THE JOURNAL OF NAVIGATION (2006), **59**, 27–42.
doi:10.1017/S0373463305003528 Printed in the United Kingdom

A New Method of Ship Routing on Raster Grids, with Turn Penalties and Collision Avoidance

Rafal Szlapczynski

(*Gdansk University of Technology*)
(Email: rafal@pg.gda.pl)

The article introduces a method of finding optimal routes on raster planes. The method presented takes advantage of a new algorithm that tends to minimize a number of direction changes within a route, while steering clear of the obstacles. Two different schemes, suitable for restricted area Vessel Traffic Service (VTS) system and collision avoidance system located on the own ship are described. The VTS-oriented scheme supports VTS priority policy that may extend or override international give-way regulations. The own-ship routing scheme in a give-way situation is capable of determining the shortest safe path to the destination point. The method takes into account own ship dynamics. It has linear time and space complexities and therefore is sufficiently fast to perform real-time routing on the raster grids. Both the general method and the algorithm it uses are presented in detail in the paper. Implementation issues are also discussed.

KEY WORDS

1. Navigation. 2. Optimal route. 3. Lee's algorithm. 4. Collision avoidance.

1. INTRODUCTION. Raster grids are a digital representation of planar data that is currently in use in a number of fields, including navigation. One of the most common operations to be performed on raster planes is to determine an optimal route between a start cell and a destination cell, which does not cross any obstacles (in sea navigation: landmass, barriers or shoals). An algorithm performing this should fulfill certain conditions. Of these the most obvious is, that it does indeed find an optimal route if such exists. Other conditions are those of time and memory space needed. In reality only algorithms of linear time and space complexities are useful as only such may be accepted by a real-time system processing large numbers of cells. The big O notation will be further used for complexities, with $O(n)$ indicating linear complexity.

The first solution to meet the conditions mentioned above was the maze routing algorithm presented by Lee [3], often described as wave propagation process. To date Lee's algorithm and its variations are among the most widely used routing methods, with applications in maze games, VLSI design and road map routing problems. However, the original algorithm proposed by Lee has one serious drawback: it works only for the 2-geometry grid plane (also known as the Manhattan geometry).

Only recently has it been upgraded to higher geometries while sustaining the linear time and space complexities. This new solution has been proposed by Chang, Jan and Parberry [1].

Despite the major progress, the potential use of the improved Lee's algorithm has still some limitations. Both the original algorithm and the upgraded version tend to find the shortest path, which is not always identical to the optimal one. In the presence of many obstacles the algorithms determine a route containing so many turning points and course alterations that no navigator would follow it, even when advised so by a real-time routing system. Also, the fall in speed during the manoeuvre, results in a longer passage time. Consequently, over larger geographic areas, both distance and number of turns will contribute to the total time spent traversing a route. Thus minimizing the number of turns is a desirable objective. Although the Chang, Jan and Parberry algorithm may cover the aspect of varied terrain (among other improvements), its data structure based on that of the original Lee's method makes it unable to include the cost of a turn in path length.

There are a number of methods (invented mostly for VLSI and automatics purposes) that cover the Minimum Bend Path and Shortest Minimum Bend Path problems. Unfortunately, determining the Shortest Minimum Bend Path is of no value for long distance sea routing. Instead, the objective is finding shortest paths with bend penalizing. Bend penalizing issue has also been considered in a number of works but the methods presented there are either not sufficiently fast or not applicable for higher geometries.

The article proposes the solution to this problem. A new data structure has been designed so as to reflect the cost of all course alterations in each cell's arrival time. An algorithm utilizing this structure has been implemented. The algorithm takes input parameters: user specified values of the course alterations time costs (penalties) and returns the determined path. A general method of ship routing with collision avoidance, based on this algorithm and alternative to the method of Chang, Jan and Parberry is introduced here. This method distinguishes two separate schemes for two use cases.

The first case is that the system (application of the method) is located on the own ship only. Given the source, destination and the raster chart of the area, the method determines the shortest route for the own ship. The route is followed until a potentially colliding target (another ship) is detected. Then, the give-way ship is determined according to the international regulations for preventing collisions at sea. If the own ship is to give way, a new route is determined in such a way that the two ship domains have no common cells at any time. Although usually the passage is regarded as safe if the own ship domain is not penetrated by the target ship, here a stronger condition is applied. The requirement that none of the cells may belong to both ship domains at the same time is an extra safety buffer.

The second case is that of the system located in the VTS centre, responsible for planning the routes for all ships within the certain restricted area. Given a priority policy (which may override the common regulations) the method determines the priority order of all ships in the region. Then, in turn from highest to lowest priority, for each ship, its shortest route is found in such a way, that the lower priority ship does not collide with any of the higher priority ships.

Section 2 is a presentation of a new routing algorithm that overcomes the limitations of the previous ones. In Section 3, a method for the system located on the own ship is presented. Section 4 describes a complementary method for the restricted

area VTS system that utilizes the same routing algorithm. Finally the conclusions are presented in Section 5. All use of the term 'algorithm' in the text refers to the algorithm introduced in section 2, unless stated otherwise. The term 'method' is later used for an application utilizing this algorithm.

2. ROUTING ALGORITHM WITH TURN PENALTIES. Since detailed description of both Lee algorithm and its upgraded version by Chang, Jan and Parberry is beyond the scope of this paper, it is assumed here that the reader is familiar with the Chang, Jan and Parberry article for this journal. For the reader's convenience the same notation as in Chang, Jan and Parberry paper is used here. To simplify the presentation, only the 4-geometry version is described. The algorithm in general, however, works also for higher geometries.

2.1. A concept of the turn penalty. In the paper it is assumed, that whenever a ship alters its course, the dynamics of this manoeuvre (the fall in speed in particular) results in a significantly longer passage time, than it would take the ship to cover the same distance without altering its course. This time difference, called the delay in the paper, should therefore be taken into account when determining a route. The fact of the course alteration manoeuvres being time consuming is the main reason for introducing a concept of the turn penalty. The other one is that the less complex routes (consisting of the lesser number of straight lines) may be preferred for safety reasons. Course alteration might be dangerous in the close presence of obstacles or misleading for other ships. Hence the idea of the turn penalty parameters in a routing algorithm. The turn penalty parameter value might be set separately for each course alteration angle by the system operator. The exact value of the parameter might be equal to the delay time of the dynamic manoeuvre or larger – to enforce determining less complex routes.

2.2. Data structure. The static data is stored in an array containing three fields for every cell:

SL (Sea/Land) – Integer number field, its value indicates whether a cell is sea or landmass (or any other static obstacle). Its value is 1 for sea, infinity for a landmass.

GAT (Gate Arrival Time) – A sub-array of the floating point numbers. The size of the array is equal to the maximum number of the neighbouring cells and is 8 for 4-geometry. Each field of this sub-array contains the gate arrival times for different incoming gates of the current cell. Gate arrival time is the time it takes to travel from the source cell to the current cell via certain neighbour of the current cell.

VIS (Visited) – Boolean field, its value indicates whether a cell has already been visited (inserted to *temp-list*) or not. True for a visited cell, false otherwise.

The dynamic data is stored in circularly used lists: $L_1 \ldots L_n$ and one extra temporary list *temp-list*. The number n of the lists that are used depends strictly on the maximum course alteration cost specified and thus is configured indirectly via algorithm input parameters.

$$n = ceiling\ (maximum\ \{single\text{-}step\ distances\} + maximum\ \{specified\ course\ alteration\ costs\} + 1),$$ where *maximum* {*single-step distances*} is $\sqrt{2}$ for 4-geometry.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

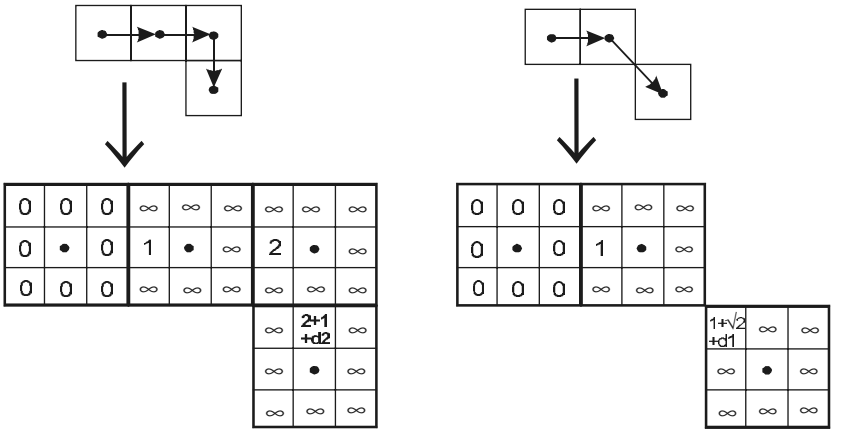

Figure 1. Left: A 90-degree turn. Right: A 45-degree turn.

2.3. Algorithm overview. The key difference between the new solution and that proposed by Chang, Jan and Parberry is a replacement of each cell's *AT* (Arrival Time) field with *GAT* array. The idea of incoming gates arrival times makes it possible to take into account course alteration costs without sacrificing the linear complexities that characterized the previous versions. Every time a cell data is updated, the arrival time of the appropriate gate is modified depending on the direction of the neighbouring cell that has initiated the update operation. The new candidate value of the gate arrival time field is determined according to the following formula:

$$GAT_{new,\ j,\ gate\_number} = \text{minimum}\ \{GAT_{i,1} + distance_{i,j} + delay_{gate\_number,1}, \ldots, GAT_{i,8} + distance_{i,j} + delay_{gate\_number,8}\},$$

where: i and j are indexes of the neighbouring cells, *gate_number* is the current gate of the c_j cell and numbers from 1 to 8 denote all gates of the c_i cell. Delay values (penalties) are equal to zero for two gates of the same direction and have appropriate parameter values d_1, d_2 or d_3 for two gates whose direction difference is 45, 90 or 135 degrees respectively. The present *GAT* value is replaced with the candidate value if the new value is lesser than the current one.

Figure 1 illustrates the way the *GAT* array values of the wave front cells are updated. A formal description of the algorithm is given in Appendix A.

2.4. Computational complexity. For each cell c_i, whose distance from the source cell is equal or lesser than that of the destination cell, the following actions are performed:

– each of its neighbours c_j is checked and possibly updated,
– for each of its neighbours c_j, each of the gates of the cell c_i is checked.

This gives a total of $(2*\lambda)*(2*\lambda)*n$ steps for the worst case, where λ is a constant denoting geometry level (eight neighbours and eight gates for 4-geometry) and n is the number of cells, whose distance from the source cell is equal or lesser than that of the destination cell. Thus the computational complexity of the proposed solution is $O(n)$.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

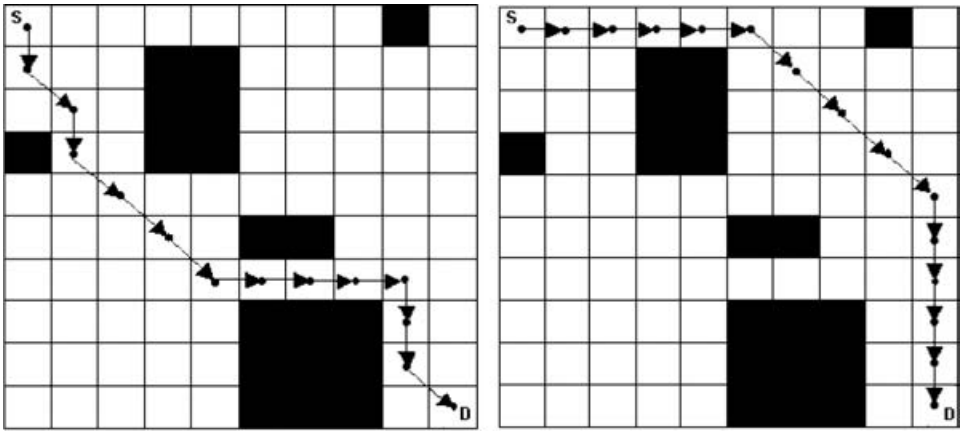

Figure 2. Left: Route determined by the Chang, Jan and Parberry algorithm. Right: Route determined by the proposed algorithm with direction change costs (penalties) d1 = 1, d2 = 2, d3 = 3.

2.5. Example results for proposed algorithm vs. Chang, Jan and Parberry method. Figure 2 presents examples of results for the new routing method and the Chang, Jan and Parberry algorithm. It has been assumed that costs of direction changes are: 1, 2 and 3 for 45, 90 and 135-degree turns respectively. In the figures, two different solutions for the same task are visualized. The proposed algorithm finds a route with two turning points as opposed to the Chang, Jan and Parberry method, which determines a route with six turning points. Thus the total path lengths and penalties for both methods are:

For the Chang, Jan and Parberry route:

$$\text{Basic path length} = 8 + 5 * \sqrt{2} \approx 15{\cdot}07$$
$$\text{Total penalties} = 5 * d_1 + 1 * d_2 = 7$$
$$\text{Total path cost} = \text{basic path length} + \text{total penalties} \approx 22{\cdot}07$$

For the proposed route:

$$\text{Basic path length} = 10 + 4 * \sqrt{2} \approx 15{\cdot}65$$
$$\text{Total penalties} = 2 * d_1 = 2$$
$$\text{Total path cost} = \text{basic path length} + \text{total penalties} \approx 17{\cdot}65$$

As illustrated above, taking a seemingly longer (3·8%) route results in a much lower (22%) overall path cost.

3. SHIP-ORIENTED ROUTING SCHEME WITH COLLISION AVOIDANCE.

The algorithm in the version described above is a general-purpose routing tool. When ship routing is considered however, collision avoidance rules must be applied. Whenever there are two potentially colliding ships, the international regulations for preventing collisions at sea determine which one is to give way to the other. This decision is made, depending on the positions and courses of

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

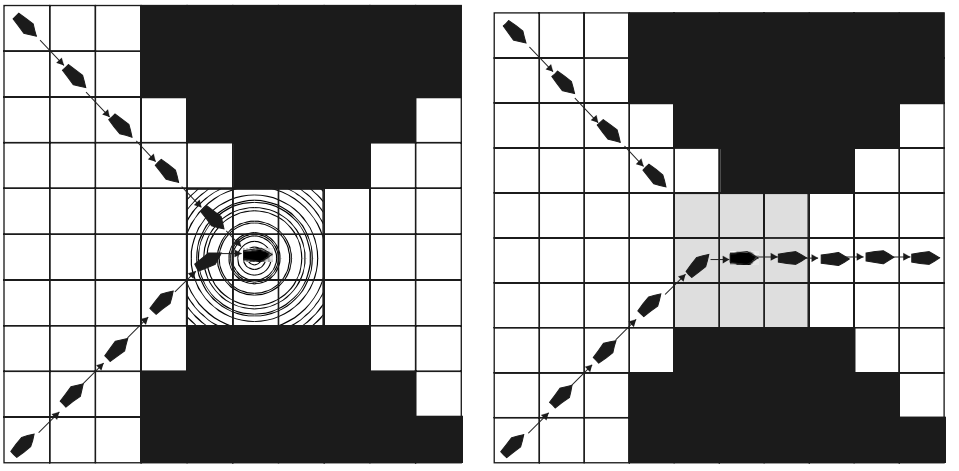

Figure 3. Left: Two ships potentially colliding. Right: An area marked as impassable for the second ship.

both ships. The resulting collision avoidance manoeuvre must fulfill the following conditions:

– the alteration of course must be large enough to be apparent to another ship – at least 15 degrees,
– the alteration of speed should only be applied if necessary, that is, if the alteration of course alone does not guarantee safe passage or the course alteration necessary is too large to be accepted (economical reasons),
– ships should keep a safe distance, while collision avoidance action is taken.

All three requirements are met in the method presented in this section. To fulfill the third condition, the domain conception has been used. Usually the route is considered to be safe if the own ship domain is not penetrated by the target ship. Here however, a stronger condition is applied. None of the cells may belong to both ship domains at the same time. In this way, an extra buffer space is reserved for the own ship manoeuvrings. A distance is hence regarded as safe if the domains of the two ships have no common cells at any given time.

3.1. Collision avoidance by course alteration. To make the key idea more vivid, the Chang, Jan and Parberry scheme will be discussed first. To guarantee a safe passage for both ships, their scheme includes marking potential collision area as impassable for the give-way ship. This proves to be an extremely strong condition, which might even lead to not finding a valid route, when such exists. Such case is exemplified in Figure 3. Because of the limited size of the figures and the need to include all relevant information, only the largely simplified situations are shown in the paper, with ship domains limited to several adjacent cells. All methods however, have been tested extensively with real size domains and much larger maps.

In Figure 3, two ships are on their way to approach a strait. Their positions, courses and speeds are such that the ships would collide, if neither of them alters its course or speed. According to the Chang, Jan and Parberry scheme, the whole collision area – a significant part of the strait in this case – will be marked as impassable

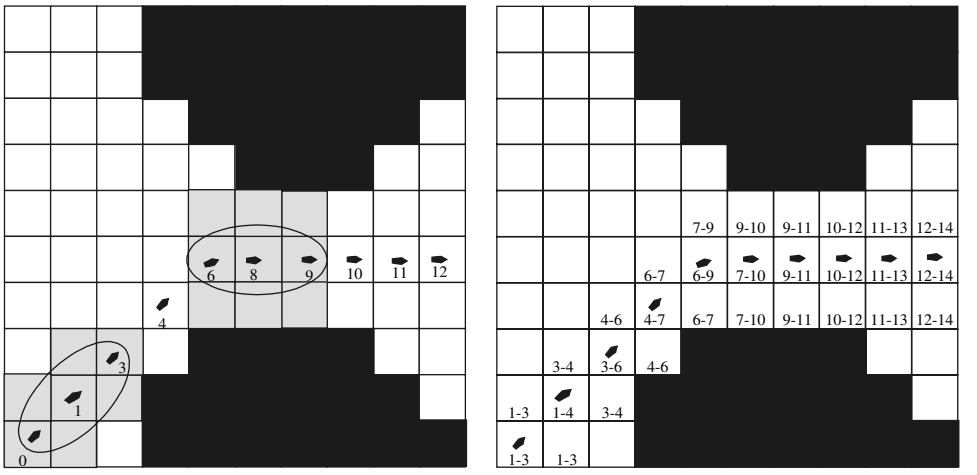

Figure 4. Left: Ship domain moving in time. Right: Cells occupied by ship domain.

for the second ship and consequently – a valid route will not be determined. Obviously this is an extreme example, but it turns out that because of marking a whole region as impassable, sub optimal trajectories are often found instead of optimal ones.

The solution to eliminate this drawback is to bring in the idea of an area being impassable at a certain time interval, instead of being impassable in general. This task involves three steps:

- determining both the impassable cells and time values corresponding to them, with linear or sub linear computational complexity,
- storing this data in a way that guarantees constant access time,
- utilizing this data in the routing algorithm without affecting its original linear computational complexity.

The realization of the first step is fairly easy. Let us call the give-way ship an own ship and the other one – the target ship from now on. For every cell occupied by the target ship, during a certain time unit, a cell's domain is determined, based on the ship's course. (In the implementation of the method, Fuji domain [2] has been used, although it is feasible to apply any other). The whole domain area including the cell occupied by the target ship should be impassable for the own ship within this time unit. If the target ship's position is undetermined (the ship traverses two or more adjacent cells instead of one) for this time unit, the ship's domain is a sum of domains for all cells traversed. The resulting marking of the cells is shown in Figure 4.

The values for each cell denote the time intervals when these cells are impassable for the own ship. Since the domain size is constant, the whole step is completed in a time proportional to the length of the fragment of the target ship route being considered.

The next step – making this data accessible in a constant time is essential for sustaining the linear computational complexity of the algorithm. This can be done on the implementation level by means of a dictionary, a map or an associative array object, depending on the programming environment. The particular entry key might be a pair of a cell index and time unit, for which the cell is impassable.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

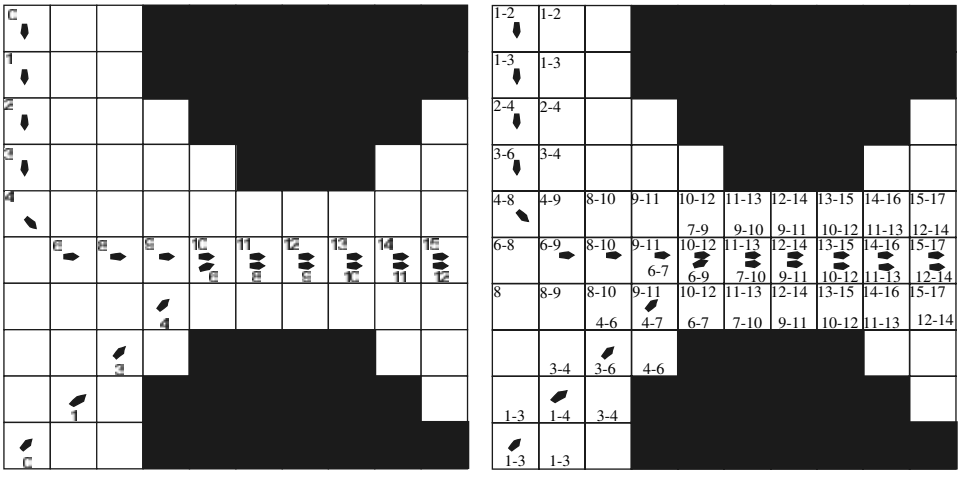

Figure 5. Left: Route for the second ship found by the proposed method. Right: Cells occupied by ship domains in time units.

The realization of the third step requires modification of the basic routing algorithm. The candidate time value of a particular gate (para 2.2) is only updated with the lesser time value, if the domain determined for this cell has no common cells with the target's domain, for the appropriate time units. According to the second step, checking whether this requirement is met, is done in constant time; hence the original linear complexity is not affected. In Figure 5 (left) the solution generated by the proposed method is shown. In Figure 5 (right) it is shown that the condition for safety is met, that is, no cell belongs to both ship domains at the same time.

In short, the routing with collision avoidance method algorithm is as follows.

1. Given the start, destination, a raster map and turn penalties of the own ship, find the shortest route using the routing algorithm.
2. Follow the route until another target ship is detected or destination point is reached.
3. Check whether there is a danger of collision (domains have common cells for any future time value), provided, that the target ship keeps its course.
4. In case of collision risk – determine the give-way ship, in case of no risk – return to step 2.
5. If own ship is to give way, find a new route with a modified routing algorithm, taking a current position for a start position. Depending on the preferences, original destination point might be kept (this would likely result in a completely different route from now on) or a point (a set of points) on the original route, past the potential collision area might be chosen for a destination (this enables own ship to follow originally determined route from the closest safe point on).
6. Follow the new route until the next target ship is detected (return to point 3.) or a destination point is reached.

As presented above, points 1 and 5 have linear complexities. Points 3 and 4 are completed within constant time and points 2 and 6 are continuous real time processes, not a part of the routing method itself.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

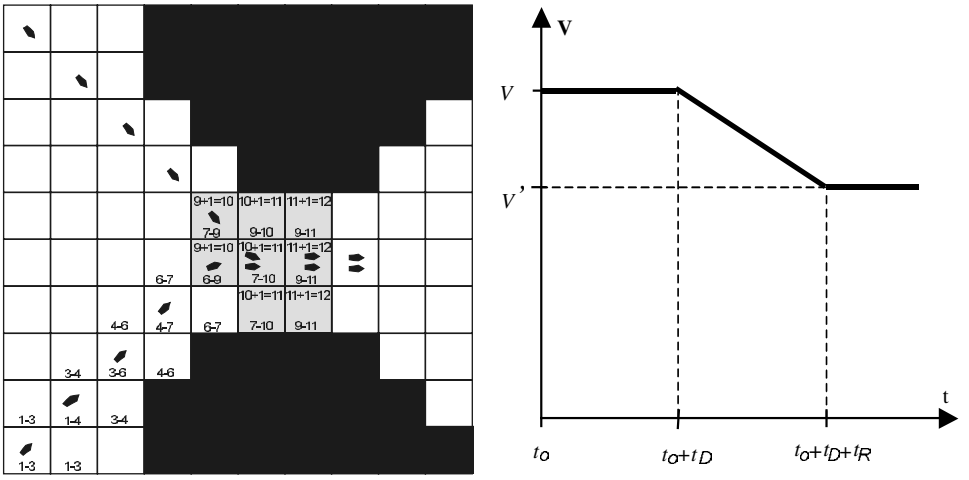

Figure 6. Left: Time units, when cells in the potential collision area might be safely reached by own ship. Right: Assumed model of the speed reduction dynamic.

3.2. Collision avoidance by speed reduction only. If course alteration alone does not guarantee a safe passage for own ship, the routing algorithm will inform the system, that there is no valid route. Also, the system may not accept a proposed route if the course alteration is too large – a 60-degree turn is often taken as a value beyond which speed alteration might be considered. A solution based on speed altering only, may enable own ship to take advantage of the shortest route – the route determined originally.

Let us denote:

- the current time unit by t_0,
- the latest time unit when the target ship (or its domain) occupies a cell by $t_{TS,i,j}$,
- the earliest time unit when the own ship (or its domain) would occupy a cell, had its speed not changed by $t_{OS,i,j}$,
- the delay time, before the speed reduction is initiated by t_D,
- the time between initialization and the end of the speed reduction operation by t_R,
- the desired time unit when the own ship domain may safely reach a cell by $t'_{OS,\ i,j}$,
- current speed by V,
- a reduced speed which results in collision avoidance for a cell $c_{i,\ j}$ by $V'_{i,j}$
- a reduced speed which results in collision avoidance for all cells by V'

The collision will be avoided, if for each of the critical cells $c_{i,j}$, an associated time value $t_{OS,i,j}$ is replaced with $t'_{OS,i,j}$ such, that:

$$t'_{OS,\,i,\,j} > t_{TS,\,i,\,j}$$

Time unit:

$$t'_{OS,\,i,\,j} = t_{TS,\,i,\,j} + 1$$

meets this requirement. In Figure 6 (left), $t'_{OS,i,j}$ values are shown in the upper parts of the cells.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

The time unit when the own ship would reach a cell without reducing its speed $c_{i,j}$ is:

$$t_{OS,i,j} = t_0 + \frac{s_{i,j}}{V},$$

where V is the current speed value (speed value in the moment t_0), and $s_{i,j}$ is the distance between the current ship position and cell $c_{i,j}$. To simplify the calculations it is assumed that the speed changes from V to V' linearly. The reduction begins after some delay time t_D and is completed after time $t_D + t_R$. The dynamics of the speed reduction is presented in Figure 6 (right). The distance that the ship covers before its speed is reduced to V' is:

$$s_R = V \cdot t_D + \frac{V + V'}{2} \cdot t_R$$

Therefore the distance that the ship covers with the speed $V'_{i,j}$, before it reaches the cell $c_{i,j}$ is:

$$s'_{i,j} = s_{i,j} - V \cdot t_D - \frac{V + V'_{i,j}}{2} \cdot t_R$$

Thus the time, when the cell may be safely reached by the ship is:

$$t'_{OS,i,j} = t_0 + t_D + t_R + \frac{s_{i,j} - V \cdot t_D - \frac{V + V'_{i,j}}{2} \cdot t_R}{V'_{i,j}}$$

The distance to the cell $c_{i,j}$ is:

$$s_{i,j} = V \cdot (t_{OS,i,j} - t_0),$$

Hence $V'_{i,j}$ is:

$$V'_{i,j} = V \cdot \frac{(t_{OS,i,j} - t_0 - t_D - \frac{t_R}{2})}{t'_{OS,i,j} - t_0 - t_D - \frac{t_R}{2}}$$

$$V'_{i,j} = V \cdot \frac{\left(t_{OS,i,j} - t_0 - t_D - \frac{t_R}{2}\right)}{t_{TS,i,j} - t_0 - t_D - \frac{t_R}{2} + 1}$$

V' that is safe for all cells $c_{i,j}$ is the minimum of all $V'_{i,j}$:

$$V' = V \cdot \min_{i,j} \left\{ \frac{t_{OS,i,j} - t_0 - t_D - \frac{t_R}{2}}{t_{TS,i,j} - t_0 - t_D - \frac{t_R}{2} + 1} \right\}$$

The determined reduced speed value V' guarantees that the own ship will not collide with the target. In reality however, often only the speed values corresponding to the basic engine room telegraph commands are considered when performing a speed reduction manoeuvre. In such case, the speed determined would have to be rounded down to the nearest engine room telegraph command speed.

The time necessary to determine the new speed value is proportional to the number of cells in the potential collision area. Thus the linear complexity of the routing algorithm is not affected.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

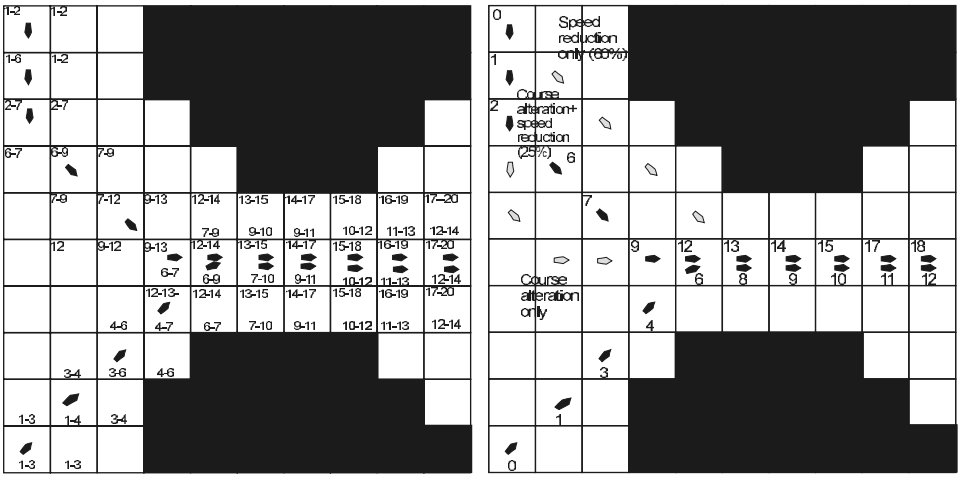

Figure 7. Left: Cells occupied by ship domains in time units for a route obtained as a result of both course alteration and speed reduction (25%). Right: A route obtained as a result of both course alteration and speed reduction (25%) compared to course alteration only and speed reduction only (60%).

3.3. Collision avoidance including course and speed alteration. If neither course nor speed alteration alone does not result in a satisfactory route, collision avoidance by both course and speed alteration must be applied. The system will determine the speed value that is closest to the current speed and still enables the algorithm to find a safe route. The method for this is a binary search algorithm with the routing algorithm used to determine, whether a certain speed reduction results in a satisfactory route. In Figure 7 (left), routes of both ships and cells occupied by their domains are shown. The reduction of the own ship's speed was 25% of the original value. In the Figure 7 (right), the same route (a result of both course alteration and speed reduction of 25% of the original value) is compared to routes from previously described cases: course alteration only and speed reduction only (60% of the original value). The alternative trajectories (course alteration only and speed reduction only) have been marked by grey icons.

The number of steps it takes for a binary search method to find the appropriate speed value within the range from the current speed to minimal speed possible is:

$$m = \log_2 \left(\frac{current\_speed - minimal\_speed}{search\_step} \right)$$

For each of these steps, it is checked whether a certain speed value meets the safety requirements, that is, whether a valid route can be found for this speed value. Therefore the resulting computational complexity for this case is $O(m*n)$, where m is given by the equation above and n is the number of cells within the area. In case, when only speed values corresponding to the engine room telegraph commands are considered during manoeuvrings, it would be enough to determine that of the speed value corresponding to the telegraph commands which is closest to the current speed and results in satisfactory trajectory found by the routing algorithm. The computational

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

complexity would then be $O(k*n)$, where:

$$k = log_2(number\_of\ \_engine\_room\_telegraph\_commands).$$

4. VTS-ORIENTED ROUTING SCHEME WITH COLLISION AVOIDANCE. Vessel Traffic Services (VTS) are popular systems for ship traffic control in restricted areas. While international regulations for preventing ship collisions at sea are always binding at open waters, VTS systems for restricted areas may override them with their own rules and priority policies. For instance, some ships might be given higher priority because of carrying an environment threatening load or for other safety or economic reasons. Therefore, there is a need for routing methods that would enable VTS system to perform central automatic route planning in accordance with its priority policy. The method presented below, is suitable for that purpose, while taking advantage of the general routing algorithm described in section 2.

1. All ships within or approaching the restricted area are sorted in descending order, according to the priority policy in force.
2. The ships routes are determined in turn, starting with the ship of the highest priority. For each ship, its route is found with the routing algorithm in such way, that the ship would not collide with any of the ships of higher priority (ship domains would not have common cells at any given time). To determine a safe route system uses the method described in section 4.1 first and if it there is a need for speed alteration – method from section 4.2.
3. Whenever a new ship approaches the area, it is given an appropriate priority and its route is determined. This route may not collide with any previously found.
4. All ships follow their routes until they reach their destination points or some emergency situation occurs.
5. In a state of emergency, the set of ships that cannot follow their original routes is determined. New routes are found for these ships in an appropriate order. New routes may not collide with those previously found. If a safe route cannot be found for given speed value, a speed is altered.

4.1. Collision avoidance by course alteration only. In this case the method works similarly to the one described in section 3.1, except that instead of assuming unchanged course of the target ship, all routes of the higher priority target ships are known. Therefore the task is to find a route that does not collide with any previously determined. An example is given in Figure 8. The cells marked as impassable by two higher priority ships for some time intervals, limit the possible routes of the lower prioritized ship in Figure. The route determined by the system for this ship is shown in Figure 8 (right). The priority policy in this example is such that the ship trajectories are determined in the following order: first *Ship1*, then *Ship2*, and then *Ship3*. The resulting computational complexity for finding a single ship's route in this case is $O(n)$.

4.2. Collision avoidance including course and speed alteration. If course alteration alone does not guarantee a safe passage for own ship, or the route determined may

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

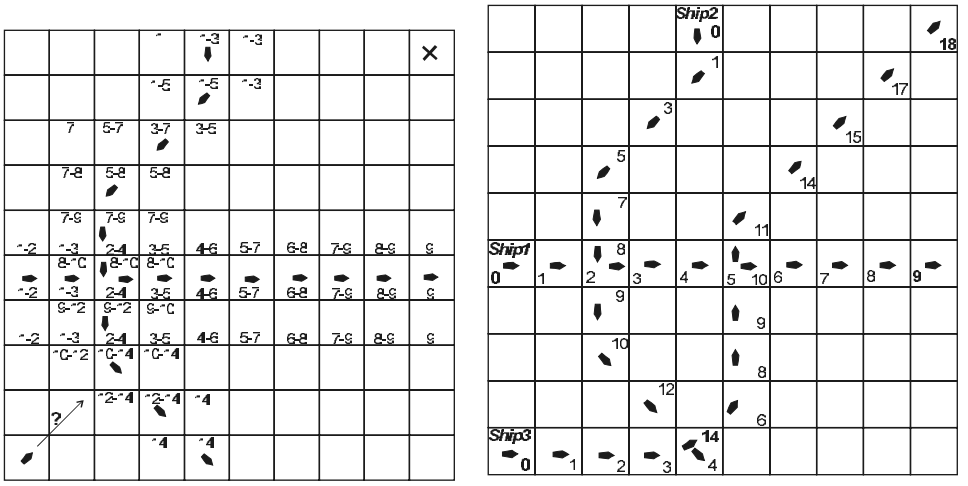

Figure 8. Left: Two routes and the cells occupied by two ship domains. Right: Solution determined by the proposed method – three routes.

not be accepted for economical reasons, speed reduction must be taken into account. However, when more than two ships in the same area have to be considered, the method of collision avoidance by speed reduction only, may not return an acceptable solution in general. Instead, routing with collision avoidance by both speed and course alteration has to be performed. Unfortunately, because there are more than two ships in the area in general, all potential reduced speed values have to be checked sequentially, instead of binary search for the two-ship problem. An algorithm performing this is presented below.

1. For speed values within range from current speed to minimal speed, with a given search step, try to find a route.
2. When a first acceptable route is found, return the route and the reduced speed value.

The number of steps the algorithm has to proceed to find the appropriate speed value within the range from the current speed to minimal speed possible, for the worst case is:

$$m = \frac{current\_speed - minimal\_speed}{search\_step}$$

The resulting computational complexity for finding a single ship's route in this case is $O(m*n)$, where m is given by the equation above and n is the number of cells within the area. However, in case when only the engine room telegraph command speeds are considered, the computational complexity would be still $O(k*n)$, where k would be the number of the engine room telegraph commands.

5. CONCLUSION. In the paper a general searching method on the raster plane was presented. Owing to its new data structure, the algorithm is capable of including costs of direction changes in the total cost of an optimal path, while

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

keeping the linear computational time and space complexities. Therefore, when costs of the direction changes are significant, the benefits of using the solution proposed here might be considerable and hence – the solution – superior to those previously known. In navigation the most obvious fields where the algorithm may be adapted are real-time routing and collision avoidance systems, since ship course alterations might be expensive or even risky in presence of obstacles. A routing method with collision avoidance, based on the algorithm has been introduced. Two separate schemes may be applied to on-ship routing systems and VTS systems respectively. The method includes speed reduction, if necessary, and is reliable to determine a safe route, if only such exists. Its additional advantages are simplicity of implementation and possibility of inclusion of any of the currently known ship domains.

ACKNOWLEDGEMENTS

The author would like to thank Prof. A. S. Lenart for his help and advice.

APPENDIX A: ALGORITHM FORMAL DESCRIPTION

The procedures INSERT and CLEAR perform the same functions as those in the Chang, Jan and Parberry method. The function GET_GATE_NUMBER (c_i,c_j) returns the number of the incoming gate of the cell c_j, through which we travel from the cell c_i. The procedure RETRACE retraces from the destination cell to the source cell and forms the output LL_{path} list. However, the rule of choosing the back way cells is different from that of Chang, Jan and Parberry procedure. Here the cell is chosen whose sum of d_i (i $\in \{1,2,3\}$) modifier and the arrival time value of previous (closer to destination) cell's incoming gate is minimal.

Algorithm 4-GEOMETRY-ROUTER-WITH-TURN-PENALTIES (*Cell-map*, *S*, *D*, d_1, d_2, d_3, LL_{path})
Input: *Cell-map*, *S*, *D*, *t1*, *t2*, *t3*
Output: LL_{path}
begin
  $bucket\_index = 0$;
  $L_{bucket\_index} = S$;
  $VIS_S = TRUE$;
  temp-list = Ø;
  path-exists = FALSE;
  all-lists-empty = FALSE;
  number-of-lists = ceiling($\sqrt{2}$ + maximum{d_1, d_2, d_3} + 1);
  while (*all-lists-empty* = FALSE) **do**
    if (*D* cell in L_{bucket_index}) **then**
      {
        path-exists = TRUE;
        break while;
      }
    for each cell c_i in L_{bucket_index} **do**

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

```
    {
      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
                distance = 1;
              Case 2: diagonal neighbours
                distance = √2;
              gate_number = GET_GATE_NUMBER(c_i,c_j);
              GAT_new = GAT_i, gate_number + distance;
              for each gate g_k of the cell c_i do
                {
                  Case 1: gate_number is the same direction as g_k
                    delay_k = 0;
                  Case 2: gate_number and g_k difference is 45 degrees
                    delay_k = d_1;
                  Case 3: gate_number and g_k difference is 90 degrees
                    delay_k = d_2;
                  Case 4: gate_number and g_k difference is 135 degrees
                    delay_k = d_3;
                  GAT_new, k = GAT_i, k + distance + delay_k;
                  if (GAT_new, k < GAT_new,) then GAT_new = GAT_new, k;
                }
              if (GAT_new < GAT_j, gate_number) then GAT_j, gate_number = GAT_new;
            }
        }
    }
  CLEAR(L_bucket_index)
  if (temp-list ≠ Ø) then
    {
      for each cell c_j in temp-list do
        {
          k = floor(minimum{GAT_j, 1, ... GAT_j, 8}) mod number-of-lists;
          INSERT(c_j, L_k);
        }
      CLEAR(temp-list);
    }
  else bucket_index = (bucket_index + 1) mod number-of-lists;
  all-lists-empty = TRUE;
  for each list L_i do
    {
      if (L_i ≠ Ø) then
```

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press

```
        {
          all-lists-empty = FALSE;
          break for
        }
    }
  end while;
  if (path-exists = TRUE) then RETRACE(Cell-map(GAT_D), LL_path)
  else path does not exist;
  end;
```

REFERENCES

[1] Chang, K. Y., Jan, G. E, Parberry, I. (2003). A Method for Searching Optimal Routes with Collision Avoidance on Raster Charts. *The Journal of Navigation*, **56**, 371–384.
[2] Fuji, Y. and Tanaka, K. (1971). Traffic capacity. *The Journal of Navigation*, **24**, 543–552.
[3] Lee, C. Y. (1961). An algorithm for path connection and its applications. *IEEE Trans. Electron. Comput*., EC-10, 346–365.

https://doi.org/10.1017/S0373463305003528 Published online by Cambridge University Press