Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

A COLLISION-FREE SIMULATION FRAMEWORK FOR ASCS IN AUTOMATED CONTAINER TERMINALS

Zhuo Sun

Department of Logistics
Dalian Maritime University
1 Linghai Road
Dalian 116026, P. R. CHINA

Ziyang Qi

School of Transportation Engineering
Dalian Maritime University
1 Linghai Road
Dalian 116026, P. R. CHINA

ABSTRACT

To address the collision avoidance problem of automated stacking cranes (ASCs) in the automated container terminal, this paper provides a collision-free simulation framework for ASCs. The framework is constructed based on Spatio-temporal information and enables the effective collision avoidance of multiple ASCs even when different path planning algorithms considering only obstacle avoidance are embedded into the framework. Finally, the effectiveness of the collision-free simulation framework is demonstrated by embedding the A* algorithm and Ant Colony algorithm into the framework.

1 INTRODUCTION

Against the background of the enlargement of vessels and the continuous expansion of container terminal scale, container terminals are in urgent need of transformation towards automation and intelligence. The operational efficiency of intelligent equipment has become one of the key issues affecting the operational capability of automated container terminals. In the study of intelligent equipment, the collision avoidance problem of automated stacking cranes(ASCs) is one of the key points and difficulties. In order to facilitate the study of the multi-ASC collision avoidance problem, it is necessary to construct a collision-free simulation framework. In this way, collision avoidance can be achieved by embedding a simple obstacle avoidance path planning algorithm into the framework, which significantly increases the efficiency of simulation experiments.

The current simulation for the ASC of the automated terminal mainly simulates the process of multiple ASCs performing the loading and unloading tasks autonomously. The working scenario of ASCs is shown in the left picture of Figure 1. After one ASC receives the task of unloading ships, it first travels to the exchange area, picks up the container, and then travels to the target yard area along the routes calculated by the path planning algorithm. When the ASC arrives at the destination, it puts down the container. The process of loading ships is the opposite of the process of unloading ships. The ASCs are allowed to travel through the middle part of the buffer for efficiency. If the routes of ASCs are not collision-free during the execution of the task, it is very easy for multiple ASCs to collide when they intersect, greatly reducing the efficiency of the terminal operation.

The steps to writing the simulation program for the automated terminal are as follows. Firstly, the environment map for ASCs is described by using the grid method; as shown in the right picture of Figure 1, the buffer and the yard area are rasterized. After the environment map is described, the simulation framework is constructed according to the agent relationship, and the running logic is written for each agent in the framework. Then, the simulation program is finished. In the process of running the simulation program, the agent responsible for task transformation will transform the requirements of loading the ships

and unloading the ships into task information acceptable to the ASCs, and assign the tasks to each ASC's task list according to the task assignment algorithm. Then the ASCs will execute the tasks in the task list one by one. In the process of executing the task, the agent responsible for path planning will plan the path for the ASC according to the starting node and the destination node of the task and ensure that the ASC will follow the planned path.

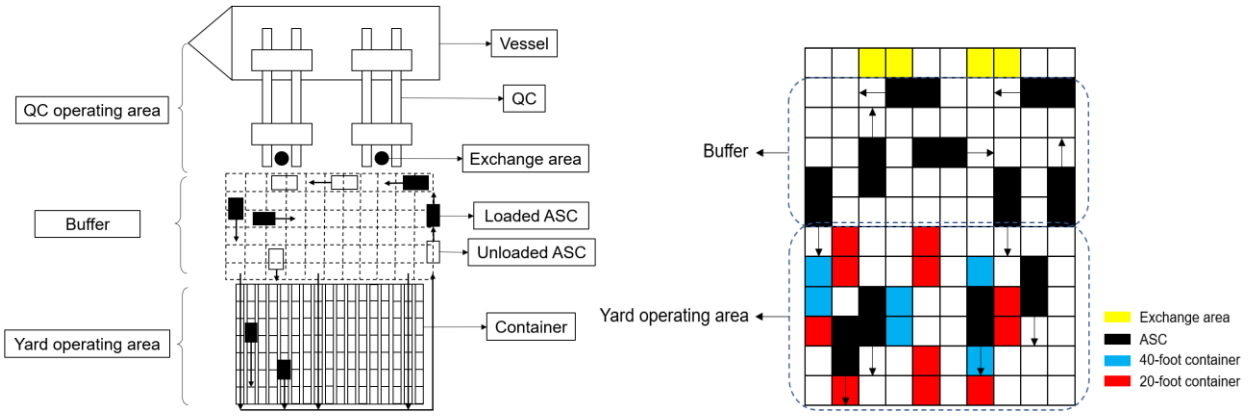


Figure 1: The working scenario of ASCs and the grid map corresponding to the scenario.

2 RELATED WORK

The collision-free path planning problem of automated guided robots can be divided into two categories, the first category is short-term collision-free path planning problems, and the second category is long-term collision-free path planning problems. Short-term path planning refers to the planning of the shortest collision-free path from the start point to the end point for all robots at a one-time point. We also call the problem multi-agent path finding (MAPF) problem. Existing researches mainly address the MAPF problem from three perspectives. The first perspective is building a model and designing an accurate algorithm to solve it. Edward et al. (2022) abstracted the MAPF problem as a network flow problem and designed a column generation algorithm combined with a branch and price strategy to solve the problem. Surynek (2012) converted the MAPF problem into a Boolean Satisfaction problem and solved them using optimization software. Such algorithms can often obtain a globally optimal solution but generally run for a long time. The second perspective is to design algorithms based on walking criteria. Luna and Bekris (2011) designed the push and pulled algorithm and its extension algorithm so that a feasible solution could be obtained quickly. Such algorithms sacrifice the quality of knowledge to ensure speed. The third perspective is search-based algorithms. Bnaya and Felner (2014) proposed a WCHA* algorithm based on collision criterion, which can dynamically place time windows and continuously adjust the planned path. Such algorithms can obtain an optimal solution or an approximate optimal solution in a relatively short period of time. The long-term path planning problem is also called the multi-agent pickup and delivery problem (MAPD),

It combines path planning and task assignment, and it also models for many other applications of multi-agent systems, including autonomous aircraft-towing vehicles (Morris et al., 2016) and other systems that use fleets of forklift robots (Pecora et al., 2018; Salvado et al. 2018) or teams of service robots (Khandelwal et al. 2017; Veloso et al. 2015), the algorithms for these models can only get feasible solutions presently.

In general, the current research on automated guided robot collision avoidance is mainly focused on short-term planning. The research on long-term planning has focused on the design of optimization algorithms. However, these algorithms cannot guarantee the quality of the solution presently, and it's also very hard to embed these algorithms into the simulation framework. So this paper designs a collision-free simulation framework to avoid collisions by embedding a 2D path planning algorithm into the framework.

3 PROBLEM DEFINITION

In discrete event simulation, the event scheduling method simulation model is frequently used. In this model, each event has a determined generation time, and then these events are added to the event list. When the simulation clock advances to the corresponding event trigger moment, and if the event trigger requirements are met, the event will be triggered. For multi-ASC conflict events, because it is not possible to know the specific time point of the conflict event, the handling of the conflict event becomes intractable in the discrete event simulation. So if a discrete event simulation framework can be designed, using the framework, it can be simple and efficient to achieve multi-ASC collision avoidance, thus can greatly increase the efficiency of doing simulation experiments.

4 COLLISION MARKING AND RESOLUTION

4.1 Spatio-temporal Barrier Table

For collision avoidance of multiple ASCs, it is very important to allow the ASC to obtain as much information as possible about the environment or the location of other vehicles. The most widely used algorithms in discrete event simulation are basically 2D path planning algorithms. They can achieve complete avoidance of static obstacles and find the shortest path from the starting node to the destination node in a short period of time. However, this type of algorithm still cannot meet the requirements of collision avoidance because each ASC can be seen as a moving obstacle while executing its task, and the path planning algorithms are often designed for static obstacles. So the key issue of our study is to design a simulation framework that satirizes moving obstacles and marks all obstacles on a public table. In this way, when the agent plans paths for ASCs, it can refer to the public table to avoid the marked obstacles and thus achieve collision avoidance. The process of stabilising moving obstacles is shown in Figure 2.

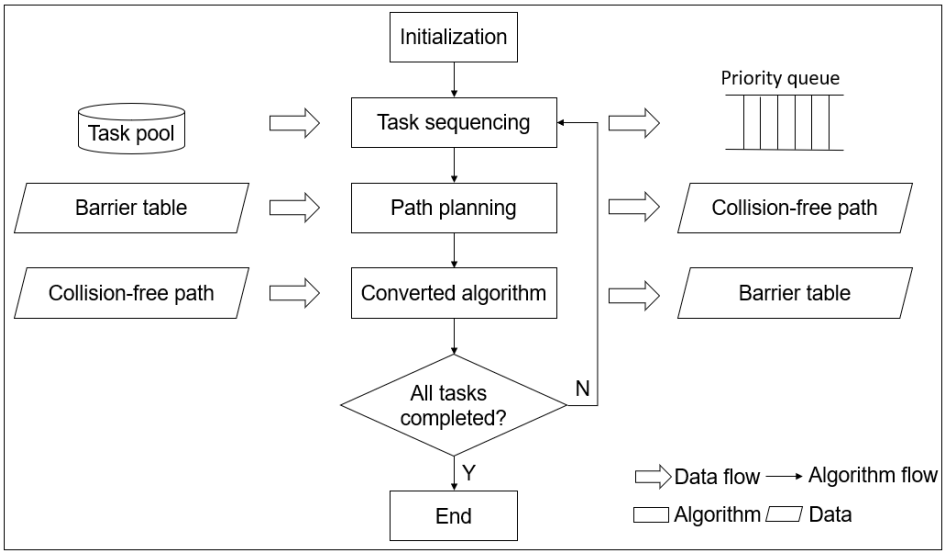


Figure 2: Flowchart for satirising moving obstacles.

In our study, all ASCs executing the task are set to maintain uniform linear motion and move at the same speed. The moment at which the ASCs send path planning requests is set as integer time points. In this study, a task manager agent is designed in the simulation framework. Each ASC needs to get a collision-free path that will send the information of the starting node and the destination node to this task manager at integer time points, and the task manager will be started every simulation second. The main purpose of designing the task manager is to sort the tasks according to the priority of the tasks and to increase the synergy of path planning. We do this because if the ASCs plan their own paths, they will only consider the shortest path for them to complete their tasks. It's hard for them to take the initiative to avoid the ASCs

whose tasks are more difficult to complete when there is a risk of collision, which may cause deadlock and other system failures. When the task manager is started, the tasks of the task pool will be sorted first, and the tasks with higher priority will be planned firstly, so as to ensure that ASCs with lower priority will give way to ASCs with higher priority. This ensures that important tasks are completed first, and the system failure is not caused.

The overall collision avoidance process is initializing the simulation parameters first. Then each ASC will continuously receive tasks of loading and unloading the vessels. ASCs will execute the assigned tasks one by one according to the time order. For the currently executed task, the ASC will send the task to the task manager. That is to say, the information of the starting node and destination node of the task is uploaded to the task pool of the task manager. The task manager then collects all the tasks that need to be solved in the current simulation second and starts to plan the path for the tasks. It sorts all the tasks in the task pool according to the priority of the task. After sorting, it plans the paths for the tasks one by one in the order of priority of the task from highest to lowest. When planning the paths, the task manager needs to refer to the Spatio-temporal barrier table to view the occupancy time period of the nodes and find the free time windows which are available for scheduling other vehicle occupancies so that it can plan the shortest path for obstacle avoidance. After the shortest path is planned, the planned path is added to the Spatio-temporal barrier table as new obstacles using a converted algorithm that takes into account collision avoidance.

The structure of the Spatio-temporal barrier table and the path corresponding to the data in the table are shown in Figure 3, respectively. Because the simulation environment of this study is described by using the grid method, in order to clearly represent the position of the obstacles at each time point, the Spatio-temporal barrier table is designed. The first column of the table indicates the grid ID, and the first row of the table indicates the simulation time node. If a grid at a certain time point is not occupied, the corresponding position in the table is marked 0. If a grid at a certain time point is occupied, the corresponding position in the table is marked 1.

Simulation time / Grid id	…	3	4	5	6	…
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	1	0	0	0	0
4	0	0	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	1	0	0
7	0	0	0	0	0	0
8	0	0	0	0	1	0

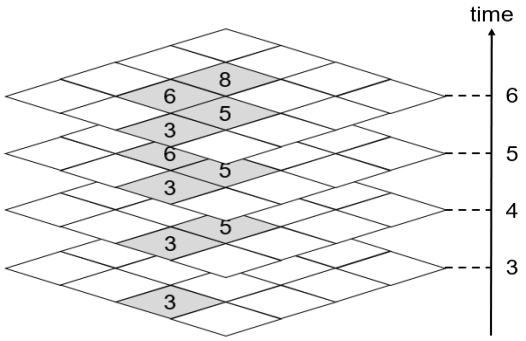


Figure 3: The data in the Spatio-temporal barrier table and the real path corresponding to the data.

ASCs actually need to avoid two types of obstacles, one is the container placed in the yard, the container becomes a static obstacle when the direction or height of the placement makes ASCs unable to cross, and the other is the moving ASCs. We designed two tables. The first Spatio-temporal barrier table is the closelist table, which is used to record the occupancy situation of all yard storage locations at each time point. The second Spatio-temporal barrier table is the reservation table, which is used to record the location of ASCs at each time point. In order to facilitate the use of Spatio-temporal barrier tables in the simulation framework, we abstract the Spatio-temporal barrier table into the form of a hash table if, during the period of planning a path, it is necessary to know whether the grid of row i and column j is occupied at time t, then the task manager can quickly check the value of closelist[i][j][t] or reservation[i][j][t]. If the value is 1, the grid is occupied at that time point. If the value is 0, it is passable.

For each container placed on the storage location, from the time it is placed on the storage location by an ASC, it will occupy the grid for a period of time. But a problem arises, when the container is placed in the storage location, it's hard to know when it will be moved. That is, we do not know how many 1 needs

to be marked in the table. Therefore, in this study, the values in the closelist table corresponding to storage location from the arrival time of the container till the end of the simulation time are set as 1. When this container is removed, change the values in the table corresponding to this storage location from the current time point to the end of the simulation time to 0. The above method solves the problem of marking the container as a temporary obstacle in the Spatio-temporal barrier table. Supposing a container from time 3 to time five is placed in the grid 2, and it will be moved at time 6, the corresponding values in the Spatio-temporal barrier table closelist will present as the form in Figure 4 so that the problem of marking the grid occupied by the container in the Spatio-temporal barrier map is solved.

Before moving

Simulation time / Grid id	…	3	4	5	6	…
1	0	0	0	0	0	0
2	0	1	1	1	1	1

After moving

Simulation time / Grid id	…	3	4	5	6	…
1	0	0	0	0	0	0
2	0	1	1	1	0	0

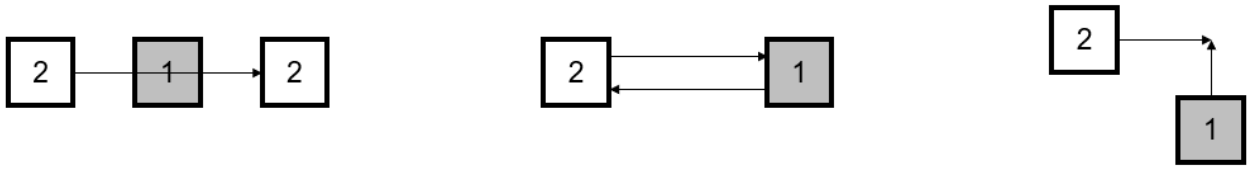
Figure 4: Diagram of how containers are marked in the closelist table as obstacles.

When planning a path for one ASC, it is necessary to determine whether a node is occupied at a certain time point. Firstly the task manager checks whether there is a risk that the current moving direction will conflict with the containers. If not, there is no need to check the closelist table, directly check the reservation table. But if a collision may happen, it's necessary to check the closelist table to see if the container is placed in that storage location. If not, the task manager continues to check the reservation table to see if there is a risk of conflicting with other ASCs. If not, the node can be chosen.

4.2 Conflict Resolution

4.2.1 Resolving Cross Conflict

There are three types of conflicts that ASCs are prone to while driving, as shown in Figure 5. The first is overtaking conflict, the second is head-on conflict, and the third is cross conflict.



(a) Overtaking conflict (b) Head-on conflict (c) Cross conflict

Figure 5: Classification of conflicts.

Since all ASCs are set to travel at the same speed and maintain uniform linear motion in this simulation, there is no overtaking conflict. For cross conflict, when the task manager plans the path for a task, it will check the Spatio-temporal barrier table to observe the obstacle situation and avoid choosing a grid that has already been marked as an obstacle at some point in the future. After planning the path, the task manager adds the planned path to the Spatio-temporal barrier table. When planning the path for subsequent tasks, the task manager just needs to avoid choosing the grid that has been marked one at the corresponding time point, thus making it hard for multiple ASCs to occupy the same grid at the same time and the cross conflicts can be solved. Assuming that the task being executed by ASC1 has a higher priority than that of ASC2 and both vehicles send a path planning request at the moment 0. ASC1 wants to travel from the grid

corresponding to (3,1) to the grid corresponding to (3,5), while ASC2 wants to travel from the grid corresponding to (1,3) to the grid corresponding to (5,3). After the two vehicles pass their tasks to the task manager, the task manager judges that ASC1's task priority is higher than ASC2's, so it plans the path for ASC1 first. Since there are no obstacles on the road by checking the Spatio-temporal barrier table, so the planned path for ASC1 is a straight line: (3,1), (3,2), (3,3), (3,4), (3,5). This path is then put into the Spatio-temporal barrier table according to the corresponding time points, which is equivalent to turning ASC1, a moving obstacle, into static obstacles recorded in the table at several integer simulation time points. When planning the path for subsequent ASCs, as long as these obstacles are avoided at the corresponding simulation time points, it is equivalent to avoiding ASC1 and achieving the collision-free effect. After planning the path for ASC1, the task manager continues to plan the path for ASC2. If the Spatio-temporal barrier table is not checked, the planned path for ASC2 should also be straight. But if using the Spatio-temporal barrier table, we can see that at the moment 2, the grid (3,3) is already occupied by ASC1. A better choice for ASC2 is to stay at the grid (2,3) for one simulation second at time 2 and then continue at time 3, thus avoiding a cross conflict with ASC1 and making ASC2 reach the destination in a relatively short period of time. The above process is shown in Figure 6, so checking the Spatio-temporal barrier table when planning a path can effectively solve the cross-conflict problem.

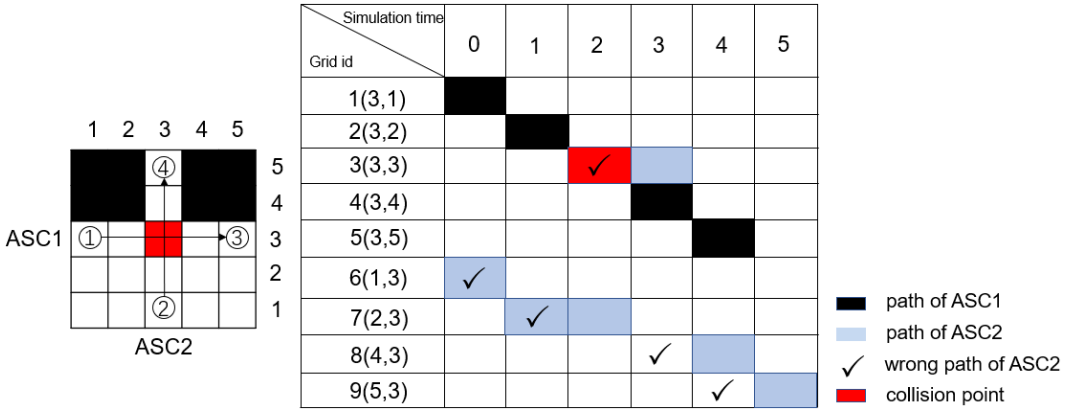


Figure 6: The process of resolving cross conflicts.

4.2.2 Resolving Head-on Conflict

Firstly, if we ignore the size difference between ASCs and containers, each of them will occupy a grid at integer time points, thus making it easier for the head-on conflict to happen. Supposing that ASC1 is now on the grid 1 and ASC2 is on the grid 2, ASC1 has a higher priority of task. The partial path planned for ASC1 is to travel to grid 1 at time t, and to travel to grid 2 at time t+1. Although the task manager can see ASC1's path when it plans a path for ASC2, it may still plan a path requiring ASC2 to travel to grid 2 at time t and travel to grid 1 at time t+1. As shown in Figure 7, the dark cells indicate ASC1's partial path and the light cells indicate ASC2's partial path. At time t and time t+1, ASC2 does not occupy grids already occupied by ASC1, but the head-on conflict still happens, thus forming a crossover structure.

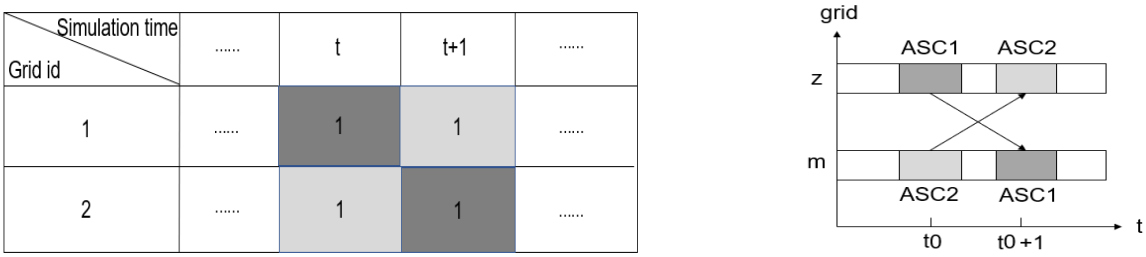


Figure 7: A two-dimensional representation of a directed conflict.

The above example shows that it's unwise to ignore the dimensions of the entities practically. To make it impossible to form a crossover structure and to make the simulation environment closer to reality. We resize each entity in the simulation experiment according to its actual size, as shown in Figure 8. A 20-feet container occupies a grid at integer time points. A 40-feet container occupies two consecutive grids at integer time points. An ASC occupies two consecutive grids either because it ensures that the ASC can grab two 20-feet containers or one 40-feet container at one time practically. In the process of updating the Spatio-temporal barrier table, we devise a method that puts a path node into the Spatio-temporal barrier table in a way that occupies two consecutive integer time points. This means that when an ASC with a higher task priority is to travel to grid m at time t and to travel to grid z at time t+1, the coordinate of grid m is (x,y) and the coordinate of grid z is (x1,y1), the nodes occupied by the ASC are (x,y,t), (x,y,t+1), (x1,y1,t+1), and (x1,y1,t+2). We do like this because when some components of the ASC have moved to (x1,y1) at time t+1, there is still some component of the ASC just arriving at (x,y) for that one ASC occupies two consecutive grids. When the ASC occupies grid m at time t, the method mentioned above is guaranteed that grid m will not be occupied at time t+1 by any of the other ASCs, thus not only being consistent with the devised size of the ASC in the grid map but also eliminating all possibility of head-on conflicts, we call this method a converted algorithm.

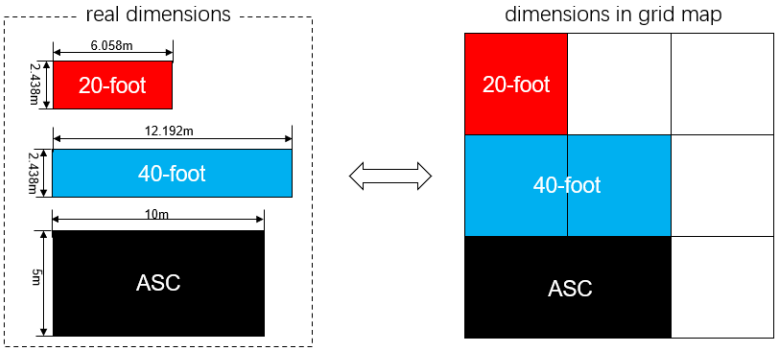


Figure 8: The correspondence of dimensions.

An example of this converted algorithm is shown below. As shown in Figure 9, ASC1 needs to travel from the grid (3,1) to the grid (3,4), and ASC2 needs to travel from the grid (3,4) to the grid (3,1). ASC1's priority is higher than ASC2's. If the converted algorithm is not used, then when ASC1 has finished planning the path, ASC2 will not be able to identify the head-on conflict according to the Spatio-temporal barrier table, thus colliding with ASC1.

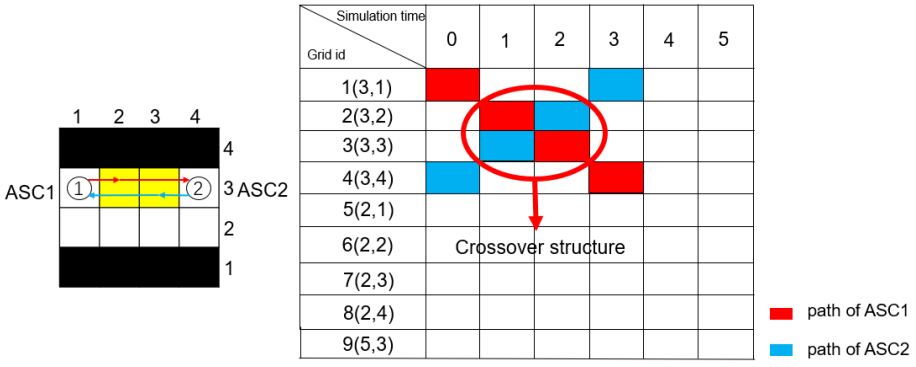


Figure 9: The crossover structure in the Spatio-temporal barrier table.

When the path of ASC1 is put into the Spatio-temporal barrier table using the converted algorithm, each path node of the path will occupy the same grid at two consecutive integer time points so that when

the table value of the grid (3,2) at time 1 is set as 1, its value at time 2 will also be set as 1, which means that ASC2 cannot move to the grid (3,2) at time 2, thus avoiding the formation of a crossover structure and solving the potential head-on conflict. The paths of the two ASCs planned by using the converted algorithm are shown in Figure 10. Although it takes longer for ASC2 to reach its destination, the head-on conflict is avoided and thus preventing great damage to the overall system.

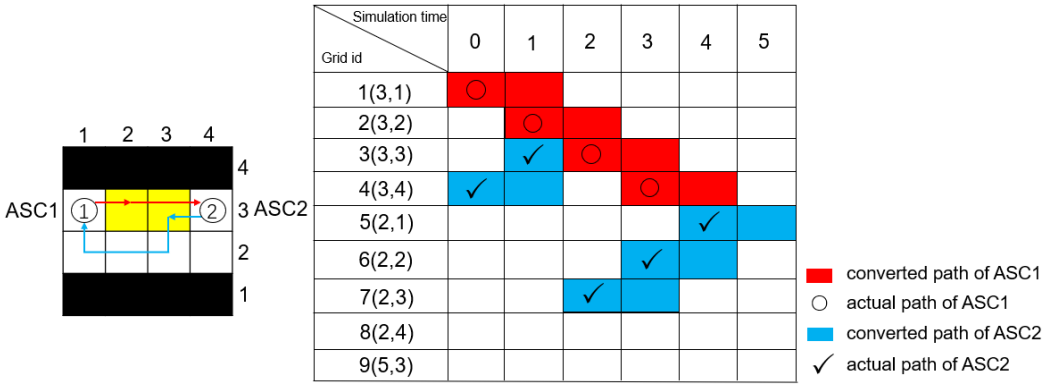


Figure 10: The converted path in the Spatio-temporal barrier table.

4.2.3 Resolving the Conflict With Stationary ASCs

When an ASC reaches the destination, it needs to stop and then pick up or put down the container. During this period of time, the ASC is equivalent to a stationary obstacle, so it is necessary to put the information of the ASC into the Spatio-temporal barrier table. Otherwise, it is likely to cause other ASCs to collide with this ASC.

The ASC in a static state will occupy the same grid for a period of time. Assuming that ASC arrives at the destination grid 100 at time t. This causes another problem that we don't know how long it takes the ASC to pick up or put down the container and how long the grid 100 will be occupied by the ASC. To solve this problem, the same processing method as in section 4.1 is used. The values in the reservation table corresponding to grid 100 from the arrival time of the ASC till the end of the simulation time are all marked as 1. The time point when the ASC leaves is assumed to be t+2, and the values in the reservation table corresponding to the time point t+2 to the end of the simulation time is marked as 0. Because the grid 100 is no longer occupied after the ASC leaves, it is necessary to modify the value in the reservation table to let other ASCs know that they can pass the grid 100 after time t+2 normally. The above operations are shown in Figure 11.

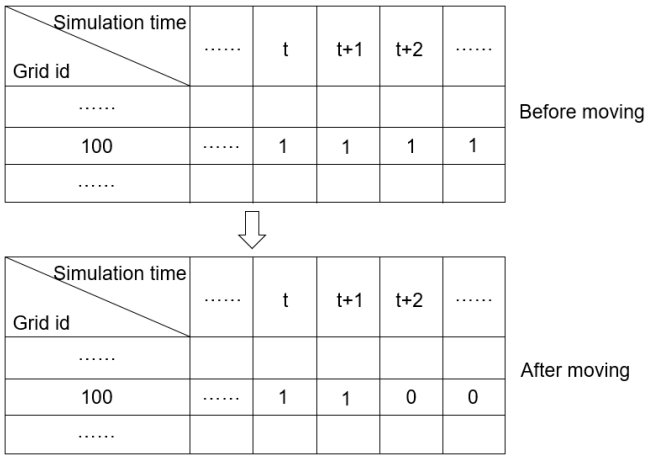


Figure 11: Diagram of how stationary ASCs are marked in the reservation table.

5 COLLISION-FREE SIMULATION FRAMEWORK

The collision-free simulation framework is shown in Figure 12. First of all, the tasks of loading the ships and unloading the ships are generated according to a certain rule, which can be determined by the actual situation. Because different automated terminals are in different situations and the number of ships arriving at the same terminal can change in the same long period of time. So using random seeds to simulate the frequency of the generation of the tasks is a feasible method.

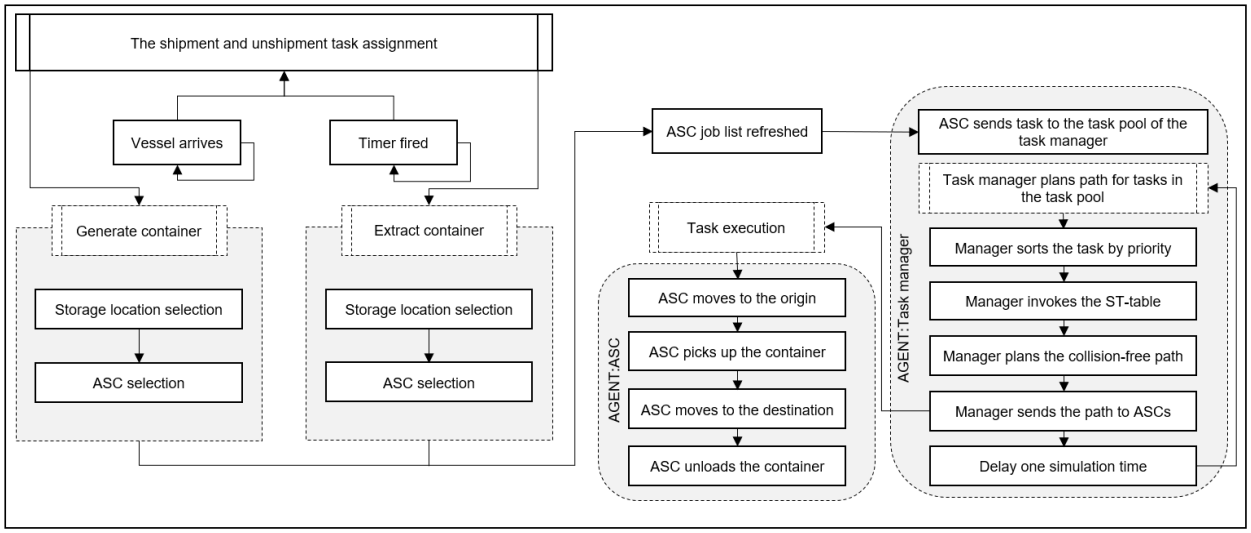


Figure 12: The collision-free simulation framework.

When the threshold for a task of loading the ships or unloading the ships is met, a corresponding task will be generated. When a task of unloading the ship is generated, there are two steps to be performed, the first step is to determine where in the yard the unloaded container should be stored, and the second step is to select a suitable ASC to perform that task. Similarly, for the task of loading the ship, there are still two steps to be performed. The first step is to select a suitable container to meet the demand and then to select a suitable ASC to perform the task. Each of these two steps can be transformed into an operations optimization problem, then a model can be built, and a suitable algorithm can be designed to get an optimal solution. Because this part is not the focus of our study, we do not discuss this part in detail. The method of violent search is used to determine the storage location, the suitable container, and ASC. For the selection of storage location, firstly, the storage space in the yard area is traversed, and the first storage location that does not have a container and is suitable for the size of the container that needs to be stored is selected. For the selection of the suitable ASC, first number the ASCs, then assign the task to the first ASC when the first task is generated, then assign the task to the second ASC when the second task is generated, until all ASCs are assigned, then assign the current task to the first ASC again, and keep rolling the assignment.

When a task is assigned to an ASC, it is first added to the task list of that ASC. Then the ASC agent determines whether the ASC is currently executing one task. If not, the new task would be executed immediately. If the ASC is busy, then the new task will be executed when all the tasks that were added to the task list ahead of the new task have been executed. For the current task to be executed, the ASC will send the task to the task manager. Each task essentially consists of two path nodes. The first path node indicates the starting node of the task, and the other path node indicates the destination node of the task. The task sent by the ASC will first be stored in the task pool of the task manager, and when the task manager has collected all the tasks that need to be executed at the current simulation second, it will sort the tasks in the task pool in order of task priority from highest to lowest, and then plan the path for the tasks according to the ranked order. In the process of path planning, the task manager will first check the Spatio-temporal barrier table and plan the path according to the data in the Spatio-temporal barrier table because the obstacles represented in the Spatio-temporal barrier table are generated by the converted algorithm, so the

path planned by referring to the table is the shortest path to avoid collisions. After planning the path for all the tasks in the task pool, the task manager will send the route to the corresponding ASC. When the ASC receives the planned route, it will first move from its current position to the starting node of the task, pick up the container, then transport the container to the destination node of the task, and put down the container.

6 CASE STUDY

In order to verify the effectiveness of the collision-free simulation framework, a collision count indicator is designed to calculate the number of collisions that occur during the whole experiment, and two-time consumption indicators are designed to verify the computational efficiency of the simulation framework. We use the idea of controlled variables and design two groups of comparative experiments. In the first group, we observe whether both the embedded A* algorithm and the ACO algorithm can achieve complete collision avoidance by changing the number of ASCs and the time interval at which the tasks arrive. In the second group, we calculate the average time for the framework to plan for each task and the average calculation time of all tasks in one simulation second to observe the computational efficiency of the framework. The simulation is realized in the spatial planning tool MicroCity (https://microcity.github.io).

6.1 Design of Indicators

In the simulation experiment of the automated terminal, the calculation of the number of collisions is set at each update time of the positions of all ASCs. If the centers of two ASCs overlap each other, then the two ASCs must be in a collision. For the average time to plan for each task, if it is short and stable when we change the number of ASCs for all embedded algorithms, we can say that the framework is practical. For the average calculation time of all tasks in one simulation second, as we mentioned in section 5, the task manager will be awakened per simulation second and plans the path for all the tasks that need to be executed at the current simulation second, if the calculation time is short, we can conclude that the framework is efficient.

6.2 Comparison of Simulation Results

The structure of the task table is shown in Table 1. The tasks of the task table are executed in turn. This experiment set a total of two types of containers, the first type is a 20-foot container, and the second is a 40-foot container. A 40-foot container can roughly equate to two 20-foot containers. In terms of time settings, the loading and unloading time of the former is shorter than that of the latter.

The results of the first group experiments are shown in Table 2 and Table 3. We can conclude that the simulation framework can be used to achieve complete collision avoidance of ASCs performing tasks when the number of ASCs or the average time interval of task generation changes. The results of the second group experiments are shown in Table 4. When we change the number of ASCs and embedded algorithms, the average time to plan for each task is stable and short. Also, the average calculation time of all tasks in one simulation second is short. So we can conclude that the framework is practical and efficient for small terminals with dozens of ASCs.

Table 1: The structure of the task table.

Id	Dimension	Number	Task
1	20	3	load
2	40	4	unload
3	40	5	load
4	20	3	unload
5	40	2	unload
6	20	4	load
7	20	2	load
8	40	3	unload

Table 2: The results of the first group expirements.

Algorithm	Number of ascs	Time interval	Number of collisions
A*	4	15	0
	5	15	0
	6	15	0
	7	15	0
	8	15	0
ACO	4	15	0
	5	15	0
	6	15	0
	7	15	0
	8	15	0

Table 3: The results of the second group expirements.

Algorithm	Time interval	Number of ascs	Number of collisions
A*	5	6	0
	10	6	0
	15	6	0
	20	6	0
	25	6	0
ACO	5	6	0
	10	6	0
	15	6	0
	20	6	0
	25	6	0

Table 4: The calculation speed of the simulation framework.

Algorithm	Number of ascs	The average time to plan for each task	The average calculation time of all tasks in one simulation second
A*	5	0.0298	0.0388
	10	0.0316	0.0494
	20	0.0286	0.0455
	30	0.0313	0.0503
	40	0.0300	0.0482
ACO	5	0.4196	0.5473
	10	0.3819	0.5977
	20	0.3836	0.6115
	30	0.3590	0.5775
	40	0.4055	0.6523

7 CONCLUSION AND FUTURE WORK

In this paper, a discrete event simulation framework is designed, which makes reasonable use of Spatio-temporal information, viewing the ASC, a moving obstacle, as a static obstacle at each integer simulation time point and putting this obstacle information into a publicly available Spatio-temporal barrier table for reference by other vehicles, thus successfully achieving multi-ASCs collision avoidance.

In future research, the speed variation of the ASC needs to be taken into account in the simulation model, but considering the speed variation of the ASC creates a new problem that there may not be a way to guarantee that all ASCs arrive at the grid node at integer simulation time points, thus making it impossible to put the paths into the Spatio-temporal barrier table designed in this study.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (61304179, 71431001, 71831002); the Humanity and Social Science Youth Foundation of the Ministry of Education (19YJC630151); the International Association of Maritime Universities (20200205_AMC); the Natural Science Foundation of Liaoning Province (2020-HYLH-32); the Dalian Science and Technology Innovation Fund (2020JJ26GX023); the Social Science Planning Fund of Liaoning Province (L19BGL011); the National Social Science Fund(21BGJ073).

REFERENCES

Lam, E., P. L. Bodic, P., D. Harabor, and P. J. Stuckey. 2022. "Branch-and-Cut-and-Price for Multi-Agent Path Finding". *Computers and Operations Research* 144:105809.

Surynek, P.. 2012. "Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving". In *PRICAI 2012: Trends in Artificial Intelligence*, edited by P. Anthony, M, Ishizuka, and D. Lukose, 564-576. Berlin, German: Springer.

Luna, R., and K. E. Bekris. 2011. "Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees". In *proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, edited by T. Walsh, NICTA, and University of NSW, 294-300. Menlo Park, California: AAAI Press/International Joint Conferences on Artifical Intelligence.

Bnaya, Z., and A. Felner. 2014. "Conflict-Oriented Windowed Hierarchical Cooperative A*". In *IEEE International Conference on Robotics and Automation ICRA*, edited by J. D. Tew, M. Manivannan, D. A. Sadowski, and A. F. Seila, 3743-3748. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Morris, R., C. S. Pasareanu, K. Luckow, W.Malik, H. Ma, T. K. S. Kumar, and S. Koenig. 2016. "Planning, Scheduling and Monitoring for Airport Surface Operations". In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, February 12th-17th, Phoenix, USA, 608-614.

Pecora, F., H. Andreasson, M. Mansouri, and V. Petkov. 2018. "A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control". In *Proceedings of Twenty-Eighth International Conference On Automated Planning And Scheduling*, edited by M. D. Weerdt, S. Koenig, G. Roger, and M. Spaan, 485-493. Palo Alto, California: AAAI Press.

Salvado, J., R. Krug, M. Mansouri, and F. Pecora. 2018. "Motion Planning and Goal Assignment for Robot Fleets Using Trajectory Optimization". In *IEEE International Conference on Intelligent Robots and Systems*, edited by J. Kosecka, 7939-7946. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Khandelwal, P., S. Zhang, J. Sinapov, M. Leonetti, J. Thomason, F. Yang, I. Gori, M. Svetlik, P. Khante, V. Lifschitz, J. K. Aggarwal, R. Mooney, and P. Stone. 2017. "BWIBots: A Platform for Bridging the Gap Between AI and Human–Robot Interaction Research". *International Journal of Robotics Research* 36(5-7):635-659.

Veloso, M., J. Biswas, B. Coltin, and S. Rosenthal. 2015. "CoBots: Robust Symbiotic Autonomous Mobile Service Robots". In *proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, edited by Q. Yang, M. Wooldridge, 4423-4429. Freiburg, German: Ijcai-Int Joint Conf Artif Intell.

AUTHOR BIOGRAPHIES

ZHUO SUN is a Professor in the School of Transportation Engineering at the Dalian Maritime University. He earned his Ph.D in Japan's Nagoya University. His research focuses on shipping network design, port planning, and port operation. Early in his undergraduate studies, it dawned on him to develop a software tool that would help analyze transport systems. The following years served as an incubation period for his idea, and after almost ten years of contemplation and devotion to hard work, his idea was finally realized as 'MicroCity'. His e-mail address is mixwind@gmail.com.

ZIYANG QI is a student in the School of Transportation Engineering at the Dalian Maritime University. He is now studying for a Master's degree and major in Logistics Engineering and Management. His e-mail address is 919948694@qq.com.