminently required for safe navigation (Norris, 1998).

The efficiency of the algorithms is an important issue when searching an optimal route in the raster chart that is composed of a million pixels (cells). The most common resources required during computation are *time* (how many steps does it take to solve a problem) and *space* (how much memory does it take to solve a problem) complexities. Complexity differs from computability, which deals with whether a problem can be solved at all, regardless of the resources required. The big O notation is useful for the analysis of algorithm complexity since it captures the asymptotic growth pattern of functions and ignores the constant multiple (which is out of control anyway when algorithms are translated into programs). The time complexity of a problem is the number of steps that it takes to solve an instance. If an instance that has N number of cells can be solved in N^2 steps, then we say it has a time complexity of $O(N^2)$ in the worst case. Thus, an algorithm that has linear time complexity, $O(N)$, or time complexity of $O(N^2)$ makes a large difference if the size of the data is huge.

Many earlier works in the shortest path problem were related to Dijkstra's (1959) single-source shortest path algorithm in the Euclidean plane that is dependent on the shape of obstacles (Viegas and Hansen, 1985; Chen and Ramanan, 1991; Fagerholt *et al.*, 2000). Among them, Fagerholt presented a vector model for solving a shortest path with obstacles and to implement its application to ocean shipping. Their worst-case time complexity was $O(N^2)$. Some efficient implementations of Dijkstra's single-source shortest path algorithm proposed using Fibonacci heaps (F-heaps) to find the shortest path in the graph model (Ahuja *et al.*, 1990; Henzinger *et al.*, 1997). But, in a raster plane the situation is naturally presented in a grid plane. In order to apply the vector-based scheme in the graph model there has to be extensive pre-processing to convert the situation to adjacency matrices in order to apply the algorithm. For a grid with N cells, the implementations use up to N F-heaps and the total complexity, including pre-processing, has not been improved to $O(N)$. Furthermore, all of the vector-based schemes are still poor at solving cases involving complicated concave obstacles.

To work out this problem efficiently, this paper proposes a new maze routing algorithm on the raster charts. The algorithm improves the 2-*geometry* maze routing to a higher geometry (4-*geometry*, 8-*geometry*, 16-*geometry*, etc.) maze routing. Our algorithm solves this problem by using the suitable data structures to perform uniform wave propagation and the correctness of which has been proven (Jan and Chang, 2002). Several similar uniform wave propagation methods arise in the field of pattern recognition (Kimmel *et al.*, 1995) and computer-aided design (Xing and Kao, 2002). But, their method is not suitable for applying in the raster plane. The application of our algorithm is naturally favourable for a pixel-based plane such as the raster charts since the algorithm is developed in a grid plane. Furthermore, the algorithm is a directional improvement of the 2-*geometry* maze routing algorithm and it therefore inherits the two main advantages of the 2-*geometry* algorithm, such that it is independent of (the shapes of) the obstacles and guarantees to find the shortest path if one exists. The algorithm is similar to the breadth-first search algorithm until the time of arrival of the planar cells is completed or the given condition is reached. The computation for searching the optimal route is not as extensive as the graph scheme since the problem is independent of the shapes of the obstacles and no pre-processing effort is required to construct a suitable search structure. In addition, the algorithm can be applied to find the optimal route in the varied terrain of a raster chart. Influence factors that can be considered as varied terrain that can affect navigational tracks include ocean currents, and safety and weather conditions.

If multiple ships sail in the same area, the collision detection and avoidance method is necessary to search for their optimal routes. Several researchers in this field have done some work in computerized collision avoidance. Davis (1982) used Goodwin's domain in simulation of multi-ship encounters. Colley (1984) designed a circle to circumscribe Goodwin's model and simulated marine traffic flow and collision avoidance, which is based on the concept of 'range to domain over range rate' (RDRR). Coenen (1989) introduced an expert and knowledge-based system to assist the progress of collision avoidance. Most of this research work was emulated by using direct vector functions in the vector (format) plane. But, our routing algorithm adopts a pixel-based scheme in the raster plane for application. The algorithm can easily handle multiple ships navigating in the same area since the cost function of the algorithm is represented by the time of arrival; such that the arrival time of each cell in the desired path has been determined. The collision detection and avoidance method for those ships can be achieved by simulating the path of multiple ships with their ship domains and space marking of the collision area. The main difference between the vector method and ours is that once it decides to take action on the alternation of a course for a specific ship, our algorithm not only precisely avoids possible collisions, but also searches an optimal route for the give-way ship.

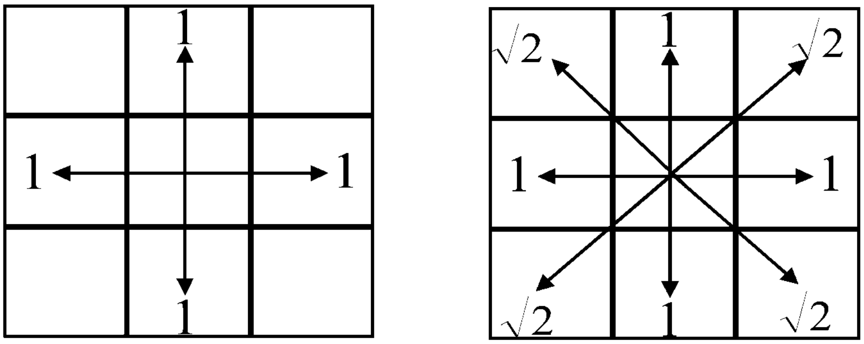
The rest of this paper is organized as follows: Section 2 describes the maze routing algorithm and its required data structures; implementation of collision detection with ship domains and collision avoidance scheme is discussed in Section 3; some application examples are shown in Section 4 and finally, the conclusions are presented in Section 5.

2. THE MAZE ROUTING ALGORITHM.

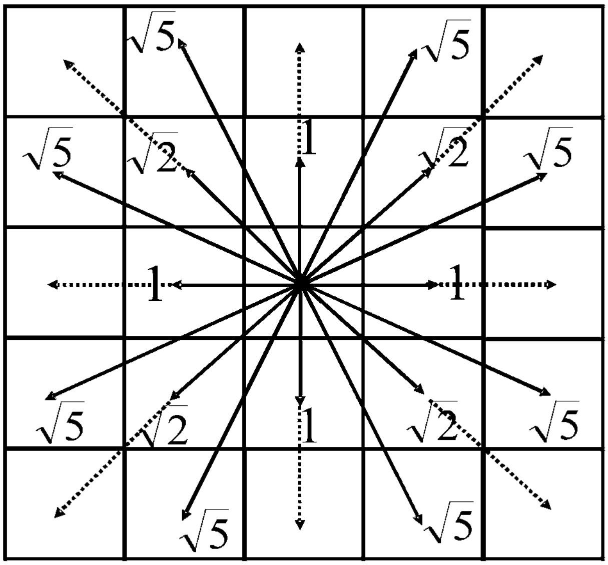
First, an overview of the original 2-*geometry* maze routing algorithm is introduced. Following that, the λ-*geometry* that is the general form of our algorithm is described. The required data structure and the algorithm will be described in the 4-*geometry* that is convenient for the reader to understand the concept of the algorithm. Further, the varied terrain problem and the 8-*geometry* or higher geometry, is discussed.

2.1. *The 2-geometry maze routing algorithm.*

The original goal of maze routing problems is to find a shortest path between a given pair of cells on a rectangular grid of cells without crossing any obstacles. The maze routing algorithm was first presented by Lee (1961). To this date, Lee's algorithm and its variations are probably the most widely used maze routing methods, with application in maze games (Rubin, 1974; Hoel, 1976), in VLSI design (Lin *et al.*, 1990; Ercal and Lee, 1997) and in road map routing problems (Fawcett and Robinson, 2000). The popularity of Lee's algorithm lies in its simplicity and the guarantee to find a shortest path if one exists. However, Lee's algorithm is intrinsically based on the 2-*geometry* (also known as the *Manhattan geometry*, *rectilinear geometry*, etc.) of the grid plane. Each cell is considered to have only four neighbours, corresponding to, at most, four directions (left, right, up and down) to move along an admissible path as shown in Figure 1(a). There were numerous works that have been presented to improve the performance of the 2-*geometry* maze routing algorithm, but none of them were able to improve it to a higher geometry, such as 4-*geometry*, 8-*geometry*, 16-*geometry*, etc. The procedure of



(a) A 2-*geometry* neighbourhood. (b) A 4-*geometry* neighbourhood.



(c) An 8-*geometry* neighbourhood.

Figure 1. Cell connection styles.

Lee's algorithm can be described as a wave propagation process. Two queues *plist* and *nlist* are defined to keep track of the cells on the wavefront (also called frontier cells) and their equal-distance step neighbouring cells respectively. Putting the source cell in queue *plist* initializes the search. After all the neighbouring cells in *plist* are included in *nlist*, the queue *nlist* is processed so that an expanded wavefront is found. Then any cell in *plist* is deleted if all of its neighbouring cells have been processed (updated) and *plist* is updated by this new wavefront. The search is terminated if the destination cell is found. Using a simple induction can prove that the path taken by the router is the quickest route from the source.

2.2. *The λ-geometry.* For an $m \times n$ rectangular grid of N cells, a *cells map* has a finite set of values and indicates, at the very least, which cells constitute obstacles. Each cell represents a pixel in the raster plane. Thus, there should not be any significant distortion between the original geometry map and the raster chart because the size and the shape of the cells are exactly the same as the pixels. The *λ-geometry* allows edges with angles $i\pi/\lambda$, for all i, $\lambda = 2, 4, 8, 16$ and ∞ correspond to rectilinear (90°), 45°, 22·5°, 11·25° and Euclidean geometries respectively (Sherwani, 1999). For a 2-*geometry* neighbourhood, we are only interested in the cells above, below, to the left and to the right of a cell, that is, all of the cells that are distance 1 unit from the centre cell as shown in Figure 1(a). If the four cells on the diagonal are included, we are working in a 4-*geometry*, also called 45° *geometry*, neighbourhood. For a 4-*geometry* neighbourhood, we are interested in eight cells that have distances of 1 and $\sqrt{2}$ units from the centre cell as shown in Figure 1(b).

In the 8-*geometry* neighbourhood, each cell has 24 related neighbours. The distance of the 24-cell connected neighbourhoods to the centre cell is shown in Figure 1(c). There are 16 solid-line reachable neighbours required to compute for each move. The 8 dashed-line reachable neighbours do not need to be calculated since they can be extended by the next computation and will be computed in the next move. Note that each divided angle in the 8-*geometry* is not exactly, 22·5°, a half of 45° because the angle is divided in the grid plane, not in a circle. For the 16-*geometry* neighbourhood, the running time for computation and condition statements is about twice that of the 8-*geometry*. But, the 16-*geometry* routing has twice the selective directions for searching a shortest path than that of the 8-*geometry*.

Lee's algorithm fails for a higher geometry if different distances occur. A straightforward attempt to improve Lee's algorithm by using equal cost wavefronts causes a substantial increase in time complexity (Fawcett and Robinson, 2000). Our algorithm works in a general context and uses a different data structure than that of Lee's algorithm. But in the case of the 2-*geometry*, the two algorithms are the same. Our algorithm has the same time and space complexities of $O(N)$ as Lee's algorithm, where N is the number of cells in the grid plane. It is worth mentioning that although we focus on the 4-*geometry* in this article, the algorithm works in more general situations. It can be easily adapted to handle general *λ-geometry* for $\lambda > 4$, and it can be used without substantial modification for higher dimensions.

2.3. *The required data structures.* The required data structure for the 4-*geometry* maze routing algorithm includes a cell map, some buckets and lists, and particular routes. The number of data fields in the cell map for collision avoidance implementation require at least four parameters for cell storage, that is, *SL*, *AT*, *SD* and *Vis*. The *SL* (*Sea or Land*) parameter distinguishes whether a cell is a landmass in which case the value is infinity, or a navigable area in which case the value is one. If the *SL* parameter of any area has a finite value that is not equal to one, we are working on the optimal path of the varied terrain. The *AT*, time of arrival, parameter stores the time needed to travel from the source cell to the current cell and its initial value is infinity. The *SD* (*Ship Domain*) parameter records whether this cell belongs to one of the ship domains. The fourth parameter *Vis* (*Visited*) distinguishes whether the cell has been visited and its initial Boolean value is *false*. The initial cell conditions are illustrated in Figure 2, where the black cells represent landmass and the white cells represent navigable areas.

In an $m \times n$ grid of N cells, any cell c_i has four parameters SL_i, AT_i, SD_i and Vis_i, where $0 \leqslant i \leqslant N-1$ and $N = m \times n$. To reduce the memory space, the buckets are replaced with three circular buckets. The three buckets are marked as LL_{bucket_index}, where the *bucket_index* is an integer variable and $0 \leqslant bucket\_index \leqslant 2$.

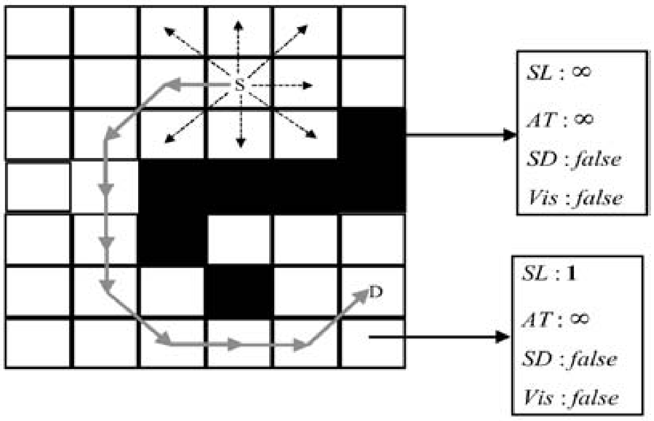


Figure 2. The cell map.

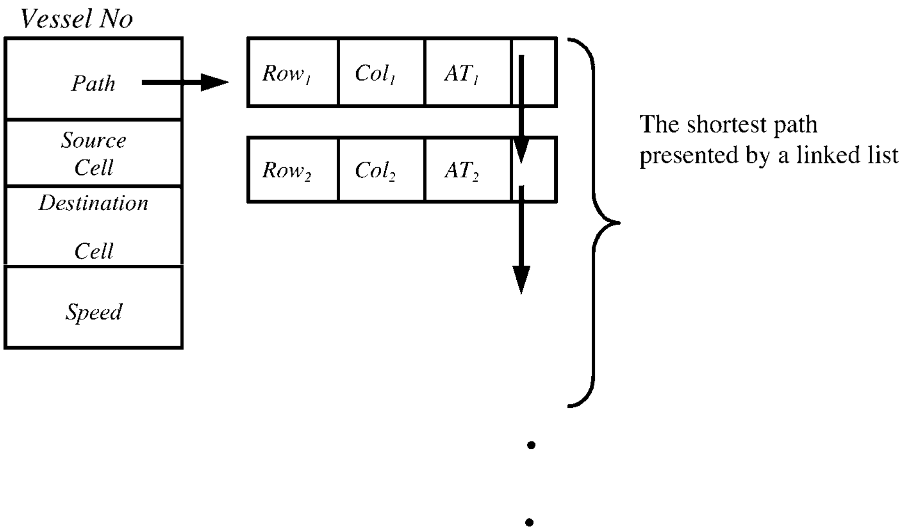


Figure 3. The data structure of a particular route.

The data structure of a particular route for this algorithm includes the particular route (or vessel path), the source cell, the destination cell, the speed, etc. as shown in Figure 3. The speed field stores the vessel speed, which is the number of cells travelled by the vessel per unit time. The vessel path is stored in a list in which each node represents a cell in the path. Each node of the list for the path contains four fields, *Row*, *Col*, *AT* and *Next*. The *Row* and *Col* fields store each cell's coordinate that is extracted from index *i*. The *AT* field stores the AT_i value of the cell c_i. The *Next* field is a link to the next node. For example, if a vessel travels from the source cell, located at (2, 2), to the destination cell (4, 3) through the cell (3, 2), the data fields (*Row*, *Col*, *AT*) in the first node of its linked list can be represented as (3, 2, 1). By modifying the value of the *AT* field in the list of the path, we can adjust the vessel speed. To illustrate multiple ships with different speeds on one chart, the *AT* field values should be modified. The method for the speed-varying system is to update the AT_i value for each node in the list, which is the value of the AT field divided by the value of the speed field.

2.4. *The 4-geometry maze routing algorithm.* The 4-*geometry* maze routing algorithm is described in a general form of one terrain. This is to simplify our presentation and make it easy to point out the main concept of the algorithm. The detailed description of the 4-*geometry* maze algorithm is shown in Appendix A.

There are some distinguishable aspects between the 4-*geometry* algorithm and the other 2-*geometry* algorithms. First, buckets are introduced to control the propagation speed of several single-step distances (or cost functions), which are non-equal to a uniform propagation without a sorting process. Second, the *Vis* parameter is introduced to make sure each cell is inserted into a temporary list (*temp-list*) and later removed from its corresponding bucket exactly once. Due to these two aspects, the algorithm keeps a linear time complexity. Furthermore, the *SL* parameter is introduced for the varied terrain problem, in which the speed of the vessel for those cells in the specific terrain is divided by the value of its *SL* parameter. The uniform wave propagation for varied terrain can be achieved by increasing a certain number of buckets.

For the 8-*geometry* maze router, the 4-*geometry* maze router can be simply modified to increase the number of circular buckets to four for the $\sqrt{5}$ step increment and add one more parameter *Dir* to keep track of which predecessor causes the minimum AT value. The 8-*geometry* uses, at most, twice the time of the 4-*geometry*. Thus, for the higher geometry, the number of circular buckets and condition statements are increased. The running time is $O(\lambda N)$ for the λ-*geometry* algorithm.

As a whole, the algorithm is capable of performing the various speeds for multiple ships and finding optimal routes for varied terrain in the raster plane. But the collision avoidance and detection scheme is required for finding the optimal routes of multiple ships. If there is only one terrain in the raster plane, the optimal route found by the maze routing algorithm is also called the shortest path of the algorithm.

3. METHOD FOR COLLISION DETECTION AND AVOIDANCE.

3.1. *Domain representation.* The international regulations for preventing collisions at sea, published by IMO (International Maritime Organization) (Cockcroft and Lameijer, 1996), are the standard to govern the behaviour for navigational collision avoidance. According to rule 8(d) of the international regulations for preventing collisions at sea, ships should keep a safe distance while collision avoidance action is taken. In general, the safe distance is defined by the concept of ship domain.

The concept of the ship domain was first presented by Fujii and others (1971). The original purpose of this model was to analyze the traffic capacity in a particular waterway or sea area. The model of his domain is an ellipse where the horizontal axis is 3·2 times the ship length and the vertical axis is 8 times the ship length. Further, Goodwin (1975) observed the required distance and considered the international regulations for preventing collisions at sea while ships are being navigated in open water. He obtained a non-symmetric ship domain that is divided by 112·5° from the central ship for port and starboard sides and three sectors with different radii. These three sector domains were used to illustrate the difference in risk. In the eighties, Coldwell (1983) established models of ship domains for end-on encounters and overtaking situations in restricted waters. Since then, the theory and models of ship

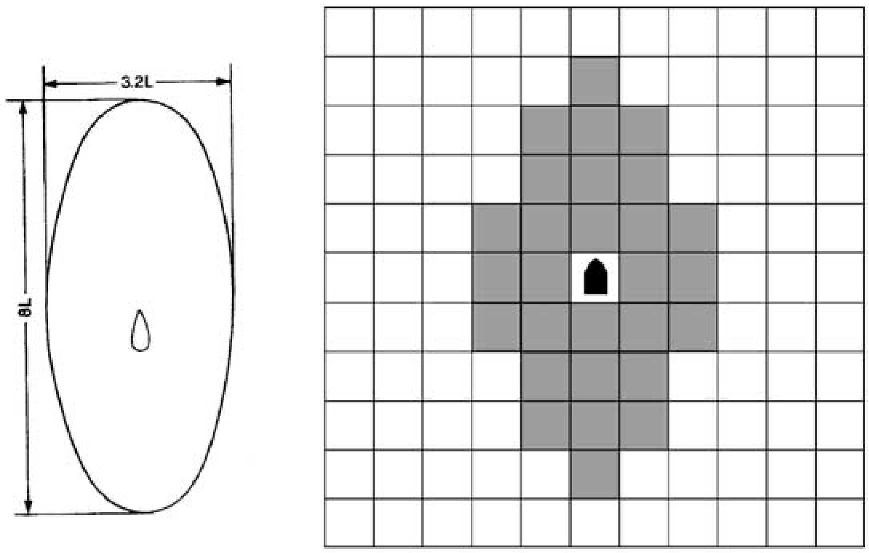


Figure 4. Fujii's domain and its basic representation (if a ship is considered as one cell) in the grid plane.

domains have been widely used in ship collision avoidance and marine traffic simulation, estimation of encounters rates, traffic lane design criteria, etc. Generally speaking, the ship domains of Fujii and Coldwell are suitable for restricted water areas where the traffic density in the passage is considered. Goodwin's model is suitable for the open water area, where the navigator has more effective sea room and safety is the first priority (Zhao, 1993).

To simplify the ship domain described in this paper, two basic models, Fujii's and Goodwin's model, are switched in the restricted water and the open water, respectively. If a ship is considered as a cell on the raster chart, the basic representation of both ship domains can be filled black as shown in Figures 4 and 5. While a ship configuration is considered as the combination of many cells in the raster plane, the model of the ship domain will be much closer to the situations in reality. If a ship is considered as an object on the raster chart, virtually expanding the obstacles (landmasses) are required before implementation of the routing algorithm. The virtual obstacles can be expanded by treating each boundary cell of obstacles as a special case of source cell with limited propagation radius (Jan *et al.*, 2003).

3.2. *Method for collision detection.* There is one necessary and sufficient condition that ships may collide at sea; that is, those two or more ships appear in one area at the same time. Thus, the path position and time domain of each ship is required for simulation. After the shortest path of each ship is obtained by the 4-*geometry* maze router, the possible collision area will be detected by simulating their particular routes with ship domains. To simulate the possible collision situation, the path lists of each ship are moved from cell to cell according to their *AT* value in the particular route. The simulation is divided into three steps: (1) Remove the previous ship domain, (2) Move each ship to the next cell according to its corresponding *AT* value, (3) Mark a new ship domain for the next cell. From these steps, the particular route of each ship is simulated by marking her ship domain from cell to cell. If the *SD* value of the

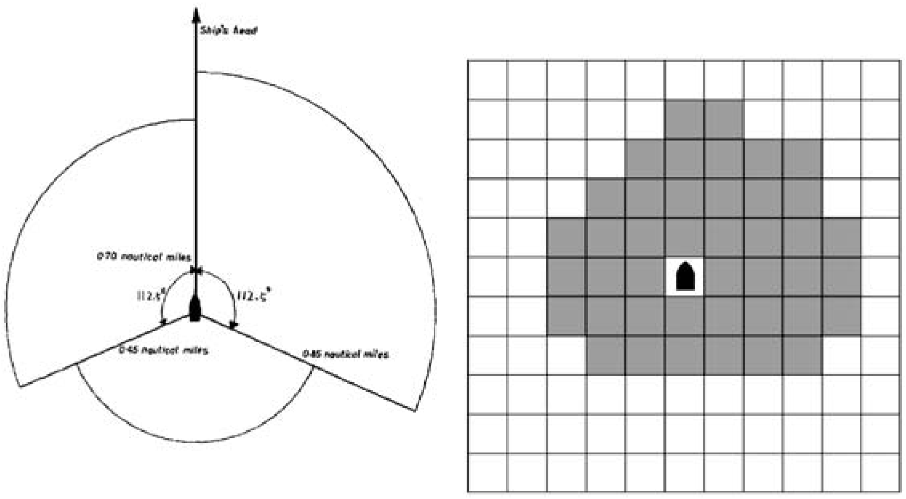
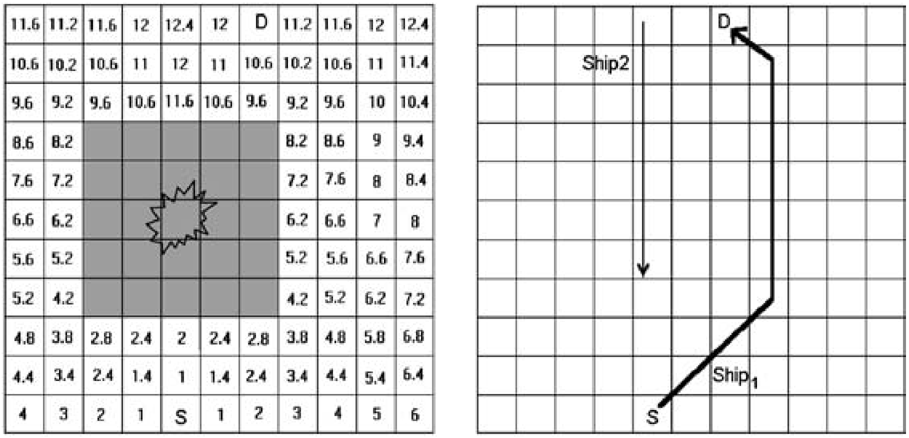


Figure 5. Goodwin's domain and its basic representation (if a ship is considered as one cell) in the grid plane.

cell has been marked by another ship domain, then they may collide in this area and the collision avoidance process is required.

3.3. *Collision avoidance scheme*. To avoid the collision, alteration of course and/or speed must be large enough to be readily apparent to another ship observing visually or by radar. According to rules 8 (c) and (e) of the international regulations for preventing collision at sea, if there is sufficient sea room, alteration of course alone may be the most effective countermeasure to avoid collision. Only if it is necessary to avoid a collision or allow more time to assess the situation, shall a ship reduce her speed or take all way off by stopping or reversing her means of propulsion. In fact, reducing or increasing speed is not very effective for collision avoidance since a moving power-driven ship has an inertial momentum. Based on the above assumption, this paper only considers the alteration of course to avoid collision. If reduction or increase of speed were to be included, an assist from the knowledge-based collision avoidance system (Coenen, 1989) would be required to ensure no collisions occur.

After the collision detection is simulated, the multi-ship collision avoidance scheme is implemented by the collision-area-marking method in the raster plane. The possible collision area that is considered as an impassable area or landmass is called the collision-area marking method. The algorithm then recalculates and obtains the optimal route of the give-way ship by taking a detour around the impassable area. The shortest path with collision avoidance scheme for multiple ships is summarized as several steps. After the maze routing algorithm is applied to obtain the shortest path of each ship, the next step is to simulate those ships by marking ship domains from time to time. If a collision is detected, then the rule will decide which ship needs to alter course. If a Ship_1 from the S (source cell) is to alter course, then Ship_1 marks the collision area as an impassable area. After that, the Ship_1 route is recomputed by the shortest path program, the AT value of each cell determines a new optimal route that detours around the impassable area as shown in Figure 6(a). Therefore, the Ship_1 follows the new optimal route and takes action in advance to alter course, so that the



11.6	11.2	11.6	12	12.4	12	D	11.2	11.6	12	12.4
10.6	10.2	10.6	11	12	11	10.6	10.2	10.6	11	11.4
9.6	9.2	9.6	10.6	11.6	10.6	9.6	9.2	9.6	10	10.4
8.6	8.2						8.2	8.6	9	9.4
7.6	7.2						7.2	7.6	8	8.4
6.6	6.2						6.2	6.6	7	8
5.6	5.2						5.2	5.6	6.6	7.6
5.2	4.2						4.2	5.2	6.2	7.2
4.8	3.8	2.8	2.4	2	2.4	2.8	3.8	4.8	5.8	6.8
4.4	3.4	2.4	1.4	1	1.4	2.4	3.4	4.4	5.4	6.4
4	3	2	1	S	1	2	3	4	5	6

Figure 6. A collision-area-marking method for collision avoidance.

give-way ship, Ship$_1$, and stand-on ship, Ship$_2$, have ample time to keep a safe passing distance as shown in Figure 6(b). To ensure the new optimal route does not collide with other ships, the re-simulation or routing with dynamic ship domains is required. The algorithm for multiple ships can be summarized in the following algorithm for the shortest path with collision avoidance scheme.

Algorithm for the shortest path with collision avoidance scheme:

Step 1: Acquisition of the shortest path for each ship.
For each Ship$_k$, we indicate the source cell S_k and destination cell D_k by the maze routing program and obtain the shortest path, where $1 \leqslant k \leqslant$ total number of ships, respectively.

Step 2: The collision detection.
Simulate all of the routes with ship domains and mark their *SD* value in the cell map from time to time.

Step 3: The collision avoidance scheme.
If any two ships have the same *SD* value in some cells, the rule of the international regulations for collision avoidance at sea decides which ship needs to alternate course and mark this area as impassable area for this ship.
- Step 3.1: Recompute the path for this give-way ship. (It is noticed that the recomputed path is no longer the shortest path, it is an optimal path.)
- Step 3.2: Simulate the new optimal path.

If the new optimal path collides with another ship, then erase the new optimal path and do the routing with dynamic ship domains.
Otherwise return, "Complete the shortest path or optimal path calculation".

END {Algorithm for the shortest path with collision avoidance scheme}

After the possible collision area is detected, the give-way ship recomputes the optimal path and considers the possible collision area as the impassable area or landmass. If more than two ships may collide in the same area, an order of priority is given to determine which ship takes action first. While routing with dynamic ship domains, the *AT* value of each cell is expanded (propagated) from the cell of the alternate

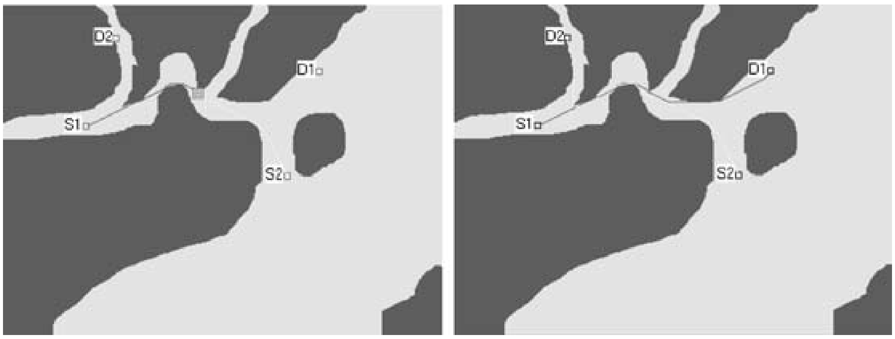


Figure 7. Illustration of the shortest path algorithm with collision avoidance scheme for two ships.

course and the domains of other ships are also moved according to their *AT* value, that is the uniform propagating cells have dynamic ship domains (*SD*) in the cell map. Thus, the collision can be avoided in the new route since those cells are considered as impassable at this specific *AT* value if the *AT* value of propagating cells in the bucket (considered as a time domain) match the domains of other ships. The routing with dynamic ship domains is useful for multi-ship sailing in the restricted water area. But, in the real world, the trajectory and speed of those ships may change with time. Thus, continual routing computation is required with dynamic ship domains. This is called real-time routing in the raster charts.

Regarding to the performance of this algorithm, step 1 obviously has time complexity of $O(qN)$, where q is the number of ships. Steps 2 and 3 also have the obvious time complexity of $O(qN)$. Thus, the algorithm has the total time complexity of $O(qN)$.

4. EXAMPLES OF APPLICATION AND DISCUSSION.

The search time for the shortest path with the collision avoidance algorithm was only a small fraction of one second on a 400×300 raster electronic chart with a Pentium III PC using DirectX. Assume that the size and speed of all ships are known for the applications. An execution result for two ships is illustrated in Figure 7. The source (expressed by *S*) and the destination cells (expressed by *D*) of two ships are illustrated. After finding the shortest path of each ship, a collision detection method is applied to simulate and spot the potential collision area as shown in Figure 7(a). Figure 7(b) indicates the recomputed optimal route presented by a curve after the collision avoidance scheme is implemented.

If there is a multi-ship simulation on a raster chart, an adjustment is made for the various speeds because multiple ships may have different or piecewise speeds. An example of the shortest path algorithm for five ships is illustrated in Figure 8.

Figure 8(a) indicates the source cells (expressed by *S*) and the destination cells (expressed by *D*) of five pairs. The shortest path algorithm would be called five times for the five ships to obtain the shortest paths. Figure 8(b) indicates the recomputed optimal and shortest path after collision detection and avoidance is implemented for the five ships. With multiple ships navigating in a restricted waterway, routing with dynamic ship domains is a feasible way to prevent collisions, but there may not be sufficient width in a restricted and crowded waterway to permit just the use of course

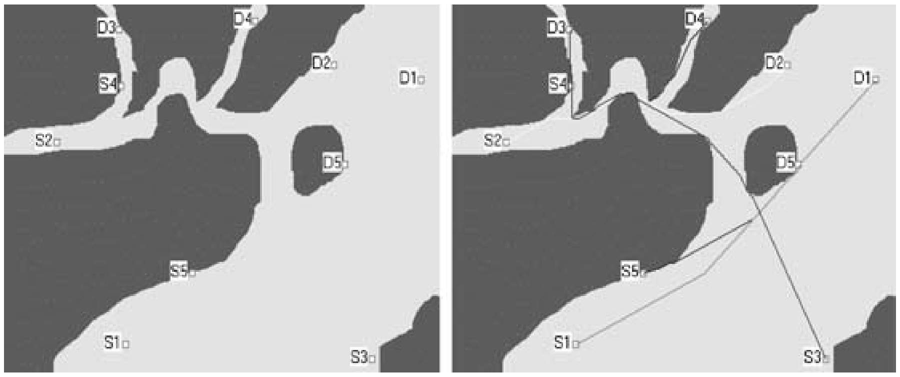


Figure 8. Illustration of the shortest path algorithm with collision avoidance scheme for five ships.

alterations. The assistance from a knowledge-based collision avoidance system would be required should it be necessary to apply speed variation to maintain safe passing distances.

5. CONCLUSIONS.

This paper presents a new and practical searching algorithm with collision avoidance for navigation on raster charts. The algorithm is also applicable for an ECDIS display on a raster plane. After the optimal route of each ship is obtained by carrying out the shortest path algorithm, the precise collision detection is simulated to verify whether or not ships using those optimal routes may collide. If a possible collision area is detected, the give-way ship recomputes an optimal route. Many of the factors that influence a navigational track can be considered as a varied terrain area on the chart. The algorithm is suitable for most navigational situations at sea. In addition, if AIS is widely used, the algorithm is capable of extending to a real-time routing system with the domains based on the actual size and speed of the target ships. This is an algorithm for a future path searching and intelligent collision avoidance system on raster charts.

The maze routing algorithm has $O(N)$ time and space complexities, and guarantees to find an optimal route if it exists. When q ships navigate in the same area, the algorithm has the time complexity of $O(qN)$. In future work, the algorithm will be extended to a search and interception system or target chasing system with a time-matching search scheme on raster electronic charts.

ACKNOWLEDGEMENTS

We would like to thank the National Science Council of Taiwan, ROC, for their financial support.

APPENDIX A

The input for the 4-*geometry* maze router is a cell map, the source (S) and destination (D). The cell c_j is the neighbour of the cell c_i. SL_j denotes if a cell c_j is an obstacle or free space in the cell map. Vis_j denotes whether or not a cell c_j is visited. AT_j denotes the time of arrival for a cell c_j from the source cell to the cell c_j. The linked list *temp-list* is used to keep track of cells on their neighbour cells. The three circular buckets

L_{bucket_index} keep a uniform propagation. The procedure INSERT, inserts the index of c_i into the *temp-list*. The procedure RETRACE, retraces from the destination cell to the source cell according to their smallest AT value and form a path LL_{path}, which is the output of the 4-*geometry* maze router. The formal description of the 4-*geometry* maze router is summarized below:

```
Algorithm 4-GEOMETRY-ROUTER (Cell-map, S, D, $LL_{path}$)
Input: Cell-map, S, D
Output: $LL_{path}$
begin
   $bucket\_index = 0$;
   $L_{bucket\_index} = S$;  $Vis_S = $ TRUE;
   $temp\text{-}list = \phi$;
   $path\text{-}exists = $ FALSE;
while ($L_{bucket\_index} \neq \phi$ or $L_{bucket\_index+1} \neq \phi$) do
  if D cell in $L_{bucket\_index}$ then
    {
     $path\text{-}exists = $ TRUE;
     break while;
    }
  for each cell $c_i$ in $L_{bucket\_index}$ do
  {
     for each cell $c_j$ neighbouring $c_i$ do
      {
         if $SL_j = 1$ then
        {
         if $Vis_j = $ FALSE then
          {
           $Vis_j = $ TRUE;
           INSERT($c_j$, temp-list);
          }
         Case 1: 2-geometry neighbours
               $AT_{new} = AT_i + 1$;
         Case 2: diagonal neighbours
               $AT_{new} = AT_i + \sqrt{2}$;
         if ($AT_{new} < AT_j$) then $AT_j = AT_{new}$
        }
      }
  }
  if $temp\text{-}list \neq \phi$ then
   for each cell $c_j$ in temp-list do
      INSERT($c_j$, $L_{floor(AT_j)\ mod 3}$);
  else $bucket\_index = (bucket\_index + 1) \bmod 3$;
end while;
if ($path\text{-}exists = $ TRUE) then RETRACE (Cell-map($AT_D$), $LL_{path}$);
else path does not exist;
end;
```

REFERENCES

[1] Ahuja, R. K., Mehlhorn, K., Orlin, J. B. and Tarjan, R. E. (1990). Faster algorithms for the shortest path problem. *Journal of ACM*, **37**, 213–223.

[2] Beattie, J. H. (1995). The future of electronic chart in merchant ships. This *Journal*, **48**, 335–348.

[3] Chen, Y. M. and Ramanan, P. (1991). Euclidean shortest paths in the presence of obstacles. *Networks*, **21**, 257–265.

[4] Cockcroft, A. N. and Lameijer, J. N. (1996). *A guide to the collision avoidance rules: International Regulations for Preventing Collisions at Sea*. (5th ed.): Oxford.

[5] Coenen, F. P., Smeaton, G. P. and Bole, A. G. (1989). Knowledge-based collision avoidance. This *Journal*, **42**, 107–116.

[6] Coldwell, T. G. (1983). Marine traffic behaviour in restricted waters. This *Journal*, **36**, 430–444.

[7] Colley, B. A., Curtis, R. G. and Stockel, C. T. (1984). A marine traffic flow and collision avoidance computer simulation. This *Journal*, **37**, 232–250.

[8] Davis, P. V., Dove, M. J. and Stockel, C. T. (1982). A computer simulation of multi-ship encounters. This *Journal*, **35**, 347–352.

[9] Dawson, J. (1997). Digital charting, now and in the future. This *Journal*, **52**, 251–255.

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

[11] Eracl, F. and Lee, H. C. (1997). Time-efficient maze routing algorithms on reconfigurable mesh architectures. *Journal of Parallel and Distributed Computing*, **44**, 133–140.

[12] Fagerholt, K., Heimdal, S. and Loktu, A. (2000). Shortest paths in the presence of obstacles: an application to ocean shipping. *Journal of the Operational Research Society*, **51**, 683–688.

[13] Fawcett, J. and Robinson, P. (2000). Adaptive routing for road traffic. *IEEE Comput. Graph. Appl.*, **20**, 46–53.

[14] Fujii, Y. and Tanaka, K. (1971). Traffic capacity. This *Journal*, **24**, 543–552.

[15] Goodwin, E. M. (1975). A statistical study of ship domains. This *Journal*, **28**, 328–344.

[16] Henzinger, M. R., Klein, P., Rao, S. and Subramanian, S. (1997). Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, **55**, 3–23.

[17] Hoel, J. H. (1976). Some variations of Lee's algorithm. *IEEE Trans. Comput.*, **c-25**, 19–24.

[18] Jan, G. E. and Chang, K. Y. (2002). An improved Lee's algorithm on electronic maps. *Proceeding of Int. Computer Symposium*, 776–786.

[19] Jan, G. E., Chang, K. Y. and Parberry, I. (2003). A new cell decomposition method to automatic path-planning for a mobile robot. *The Seventh Int. Conference on Automation Technology*, Sept. 2003.

[20] Kimmel, R., Amir, A. and Bruckstein, A. M. (1995). Finding shortest paths on surfaces using level sets propagation. *IEEE Trans. Pattern Anal. Machine Intell.* **17**, 635–640.

[21] Lee, C. Y. (1961). An algorithm for path connection and its applications. *IRE Trans. Electron. Comput.*, **EC-10**, 346–365.

[22] Lin, Y. L., Hsu, Y. C. and Tsai, F. S. (1990). Hybrid routing. *IEEE Trans. Computer-Aided Design*, **9**, 151–157.

[23] Norris, A. P. (1998). The status and future of the electronic chart. This *Journal*, **51**, 321–326.

[24] Rubin, F. (1974). The Lee path connection algorithm. *IEEE Trans. Comput.*, **c-23**, 907–914.

[25] Sherwani, N. A. (1999). *Algorithms for VLSI physical design automation*, 3rd ed. Boston: Kluwer Academic Publishers, 260–279.

[26] Viegas, J. and Hansen, P. (1985). Finding shortest paths in the plane in the presence of barriers to travel (for any l_p-norm). *European Journal of Operational Research*, **20**, 373–381.

[27] Xing, Z. and Kao, R. (2002). Shortest path search using tiles and piecewise linear cost propagation. *IEEE Trans. Computer-Aided Design*, **21**, 145–158.

[28] Zhao, J., Wu, Z. and Wang, F. (1993). Comments on ship domains. This *Journal*, **46**, 422–436.