Int j simul model 23 (2024) 2, 323-334
Original scientific paper

CONTINUOUS PATH PLANNING FOR MULTI-ROBOT IN INTELLIGENT WAREHOUSE

Shen, G. C.[*]; Liu, J.[*]; Ding, Y. L.[*]; Zhang, C.[**] & Duo, J. Y.[*,#]
[*] School of Information, Beijing Wuzi University, Beijing 101149, China
[**] School of Computer Science and Information Engineering, Anyang Institute of Technology, Anyang 455000, China
E-Mail: shenguicheng@bwu.edu.cn, liujiebwu@163.com, d15255251372@163.com, zc_code@163.com, duojingyun@bwu.edu.cn ([#] Corresponding author)

Abstract
In smart warehouses, logistics robots need to continuously execute tasks, and therefore traditional one-time multi-robot path planning cannot meet practical needs. To address this challenge, we proposed Pre-judgment Conflicts Search (SPC) algorithm. The SPC algorithm is divided into two layers: the higher layer allocates tasks to robots based on priority rules, and the lower layer plans paths for these robots with an improved A* algorithm, which includes vertex and edge collision checks during path searching and effectively prevents collisions among robots. We simulated SPC and Conflict-Based Search (CBS) on random maps and warehouse maps. The results show that compared to CBS, SPC provides nearly optimal paths with a significantly reduced computation time. Additionally, as the number of robots increases, SPC exhibits better scalability, offering a practical solution for continuous multi-robot path planning in smart warehouses, with the potential to enhance logistics efficiency.
(Received in December 2023, accepted in March 2024. This paper was with the authors 1 month for 2 revisions.)

Key Words: Intelligent Warehouse, Continuous Path Planning, Pre-Judgment Conflict Search, Priority

1. INTRODUCTION

In the era of Industry 4.0, the rapid development of technologies such as the Internet of Things, 5G, and artificial intelligence has facilitated our production and daily life. Various functional robots are widely used, including handling robots (such as Automated Guided Vehicle, AGV), detection robots [1], and patrol robots [2]. This not only reduces labour costs but also enhances production efficiency and detection accuracy. At the same time, with the continuous increase in online shopping users, urban logistics [3] and intelligent warehouse [4] are rapidly developing. As a crucial tool in current intelligent warehousing, the scheduling problem of AGVs [5] has become the focus of current research. The multi-robot path planning algorithm, as the core technology for AGVs scheduling, holds important research significance and value.

Many algorithms have been proposed for multi-robot path planning, including heuristic algorithms [6-10], meta-heuristic algorithms [11-15], artificial potential field methods [16, 17], and deep reinforcement learning methods [18, 19]. Zeng et al. [8] planned paths for mobile robots with the A* algorithm in game scenarios. Raheem and Abdulkareem [9] used A* algorithm to find the shortest path in a constructed roadmap. Kennedy and Eberhart [12] proposed the original Particle Swarm Optimization algorithm (PSO). Tang et al. [13] proposed a co-evolutionary-based PSO algorithm, which can fix the global and local searching capabilities to solve PSO recession problem. Mirjalili and Lewis proposed Whale Optimization Algorithm (WOA) [14] in 2016. Chhillar and Choudhary [15] improved WOA to guarantee the optimal collision-free path. Khatib [16] proposed the Artificial Potential Field method (APF) which plans a local path in 1985. However, the traditional APF method suffers from target unreachability problems and local minima problems in certain environments [17]. The deep reinforcement learning method [19] is used for path planning, but the planning success rate is only around 90 %.

However, traditional path planning algorithms like CBS [20] have long computation times and may not find solutions in environments with a high number of robots. Intelligent computing methods, such as Ant Colony Optimization (ACO) [21], Genetic Algorithm (GA) [22], PSO, etc., suffer from premature convergence, low efficiency, and long computation times. Additionally, these algorithms primarily address one-time path planning problems, while in real-life scenarios, robots often need to continuously handle a series of tasks.

Based on the problems in the mentioned papers, this paper proposes the Pre-judgment Conflicts Search (SPC) algorithm. The algorithm runs in two layers: the higher layer assigns tasks to robots based on their priority, and the lower layer uses improved A* algorithm to plan paths for each robot. The specific contributions are as follows:

(1) By learning from the processing mechanism of the CBS, the SPC combines the collision detection with the path planning, forming an iterative process of planning, detecting, and marking.

(2) SPC can receive, analyse and process newly added tasks in real time, dynamically adjust the task planning strategy, realize the function of planning tasks dynamically.

(3) SPC performs loop iterations at predefined time intervals, selecting the next set of tasks to plan from the task queue each time window. Once a set of tasks is obtained, the path planning algorithm is used to generate the robots' movement paths, taking into account factors such as task priority, robot status, and environmental obstacles. After completing path planning, the algorithm records the completion time as a time benchmark for planning the next set of tasks, ensuring the continuity of tasks and enabling the robots to efficiently execute a series of tasks.

2. PROBLEM STATEMENT

In smart warehouses, we will use the results of multi-robot task balancing to plan continuous paths. The multi-robot path planning in the warehousing environment is modelled as a two-dimensional grid. The model consists of robots (AGVs), shelves, storage area, picking stations, and charging stations. The warehouse map from COAP (Center for Optimization Algorithm Platform, data download link: *https://www.coap.online/ competitions/2*) is simplified, and the warehouse layout is depicted in Fig. 1.

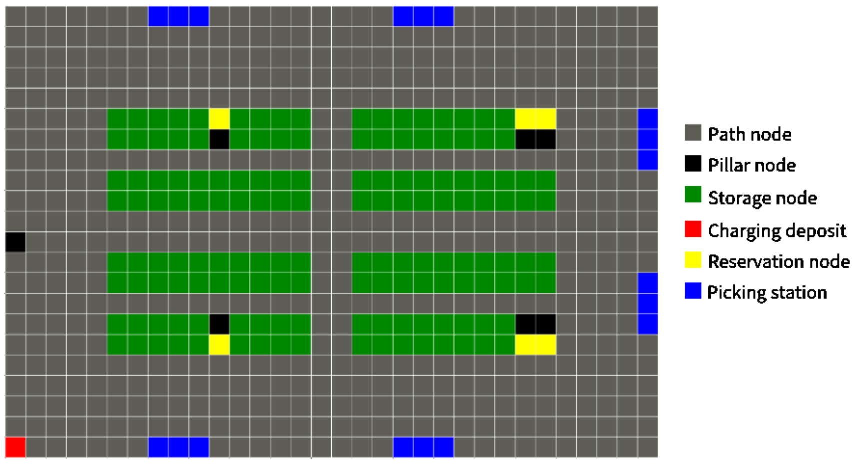

Figure 1: Warehouse map.

AGVs can pass freely on the path nodes; they can reach the storage nodes, such as placing pallets or ordinary shelves; they can't reach the pillar nodes; picking robots pack the goods from the conveyor belt out of the warehouse at the picking station nodes, and every picking station has more than one shelf stop; the charging deposit can allow up to 20 robots to be charged at the same time.

3. SPC ALGORITHM

3.1 Algorithm introduction

SPC is divided into two layers to achieve efficient management and path planning of robot tasks, thereby ensuring that every robot can continuously and stably perform tasks in complex and changeable environments.

(1) Higher layer structure: task management and priority setting

(a) Dynamic task reception: The higher layer structure is first responsible for receiving new tasks from the warehouse management system in real time. Then, the algorithm will quickly analyse the task, including tasks type, starting location, target location, urgency and other information. This real-time receiving mechanism ensures that the system can quickly respond to new job requirements.

(b) Priority setting: Based on the results of task analysis, the higher layer structure will set the priority for each task according to a priority setting mechanism. Priority is set based on the urgency, importance and other relevant factors of the task to ensure that urgent and important tasks are handled first.

The task from a shelf to a picking station is set to 4 for its priority because a robot with a shelf is difficult to start and stop; the task from a picking station to a shelf is assigned 3; the task from the robot's initial position to a shelf is set to 2; the task from a shelf to another shelf is 1; the task from a shelf to the charging station is 0.

To reduce the distance travelled by robots, calculate the Manhattan distance from the starting point to the goal point of a task, and a task has the highest priority if its corresponding path is the longest. SPC plans paths according to the order of the tasks when these tasks have the same priority.

(c) Continuous operation mode: One of the core goals of the higher layer structure is to ensure that robots can achieve continuous operation mode. This mechanism ensures that a robot can quickly transition its next task after completing a task, avoiding free time or conflicts between tasks.

(2) Lower layer structure: path planning and execution

The lower layer uses the improved A* algorithm to find the optimal path from starting point to target point. When planning a path for a robot, safety measures, such as preventing vertex collisions and edge collisions, ensure the stable operation of the robot in complex environments.

Vertex collision refers to that two robots reach the same location simultaneously at a certain moment; Edge collision refers to that two robots swap their positions simultaneously at a certain moment. As shown in Fig. 2.

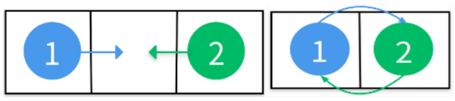

Figure 2: Two collision types (Vertex collision – left, Edge collision – right).

3.2 Algorithm design

SPC is a nearly real-time path planning algorithm. Once one robot path is successfully planned, the 2D grid map records its travelled positions and the corresponding moments. The A* algorithm is modified by preventing vertex collisions and edge collisions between the planning path and the successfully planned paths during the searching process.

(1) Algorithm definition

(a) **Objective**: Set up an intelligent warehouse with *K* robots, each of which must perform a series of tasks sequentially, and plan global conflict-free routes for the robots.

(b) **Conflicts**: Vertex collision is expressed as (k_i, k_j, v, t), which means that two robots k_i and k_j arrive at the vertex v at the moment t; Edge collision is expressed as (k_i, k_j, e, t), which means that two robots k_i and k_j exchange their position through the edge e at the moment t. These collisions generate conflicts in the path planning. During the process of planning a path for a robot, only the points without vertex collisions and edge collisions can be added into its path, and in this way the robot can arrive at its target position. A robot must wait for a moment when a path without collisions cannot be found.

(2) Improved A* Search Pseudo-Code

Algorithm one is the pseudo-code for modified A* algorithm, and the steps are as follows:

Input: $task$, $planned\_paths$. $task$ contains start point $start$, target point $goal$ and start time $starttime$. $planned\_paths$ marks track points that have been successfully planned, and each point includes the time $time$, location $location$,and name of agent $agentName$.

Output: planned robot trajectories $result\_solution$.

(a) Get the start point $start$, the target point $goal$, and the start time $starttime$ of the task (line 1).

(b) Initialize $start.edgenum = 0$ which denotes the distance traveled from the start point, and which is used to calculate the walking time; $start.father = null$; $start.cost = h(start)$, the heuristic function $h(start)$ is the Manhattan distance from $start$ to $goal$ (line 2).

(c) Initialize $open$ and $close$ to null (line 3).

(d) Add $start$ to $open$ (line 4).

(e) When $open$ is not empty, loop through steps from (f) to (g).

(f) Ascending sort $open$ list based on the distance computed by the heuristic. Get the first point pt in the $open$ list, and remove pt from the $open$ (line 6); if pt is the target point, trace the $father$ node from pt to $start$, and put this path into $task.path$, and $task.path$ contains $time$ and position (x, y) of the robot for each step, and update the task start time, and return the result of the task planning $task.path$, and the algorithm ends (lines 7-9).

(g) If pt is not a target point $goal$, add pt to the $close$ list (line 10), update the time of pt node (line 11), call $no\_collision$ function to get pt's valid neighbours node $valid\_neighbors$ (line 12); travers $valid\_neighbors$, determine whether npt is in the $close$ list, if it is, and indicates that the point has been visited. Then stop this loop and continue traversing the next node (lines 14-15); otherwise, continue to determine whether npt is in the $open$ list; if it is not (line 16), add npt to $open$ list (line 17) and set the $father$ node of npt to pt,update $edgenum$ and $cost$ of npt (lines 20-22). Otherwise, determine whether the value $edgenum$ of pt plus 1 is greater than or equal to the value $edgenum$ of npt, if it is less than or equal, executes (lines 20-22), otherwise end this loop.

(h) If $open$ is empty, return $fail$ and the algorithm ends (line 23).

Algorithm 1: Improved A* _Search($task$, $planned\_paths$)

```
1 $start \leftarrow task.start, goal \leftarrow task.goal, starttime \leftarrow task.time$
2 $start.edgenum \leftarrow 0, start.father \leftarrow null, start.cost \leftarrow h(start)$
3 $open \leftarrow \phi, close \leftarrow \phi$
4 $open.add(start)$
5 while $open! = empty$ do
6    $pt = (open.sort()).pop(0)$
7    if $pt == goal$ then
8        Trace the $father$ node step by step from $pt$ to the starting point $start$
             and put this path into $task.path$
9        return $task.path$
```

```
10 $close.add(pt)$
11 $time = starttime + pt.edgenum$
12 $valid\_neighbors = no\_collision(pt, time, planned\_paths)$
13 **foreach** $npt\ in\ valid\_neighbors$ **do**
14    **if** $npt\ in\ close$ **then**
15       **continue**
16    **if** $npt\ not\ in\ open$ **then**
17       $open.add(npt)$
18    **elif** $pt.edgenum + 1 >= npt.edgenum$
19       **continue**
20    $npt.father \leftarrow pt$
21    $npt.edgenum \leftarrow pt.edgenum + 1$
22    $npt.cost \leftarrow npt.edgenum + h(npt)$
23 **return** $fail$
```

(3) Collision checking strategy Pseudo-Code

Searching information about surrounding neighbour nodes, designing point collision and edge collision avoidance strategies to implement conflict prevention. The algorithm steps are as follows:

Input: The position of the current node pt, time $time$, and track points that have been successfully planned $planned\_paths$.

Output: Effective neighbour nodes of pt without point conflicts and edge conflicts.

(a) Get the information of the four neighbour nodes includes above, below, left, and right of the current node and the current node (line 5). Initialize $valid.neighbors$ to null used to store conflict-free neighbour nodes (line 6).

(b) Traverse $neighbors$ and call $vertex\_collision$ function. Determine whether npt is marked in $planned\_paths$ at $time + 1$. If it is, $vertex\_flag$ is true, there are point conflicts. Stop this cycle and continue traversing the next node. Otherwise, continue to call the $edge\_collision$ function to determine whether the current robot's trajectory from pt to npt has edge conflicts with the robots' trajectory marked in $planned\_paths$. If it is, $edge\_flag$ is true, end this loop and continue to traverse the next node.

(c) If there are no point conflicts and edge conflicts, add npt to $valid\_neighbors$.

(d) After traversing all neighbour nodes, return $valid\_neighbors$.

Algorithm 2: Collision checking strategy

```
1 $vertex\_flag \leftarrow vertex\_collision(pt, time, planned\_paths)$
2 $edge\_flag \leftarrow edge\_collision(pt1, pt2, time, planned\_paths)$
3 $neighbors \leftarrow get\_neighbors(pt)$
4 **function** $no\_collision(pt, time, planned\_paths)$
5   $neighbors = get\_neighbors(pt)$
6   $valid\_neighbors \leftarrow \phi$
7   **for** $npt\ in\ neighbors$ **do**
8      $vertex\_flag = vertex\_collision(npt, time + 1, planned\_paths)$
9      **if** $vertex\_flag$ **then**
10        **continue**
11     $edge\_flag = edge\_collision(pt, npt, time, planned\_paths)$
12     **if** $edge\_flag$ **then**
13        **continue**
14     $valid\_neighbors.$add(npt)
15 **return** $valid\_neighbors$
```

(4) Continuous path planning Pseudo-Code

Input: $agents\_plans$ with dictionary data type. The keys in the dictionary correspond to names of the agents $agent_n$, and the values stand for the series of tasks executed by each agent. Each task contains the starting position $start$, target position $goal$, and the task start time

$time$. The task start time is initially set to 0. The tasks for each agent have a specific order, and the next task can begin only after its earlier task is finished. The completion time of the earlier task serves as the earliest start time for the later task.

Output: globally conflict-free paths.

Initialize $Tasks$, $result\_solution$ and $planned\_paths$ to be empty, and $step\_time$ to be 0 (line 1). In continuous path planning, each agent needs to perform a series of tasks. It traverses $agents\_plans$ to obtain the first task for each agent, insert it into $Tasks$, and deletes the removed tasks from $agents\_plans$ (lines 2-3). If $Tasks$ is not empty, loop to execute (lines 5-21).

First, sort the Tasks according to the task priority and assign it to $TaskSorted$ (line 5). Then traverse $agentName, task$ in $TaskSorted$ (line 6). If the task start time is not equal to $step\_time$ (line 7), it means that it has not yet been the task's turn to plan the path. Otherwise, call Algorithm 1 to plan a path for the task, and the path planning result is assigned to the $solution$, which mainly includes time and corresponding location information (line 9). If the $solution$ is fail, the path planning fails, and the task waits for a moment. Then, updates the task start time, and saves the path trajectory (lines 11-12). If the $solution$ is not fail, path planning is successful, and removes the task from $Tasks$, record the path planning results, and save the successfully planned path trajectory points to $planned\_paths$ (lines 14-16). Determine whether $agents\_plans[agentName]$ is empty. If it is not, take the agent's $next\_plan$, delete $next\_plan$ from $agents\_plans[agentName]$ (line 18). Update the task start time of $next\_plan$ to the completion time of the previous task plus 1 (line 19), and put $\{agentName: next\_plan\}$ into $Tasks$ (line 20). If $TaskSorted$ has been traversed, the system task planning time is plus 1 (line 21). If $Tasks$ is empty, all tasks are planned and return the global collision-free path $result\_solution$ is (line 22).

Algorithm 3: Continuous path planning

```
Input: $agents\_plans \leftarrow \{"agent_0": [plan_{01}, \dots, plan_{0i}], \dots, "agent_n": [plan_{n0}, \dots, plan_{nj}]\}$
Output: Global conflict-free path
1 $Tasks \leftarrow \phi$; $result\_solution \leftarrow \phi$; $step\_time \leftarrow 0$; $planned\_paths \leftarrow \phi$
2 foreach $agentName, plans\ in\ agents\_plans.items()$ do
3   $Tasks.update(\{agentName : plants.pop(0)\})$
4 while $Tasks\ is\ not\ empty$ do
5   $TaskSorted \leftarrow Tasks.sort()$
6   foreach $agentName, task\ in\ TaskSorted$ do
7     if $task["time"] != step\_time$ then
8       continue
9     $solution \leftarrow A*\_search(task, planned\_paths)$ //Algorithm 1
10    if $solution\ is\ fail$ then
11      $task.time \leftarrow step\_time + 1$
12      $result\_solution[agentName].append($
                  $\{"time" : step\_time, "start" : task.start, "goal" : task.goal\})$
13    else
14      $Tasks.remove(task)$
15      $result\_solution[agentName].append(solution)$
16      Add coordinate points from $solution$ to $planned\_paths$
17      if $agents\_plans[agentName] != Empty$ then
18        $next\_plan \leftarrow agents\_plans[agentName].pop(0)$
19        $next\_plan["time"] \leftarrow solution[len(solution) - 1].time + 1$
20        $Tasks.update(\{agentName : next\_plan\})$
21  $step\_time \leftarrow step\_time + 1$
22 returns $result\_solution$
```

3.3 Sample illustration

When the start position of a task or its goal position is surrounded by obstacles, or the road between the start position of a task and its goal position is inaccessible as shown in Fig. 3, the path planning fails.

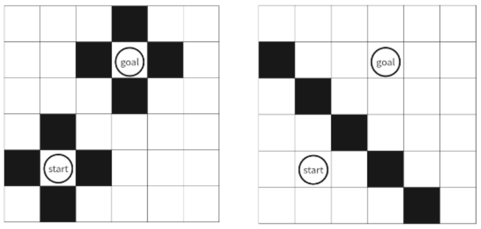

Figure 3: Inaccessible road between the start location and target location of a task.

Assuming the road from the start position of the task to its goal position is unblocked, the worst-case scenario for SPC is that only when the current robot completes the path planning for its task and gives way to the road, a conflict-free path be planned for the later robot. Taking Fig. 4a as an example, there are three robots represented by r1, r2, and r3 in the 2D grid map, and the start positions of these robots are s1, s2, and s3, and the execution order of robots' tasks is s1→g11→g12, s2→g21→g22, and s3→g31→g32. Only when robot r1 completes the planning path of s1→g11→g12 and gives way to the path, it plans the path for r2, as shown in Fig. 4b, and r1 stays at position g12 after completing the planning task, and so on r2, r3.

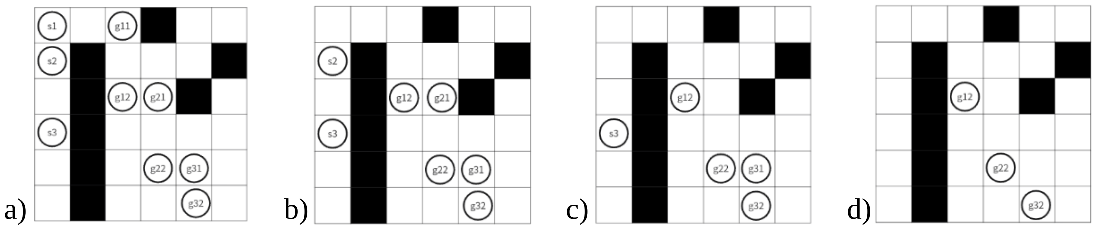

Figure 4: Worst case path planning.

3.4 Algorithm complexity analysis

(1) Time complexity analysis

Assuming that the maximum length of a single planning path is L (it is the length plus the width of a warehouse, then it is the longest distance walked by a robot at a time), i.e., there are L points on it. For each point in the $open$, it is necessary to find four neighboring points, so it has four branches at a time, and the maximum depth of the layer is L. Therefore, its time complexity is $1 + 4^1 + 4^2 + \cdots + 4^L \approx 4^{L+1}$, so the time complexity of n robots is $4^{L+1}n$.

There are also comparisons among points in previously generated paths, and the number of comparisons per path is $3L$ (to determine point collisions and edge collisions), so the total number of comparisons $= 0 + 3(L + 2L + \cdots + (n-1)L) = (1 + n - 1)\frac{n-1}{2}3L = 3n(n-1)L/2$.

Adding these time complexities yields the formula $4^{L+1}n + 3n(n-1)L/2$, which shows that for a fixed warehouse size, the time complexity is a polynomial function of the number of robots while the complexity of CBS is an exponential function.

(2) Analysis of space complexity.

The space complexity is like the time complexity, and it is 4^{L+1} when planning a path for a robot; In addition, the space occupied by each planned path is proportional to L, and since there are n paths, Therefore, the entire space complexity is equal to $4^{L+1} + nL$, which is linearly related to n.

4. SIMULATION

To verify the effectiveness of SPC, this section will compare the path length and planning time in random maps and warehouse maps, as well as conduct simulation experiments of multi-robot sequential path planning in intelligent warehouses. The running environment is Intel® Xeon® CPU E5-2650 v4 @ 2.20 GHz processor, 256 GB RAM, Window 10 system, and Python code.

4.1 Random maps

We model the map as a two-dimensional grid map and set the number of robots to 10, 20, 30, 40, 50, and 60 under the same robot density. We set the robot density to 0.025 and the obstacle density to 10 % in 20×20, 28×28, 35×35, 40×40, 45×45 and 50×50 environment maps. SPC and CBS are compared in terms of two dimensions: path length and planning time. Fig. 5 shows environment maps randomly generated.

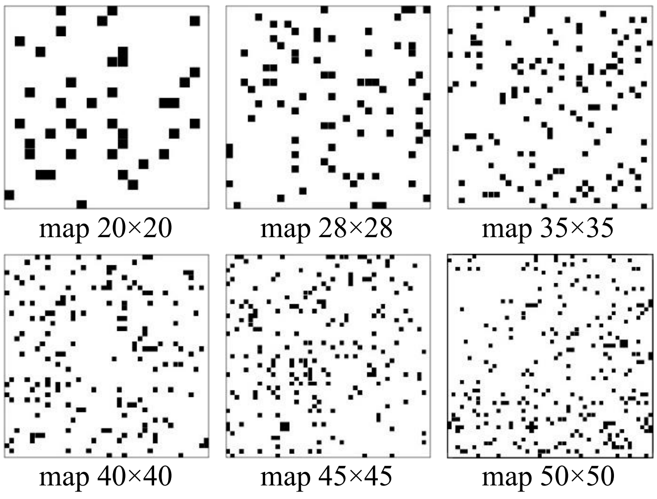

Figure 5: Random maps of varied sizes.

(1) Comparison experiment on planning-path length

Comparing the path length of SPC and that of CBS under the same robot density condition. SPC_c denotes task prioritization according to the task types, and SPC_d denotes task prioritization according to the task distance. Since the planning time of the CBS is about 30 hours with 40 robots, only the path length of CBS is given for the number of robots up to 40, and Fig. 6 shows the comparison results. From the perspective of path length, SPC is within 1.6 % relative difference compared with CBS when the number of the robots is less than 40.

Table I: Comparison of path lengths at different map sizes.

Map size	**Number of robots**	**Length of path plan** (m)			
		CBS	**SPC_c**	**SPC_d**	**Bias**
20×20	10	128	128	128	0 %
28×28	20	380	386	386	1.58 %
35×35	30	662	664	668	0.30 %, 0.91 %
40×40	40	1090	1107	1107	1.56 %
45×45	50	-	1605	1599	-
50×50	60	-	2133	2133	-

Note: Bias = $(\text{Length}_{\text{SPC}} - \text{Length}_{\text{CBS}}) / \text{Length}_{\text{CBS}} \times 100\,\%$.

(2) Comparison experiment on path-planning time

Comparing path-planning time between SPC and CBS under the same robot density condition, and Fig. 7 shows the comparison effect. With 40 robots, the path-planning time of

CBS is 106,792.76 seconds, which is 37,210 times longer than that of SPC. From the perspective of path-planning time, SPC is significantly better than CBS.

In Figs. 6 and 7, the path length and planning time of SPC_c and SPC_d are nearly the same, and it indicates that the SPC has good robustness. In real logistic scenarios, the way of prioritizing tasks can be chosen according to the demand. Both the warehouse map and smart warehouse simulation experiments in this paper prioritize tasks according to these tasks' types.

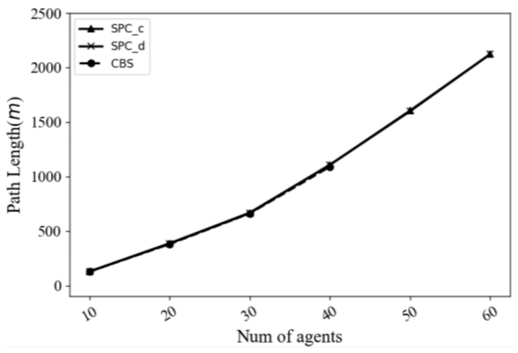

Figure 6: Path length comparison.

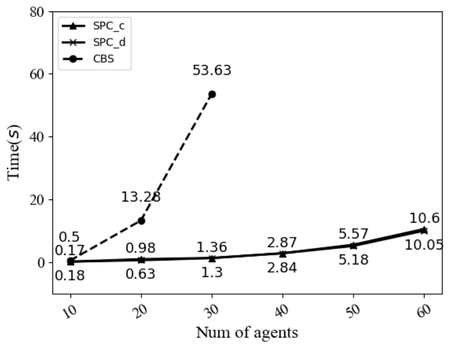

Figure 7: Comparison of planning time.

4.2 Warehouse map

The warehouse maps come from the benchmark data (*https://movingai.com/benchmarks/mapf/index.html*) (warehouse-10-20-10-2-2) in Stern [23], and the map size is 170×84, and there are 9776 free passable nodes. We use SPC to plan the paths for 98 agents (compare SPC with Continuous-time CBS (CCBS) from literature [24]) with the settings from the 25 scenario files provided in the benchmark data. Fig. 9 illustrates details of the path planning time.

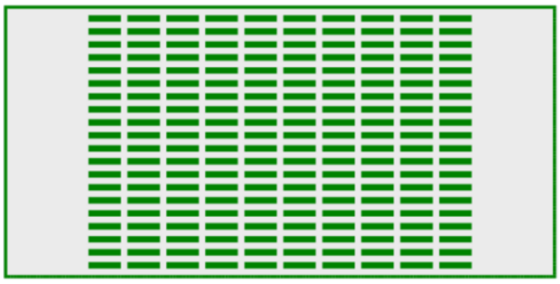

Figure 8: Warehouse-10-20-10-2-2.

Andreychuk et al. [24] proposed an improved CCBS (Continuous-time CBS) algorithm which has an almost 0 % success rate in planning paths for 98 robots, whereas SPC can plan a conflict-free path for each robot in 4 minutes.

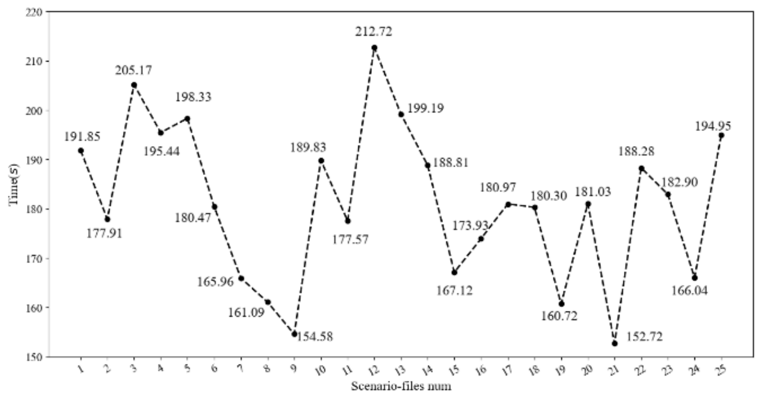

Figure 9: Warehouse-10-20-10-2-2 path-planning time.

4.3 Intelligent warehouse simulation experiment

The data for AGVs, orders, and inventory in an intelligent warehouse model is downloaded from the COAP platform. There are 20 AGVs, 140 shelves to be picked, and twenty picking stations. Each robot performs a series of tasks in a specified order. The assignment of the shelves to each robot and the order of service details are shown in Table II.

Table II: AGV travel routes.

AGV ID	Shelves service order
0	0→18→11→135→96→110→139→0
1	0→64→49→31→94→131→106→0
2	0→47→22→112→3→123→0
3	0→102→72→93→140→88→41→0
4	0→38→114→121→37→137→40→0
5	0→23→39→34→20→86→0
6	0→108→73→91→52→71→29→0
7	0→30→134→132→117→87→45→0
8	0→1→78→13→97→113→81→43→0
9	0→5→10→16→92→50→136→2→0
10	0→35→130→90→33→21→89→12→0
11	0→138→46→36→55→133→111→76→25→0
12	0→83→99→53→51→125→115→27→118→103→0
13	0→122→77→104→105→79→7→56→4→0
14	0→68→75→65→61→48→14→0
15	0→59→98→124→95→9→85→120→119→0
16	0→44→19→116→63→57→17→62→0
17	0→15→67→82→128→58→100→8→126→129→0
18	0→127→101→69→70→107→74→66→24→42→60→0
19	0→109→6→28→54→32→84→80→26→0

Shelf selection, picking station assignment and the balanced task allocation of robots have been studied in an in-submission paper. The visualization of shelf choices and assignment to picking stations is respectively illustrated in Figs. 10 and 11.

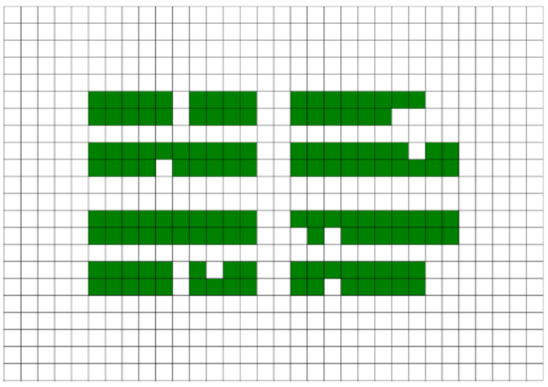
Figure 10: Shelf selection visualization.

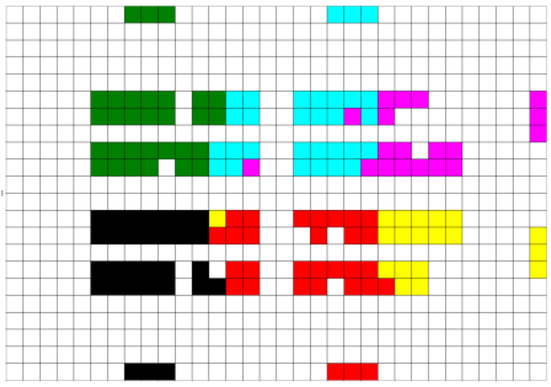
Figure 11: Shelf selection visualization.

Each robot starts from a random location to its first shelf, carries the shelf to the designated picking station, back to the original place after goods are picked, and then carries the second shelf until it has carried all the selected shelves, and finally it returns to the charging station. SPC algorithm plans a globally conflict-free route for each robot, with a path length of 5,744 meters and its path-planning time of 28.73 seconds, and Fig. 12 displays a screenshot of the simulation result. The code can be downloaded at *https://github.com/ljbwu/SPC.git*.

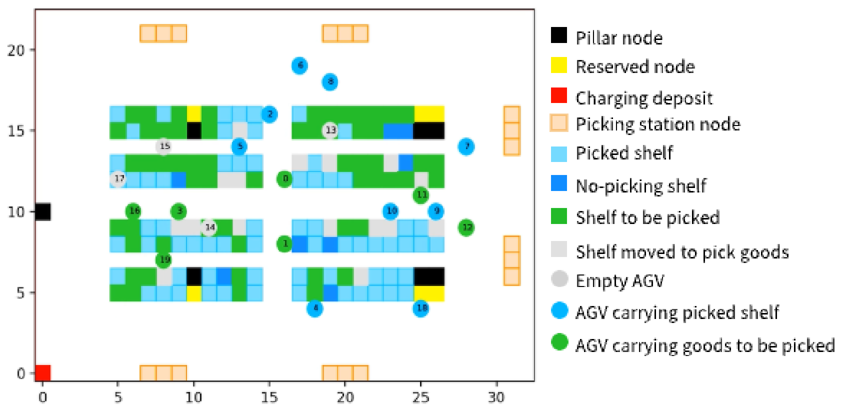

Figure 12: Global path planning simulation effect.

5. CONCLUSION

Continuous multi-robot path planning is crucial for intelligent warehouse systems. Compared with CBS algorithm, the SPC algorithm shifts from conflict resolution to conflict prevention. This significantly reduces the algorithm's iteration count and computational burden, enhancing the efficiency of path search to ensure faster attainment of optimal solutions and comprehensively improve the overall system performance. The SPC algorithm possesses the capability for real-time updates and adjustment of task priorities, enabling flexible adaptation to task requirements. Through the iterative loop with time window settings, it achieves a continuous operation mode. Through experimental comparisons with CBS, it was found that SPC exhibits superior continuous path planning capabilities and scalability. With an increasing number of robots, the algorithm continues to efficiently plan paths close to optimal within short time. As a result, it enhances warehouse logistics efficiency and reduces operational costs.

In logistic scenarios, the obstacles do not stay at the same place all the time, and thus the future research will be focused on how to avoid dynamical obstacles.

ACKNOWLEDGEMENT

This research was supported by National Natural Science Foundation Project (72101033), Key Project of Science and Technology Plan of Beijing Municipal Education Commission (KZ202210037046) and Excellent Science and Technology Innovation Team Project in Tongzhou District (CXTD2023010).

REFERENCES

[1] Ajidarma, P.; Nof, S. Y. (2021). Collaborative detection and prevention of errors and conflicts in an agricultural robotic system, *Studies in Informatics and Control*, Vol. 30, No. 1, 19-28, doi:10.24846/v30i1y202102

[2] Chai, G. F.; Xia, Y. Z. (2023). Multi-robot path optimization and simulation for multi-route inspection in manufacturing, *International Journal of Simulation Modelling*, Vol. 22, No. 1, 121-132, doi:10.2507/IJSIMM22-1-CO1

[3] Sárdi, D. L.; Bóna, K. (2021). City logistics analysis of urban areas: an analytic hierarchy process based study, *Journal of System and Management Sciences*, Vol. 11, No. 2, 77-105, doi:10.33168/JSMS.2021.0206

[4] Legowo, N.; Wijaya, W. (2022). Measurement of warehouse management system performance using IT-BSC method in the FMCG industry, *Journal of System and Management Sciences*, Vol. 12, No. 5, 230-251, doi:10.33168/JSMS.2022.0514

[5] Isik, M.; Sahin, C.; Hamidy, S. M. (2023). Novel dispatching rules for multiple-load automated guided vehicles, *International Journal of Simulation Modelling*, Vol. 22, No. 1, 76-87, doi:10.2507/IJSIMM22-1-632

[6] Hasan, A. H.; Mosa, M. A. (2018). Multi-robot path planning based on max–min ant colony optimization and D* algorithms in a dynamic environment, *2018 International Conference on Advanced Science and Engineering*, 110-115, doi:10.1109/ICOASE.2018.8548805

[7] Raheem, F.; Ibrahim, U. (2019). Interactive heuristic D* path planning solution based on PSO for two-link robotic arm in dynamic environment, *World Journal of Engineering and Technology*, Vol. 7, No. 2, 80-99, doi:10.4236/wjet.2019.71005

[8] Zeng, Z.; Sun, W.; Wu, W.; Xue, M.; Qian, L. (2019). An efficient path planning algorithm for mobile, *2019 IEEE 15th International Conference on Control and Automation*, 487-493, doi:10.1109/ICCA.2019.8899975

[9] Raheem, F. A.; Abdulkareem, M. I. (2020). Development of A* algorithm for robot path planning based on modified probabilistic roadmap and artificial potential field, *Journal of Engineering Science and Technology*, Vol. 15, No. 5, 3034-3054

[10] Krell, E.; Sheta, A.; Balasubramanian, A. P. R.; King, S. A. (2019). Collision-free autonomous robot navigation in unknown environments utilizing PSO for path planning, *Journal of Artificial Intelligence and Soft Computing Research*, Vol. 9, No. 4, 267-282, doi:10.2478/jaiscr-2019-0008

[11] Luo, Q.; Wang, H.; Zheng, Y.; He, J. (2020). Research on path planning of mobile robot based on improved ant colony algorithm, *Neural Computing and Applications*, Vol. 32, 1555-1566, doi:10.1007/s00521-019-04172-2

[12] Kennedy, J.; Eberhart, R. (1995). Particle swarm optimization, *Proceedings of ICNN'95 – International Conference on Neural Networks*, 1942-1948, doi:10.1109/ICNN.1995.488968

[13] Tang, B. W.; Xiang, K.; Pang, M. Y.; Zhu, Z. X. (2020). Multi-robot path planning using an improved self-adaptive particle swarm optimization, *International Journal of Advanced Robotic Systems*, Vol. 17, No. 5, 19 pages, doi:10.1177/1729881420936154

[14] Mirjalili, S.; Lewis, A. (2016). The whale optimization algorithm, *Advances in Engineering Software*, Vol. 95, 51-67, doi:10.1016/j.advengsoft.2016.01.008

[15] Chhillar, A.; Choudhary, A. (2020). Mobile robot path planning based upon updated whale optimization algorithm, *10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 684-691, doi:10.1109/Confluence47617.2020.9058323

[16] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots, *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, 500-505, doi:10.1109/ROBOT.1985.1087247

[17] Koren, Y.; Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, 1398-1404, doi:10.1109/ROBOT.1991.131810

[18] Zhu, C. Y. (2023). Intelligent robot path planning and navigation based on reinforcement learning and adaptive control, *Journal of Logistics, Informatics and Service Science*, Vol. 10, No. 3, 235-248, doi:10.33168/JLISS.2023.0318

[19] Kulathunga, G. (2022). A reinforcement learning based path planning approach in 3D environment, *Procedia Computer Science*, Vol. 212, 152-160, doi:10.1016/j.procs.2022.10.217

[20] Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N. (2015). Conflict-based search for optimal multi-agent path finding, *Artificial Intelligence*, Vol. 219, 40-66, doi:10.1016/j.artint.2014.11.006

[21] Wang, Y. J.; Liu, X. Q.; Leng, J. Y.; Wang, J. J.; Meng, Q. N.; Zhou, M. J. (2022). Study on scheduling and path planning problems of multi-AGVs based on a heuristic algorithm in intelligent manufacturing workshop, *Advances in Production Engineering & Management*, Vol. 17, No. 4, 505-513, doi:10.14743/apem2022.4.452

[22] Sebo, J.; Busa Jr., J. (2020). Comparison of advanced methods for picking path optimization: case study of dual-zone warehouse, *International Journal of Simulation Modelling*, Vol. 19, No. 3, 410-421, doi:10.2507/IJSIMM19-3-521

[23] Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K.; Bartak, R.; Boyarski, E. (2019). Multi-agent pathfinding: definitions, variants, and benchmarks, *Proceedings of the 12th International Symposium on Combinatorial Search*, 151-158, doi:10.1609/socs.v10i1.18510

[24] Andreychuk, A.; Yakovlev, K.; Boyarski, E.; Stern, R. (2021). Improving continuous-time conflict based search, *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 11220-11227, doi:10.1609/aaai.v35i13.17338