Volume 2, Issue 1, January 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Container Yard Route Planning System

D.M.H.U. Dissanayake
*Department of Computer Science and Statistics,*
*Faculty of Science,*
*University of Peradeniya,*
*Sri Lanka.*

Saluka Kodithuwakku
*Department of Computer Science and Statistics,*
*Faculty of Science,*
*University of Peradeniya,*
*Sri Lanka.*

**Abstract - This research introduces a system that plans routes for container carriers of a container yard in an efficient and a suitable way using artificial intelligence. Once the starting point and the destination point is specified by the user, the container carriers find the best possible path to reach its destination avoiding all the obstacles that it may encounter on its way. This system also provides the user the ability of specifying storage areas at runtime since ad hoc storage areas might be created. Special collision avoidance techniques are used for container carriers to avoid collision with each other.**
**Keywords – Path-finding, route planning, search algorithms, container carrier automation, collision avoidance.**

## I. INTRODUCTION

The Container Yard Route Planning System can be considered as a part of automating container carriers in a container yard. Installing sensory systems and route planning become the two major areas of importance, when automating container carriers. In this research, what is proposed is a route planning system for container carriers in a container yard, that can be used as the route planning system for a container carrier automation system.

A container yard is an area that is designated for building, repairing, outfitting, and maintaining boats, ships, and other sea bound vessels. Apart from that, container yards also handle loading and unloading cargo ships, and storing cargo in wide land spaces inside the yard.

These cargo mostly come in the form of containers. Container carriers or trucks are used to transport the containers inside the land area. Inside a yard, transporting containers can take place in different forms containers can be moved to storage areas from the vessel or ship (unloading), or it can be moved from the storage areas to the ships (loading).

If the container trucks were automated, the process would be more efficient and time-saving. Automation thus functions as a useful mechanism in a process where time becomes a crucial factor that decides the cost of the whole process.

## II. METHODOLOGY

As the first step taken at the start of this project, the shortest path was found using Dijkstra's Algorithm, for a 4x4 grid defining starting point and destination point.

### *A. Dijkstra's algorithm*

Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency lists [5].

In the first step all the edges of the grid were given equal weight values. So there may be more than one shortest path s, out of which one of them will be selected as the shortest path.

As the second step of the process, the grid is widened and obstacles are introduced to junctions, so that the vehicle may take a different path rather than the one with obstacles.

Next, obstacles similar to storage areas are introduced in between the junctions as shown in the fig.1 as a result of which 3 kinds of junctions: 4-directional, 3-directional, 2 directional are created as shown in the fig. 2. These junctions too can be categorized into different types of junctions.

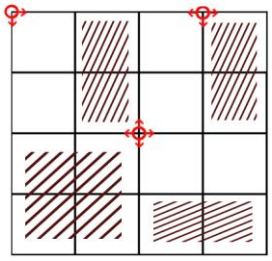

Fig. 1. Introducing obstacles in between junctions

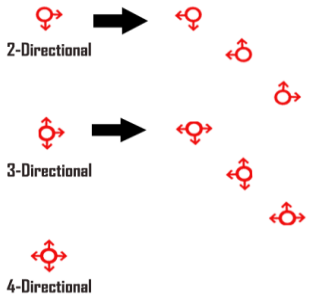


Fig. 2. Junction types and sub-categories.

An attribute is introduced to the junction class in order to hold the direction, for which a booloean array is used. As the next step, the grid was expanded and two vehicles were introduced for path finding. Destinations sta

*B. A* Algorithm*

A* algorithm is a graph/tree search algorithm that finds a path from a given initial node to a given goal node It employs a "heuristic estimate" h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. It follows the approach of best first search [6] .

The A* algorithm has three important properties [7]:

- It will always return the least expensive path if a path exists to the destination, other algorithms may find a path faster but it is not necessarily the "best" path we can take.
- A* uses a heuristic (a "guess") to search nodes considered more likely to lead to the destination first, allowing us to often find the best path without having to search the entire map and making the algorithm much faster.
- A* is based on the idea that each node has some cost associated with it. If the costs for all nodes are the same then the best path returned by A* will also be the shortest path but A* can easily allow us to add different costs to moving through each node.

A* creates two lists of nodes; a closed list containing all the nodes we have fully explored, and an open list containing all the nodes we are currently working on (the perimeter of our search). Each node will have 3 values associated with it; F, G, and H. Each node will also need to be aware of its parent so we can establish how we reached that node.

**G** the exact cost to reach this node from the starting node.

**H** the estimated (heuristic) cost to reach the destination from here.

**F = G + H** As the algorithm runs the F value of a node tells us how expensive we think it will be to reach our goal by way of that node.

***1) Heuristics:*** Selecting an appropriate heuristic is critical in determining the performance A* can achieve. Ideally we would select a value of H exactly equal to the cost of reaching our destination. If we can do so then A* will only follow the best path and never waste time exploring extra nodes. Of course we don't normally know the exact cost to reach our goal, finding it is the reason we are running a path finder in the first place. We can choose a method which will give us the exact value some of the time, such as when traveling in a straight line with no obstacles, and A* will be perfectly efficient in such cases. If we choose a value for H greater than the actual cost of reaching our goal we will allow A* to search faster but less accurately and we can no longer be certain of finding a path to the goal. Therefore we normally want to make certain that H is never accidentally greater than the real cost.

If we select a value of H less than the actual cost A* will always find the best possible path. However the lower our value of H the longer A* will take to complete its search. In the worst case of H = 0 our A* will give the same performance as Dijkstra's algorithm [7].

***Manhattan distance:*** Initial heuristic, the Manhattan distance:

$$H = |(X_{goal} - X_{start})| + |(Y_{goal} - Y_{start})| \quad (1)$$

$X_{start}$ = X co-ordinate of the starting point
$X_{goal}$ = X co-ordinate of the ending point
$Y_{goal}$ = Y co-ordinate of the ending point
$Y_{start}$ = Y co-ordinate of the starting point

At the initial phase which uses the A* algorithm, the Manhattan distance is used as the heuristic. Here when a vehicle searches a path and after finding it, the junctions of the path will be unavailable for the other vehicle. In simple words a path of one vehicle will be locked for the other. Once a vehicle finds this path, this path will be viewed as an obstacle for the other vehicle. A first come-first serve method is used to lock paths. This step is removed at the next stage.

***Euclidean distance:*** At the next phase the Euclidean distance is introduced as the heuristic, to improve the result. This is because the Euclidean distance would consider using both rectilinear and diagonal paths, whereas the Manhattan distance would only consider using the rectilinear path, making the process less efficient and more time consuming. But still it takes a considerably long time to find this path.

$$H = \sqrt{(Xgoal - Xstart)^2 + (Ygoal - Ystart)^2} \quad (2)$$

As mentioned before, the time factor plays a critical role in this system. One way of minimizing the time factor is to reduce the time taken to find the path as much as possible. Although except for some cases the travelling time is reduced by using the Euclidean distance

as the heuristic, rather than the Manhattan distance, it is noticeable that in both cases a considerable amount of time is consumed to find the path.

To minimize the time taken to find the path another heuristic is introduced to the system: the Euclidean squared distance.

***Euclidian Squared Distance:*** The Euclidean Squared distance metric uses the same equation as the Euclidean distance metric, but does not take the square root. As a result, clustering with the Euclidean Squared distance metric is faster than clustering with the regular Euclidean distance[9].

$$H= ( X_{goal} - X_{start} )^2 + (Y_{goal} - Y_{start} )^2 \quad (3)$$

When Euclidean squared distance is used it can be observed that the time taken to find the path is considerably low.

### C. *Methods for collision avoidance*

The second most important thing is to avoid various kinds of collisions that may happen during the movement of the trucks. Mainly there are two kinds of collisions.

- Collisions with Container Storage Areas, buildings and static machinery like cranes
- Collisions with moving objects (other container carriers)

The first kind of collision is avoided at the time the truck finds the path, since the storage area buildings and cranes are taken as predefined obstacles. The main concern therefore becomes the second type of collision which cannot be predefined at the time the system finds a path. As a solution to this problem several steps are taken considering secondary collisions of the type two.

#### 1) *Avoiding collision with moving objects:*

Collisions with moving objects, mainly with other container carriers, can be sub-categorized into three .

- Side-to-Side collisions
- Head-to-Head collisions
- Back-to-Head collision

***Side-to-Side collisions:*** These kinds of collisions happen when two containers meet at an angle, at the same time, at a junction. These collisions seldom happen, but there is a possibility for these kinds of collisions to happen. In order to avoid these collisions, either, one of the trucks can be stopped to let the other pass or one truck can be directed onto another path. But taking the vehicle through another path can sometimes be costly since A* algorithm has already found the best possible path. Stopping the vehicle and staying for a moment until the other vehicle passes is a more practical solution than taking another path in these kinds of situations.

A problem may arise when deciding which vehicle should be stopped and which one should be allowed to pass the junction. To avoid this confliction, a priority level is introduced for each vehicle. The priority level of a vehicle will be set to zero by default and will be incremented each time it stops so that out of the two vehicles, the vehicle with the most number of stops will be allowed to pass. Apart from that we can initialize a high priority level for a vehicle if it is an urgent delivery, so that it will not be stopped until it meets a vehicle with a higher priority level. Assume that two vehicles with the same priority level met at a junction. If something of this sort happens the distance that the vehicle has to travel to reach the destination is calculated for both the vehicles, and the vehicle with the longer distance is allowed to pass, while the priority level of the other vehicle is increased

After the paths are assigned to the trucks, the collisions can be detected by examining the arrival time of each vehicle if both the vehicles pass the same junction.

- Traverse through both paths and find if any node's,
  (Xpath1=Xpath2) AND (Ypath1=Ypath2)
- If found a point, check the arrival time of each vehicle to the point.
- If $t_{arrival}$ of vehicle 1 = $t_{arrival}$ of vehicle 2 → collision detected.
- Check the traveling direction of both vehicles. If directions are not at a $180^0$ degree angle Side-to-Side collision detected.
- Check the priority levels of the vehicles $Pr_{vehicle1}$ , $Pr_{vehicle2}$
- If,
  - $Pr_{vehicle1} > Pr_{vehicle2}$ → allow vehicle 1 to pass and increase $Pr_{vehicle2}$
  - $Pr_{vehicle2} > Pr_{vehicle1}$ → allow vehicle 2 to pass and increase $Pr_{vehicle1}$
  - $Pr_{vehicle1} = Pr_{vehicle2}$ → allow vehicle with longer distance to reach the destination to pass and increase the priority level of the other.

***Head-to-Head collisions:*** These kinds of collisions are more likely to happen since there is a great possibility of the two trucks taking the same path due to traffic. When these kinds of collisions take place, we cannot apply the algorithms used in the previous case, because we cannot avoid a head-to-head collision by merely stopping a vehicle.

As a solution for these kinds of collisions, we could either switch tracks of one of the trucks or direct one truck to go around the other, once the collision is detected as shown in the fig.3. Both these mthods can be utilized in order to avoid the collision.

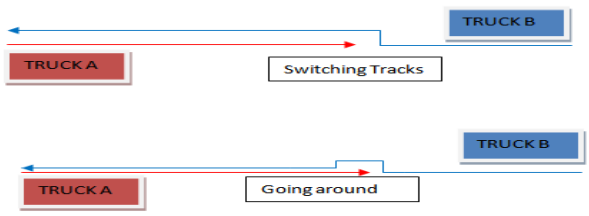


Fig. 3. Avoiding Head-to-Head collisions

Here the same criteria followed at the previous situation is applied in terms of the priority level. The vehicle with the lower priority level will amend its track

accordingly while increasing its priority, and if by any chance the priority levels are equal then the decision is made using distance measurements to the destination of each vehicle.

- The vehicle with the low priority level changes its path accordingly while increasing its priority level
- If the two vehicles are equally prioritized the vehicle with the shorter distance to its destination changes its path while increasing its priority level.

After the Paths are assigned to the trucks, the following is used to detect the Head-to-Head collisions.

- Traverse through both paths and find if any node's,
  (Xpath1=Xpath2) AND (Ypath1=Ypath2)
- If found a point, check the arrival time of each vehicle to the point.
- If $t_{arival}$ of vehicle 1 = $t_{arrival}$ of vehicle 2 → collision detected.
- Check the traveling direction of both vehicles. If directions are at a $180^0$ degree angle and directions are pointing each other Head-to-Head collision detected.
- Check the priority levels of the vehicles $Pr_{vehicle1}$ , $Pr_{vehicle2}$
- If,
  - $Pr_{vehicle1} > Pr_{vehicle2}$ → change path of vehicle 2 to and increase $Pr_{vehicle2}$
  - $Pr_{vehicle2} > Pr_{vehicle1}$ → change path of vehicle 1 and increase $Pr_{vehicle1}$
  - $Pr_{vehicle1} = Pr_{vehicle2}$ → change path of the vehicle with shorter distance to reach the destination and increase the priority level.

***Back-to-Head Collisions:*** These types of collisions can happen when two vehicles go in the same direction on the same track. If the velocity of the vehicle in the front is lower than the velocity of the vehicle at the back, these vehicles will collide.

But, since this system is designed for auto-piloted container careers that are travelling in the same velocity, these kinds of collisions do not happen unless the starting point of both vehicles is the same. In case the starting positions of the vehicles are the same, the vehicle with the longer path will be made to travel first, while the others will start moving afterwards, thus avoiding a back-to-head collision.

- Check the Starting nodes of both vehicles,
  (X1=X2) AND (Y1=Y2)
- Collision detected.
  Check the priority levels of the vehicles $Pr_{vehicle1}$, $Pr_{vehicle2}$
- If,
  - $Pr_{vehicle1} > Pr_{vehicle2}$ → allow vehicle 1 to go and increase $Pr_{vehicle2}$ and send it next
  - $Pr_{vehicle2} > Pr_{vehicle1}$ → allow vehicle 2 to go and increase $Pr_{vehicle1}$ and sent it next
  - $Pr_{vehicle1} = Pr_{vehicle2}$ → allow vehicle with longer path to start travelling, increase the priority level of the other vehicle and start it next.

2) ***Assigning obstacles on the container yard:*** There are several container storage areas, cranes, etc. in a container yard. There are some dedicated areas for the storage of containers, and if these areas are full some other spacious place in the container yard also might be used as a storage area. The dedicated areas can be predefined, but the second type of storage areas cannot be predefined. So the system provides an option to define temporary storage areas that the user desires.

## III. CONCLUSION

The objective of this system is to provide a route planning system for automating containers in a container yard. That is to move the containers from one place to another without any kind of a collision. All the above objectives are met successfully and proven with a simulation of a container yard route planning system. This system can be used as a route planning system in an automated container yard with necessary hardware installations like installing sensory systems to automate the container carriers. Hardware installations that are required to automate the container carries have not been included in the research.

We can use improved heuristics and different path finding algorithms in order to achieve more efficiency.

## REFERENCES

[1] RODNEY LAY, *Mitretek Systems* ,LYLE SAXTON, *Transportation Consultant,* Vehicle Highway Automation *Directions, Challenges, and Contributing Factors. (January 2011)*

[2] cartech blog, *reviews,http://reviews.cnet.com/8301-13746_7-2000066648.html(January 2011)*

[3] inro robotic vehicle automation, case Study , *http://www.inro.co.nz/case-study/fonterra-kauri/ (January 2011)*

[4] Machinery Automation, *www.machineryautomation.com.au/automation.html (January 2011)*

[5] *http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/ GraphAlgor/dijkstraAlgor.htm (February 2011)*

[6] Merin Puthuparampil, *Report Dijksra's algorithm. http://cs.nyu.edu/courses/summer07/G22.2340-001/Presentations/ Puthuparampil.pdf (February 2011)*

[7] Game Gardens,*http://wiki.gamegardens.com/Path_Finding_Tutorial (March 2011)*

[8] *http://www.edenwaith.com/products/pige/tutorials /a-star.php (March 2011)*

[9] *http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/ Clustering_Parameters/Euclidean_and_Euclidean_Squared_Distance_ Metrics.htm (March 2011)*