



# Decentralized Multi-Agent Path Finding in Warehouse Environments for Fleets of Mobile Robots with Limited Communication Range

Abderraouf Maoudj<sup>(✉)</sup> and Anders Lyhne Christensen

SDU Biorobotics, MMMI, University of Southern Denmark (SDU), Odense, Denmark  
{abma, andc}@mmmi.sdu.dk

**Abstract.** Mobile robots have already made their way into warehouses, and significant effort has consequently been devoted to designing effective algorithms for the related multi-agent path finding (MAPF) problem. However, most of the proposed MAPF algorithms still rely on centralized planning as well as simplistic assumptions, such as that robots have full observability of the environment and move at equal and constant speeds. The resultant plans thus cannot be executed directly on physical robots where these assumptions generally do not hold. To mitigate these issues, we consider the decentralized partially observable multi-robot setting where robots do not have access to the full state of the world. Instead, each robot coordinates with neighbors within a limited communication range. In the proposed approach, each robot independently plans its own path using A\* without taking into account other robots, and the robots then solve potential conflicts locally as they occur. Experimental results obtained in various benchmark scenarios confirm that the proposed decentralized approach is effective and scales well to large numbers of robots.

## 1 Introduction

With the rapid development of low-cost sensors and computing devices, it is becoming increasingly feasible to deploy large-scale systems of mobile transportation robots in industrial environments. Nowadays, many industrial applications benefit from fleets of mobile robots transporting goods and materials between workstations and storage pipes [27]. The increased use of robot fleets has given rise to a number of challenging optimization problems, such as multirobot path planning [24] and multirobot scheduling [1].

Planning conflict-free paths for a team of mobile robots, known as the *multi-agent path finding* (MAPF) problem, remains a major challenge [15, 20]. Given a set of agents, each with a pre-specified initial location and a pre-specified goal location in a known environment, MAPF is concerned with finding collision-free paths for the agents such that certain objectives are minimized. MAPF is inspired by real-world applications, such as automated warehouses [11], traffic management [3], and valet parking [12].

The original version of this chapter was revised: The quantitative results have been revised to accurately reflect the research findings. The correction to this chapter is available at [https://doi.org/10.1007/978-3-031-20176-9\\_34](https://doi.org/10.1007/978-3-031-20176-9_34)

MAPF is NP-hard to solve optimally [25]. As a result, a significant amount of research has been conducted and the resulting state-of-the-art algorithms can be divided into four categories [7]:

**Systematic search algorithms** are centralized planning approaches, which enable finding all possible solutions, including an optimal one. In this category, numerous algorithms have been proposed, such as the branch-and-cut-and-price (BCP) algorithm [5], pairwise symmetry reasoning [8], conflict-based search (CBS) algorithms and their variants [8, 9]—which are currently among the most popular algorithms for solving the MAPF problem optimally. Although these planners achieve optimal or bounded sub-optimal solutions, they often suffer from a computational complexity that increases exponentially with the problem size.

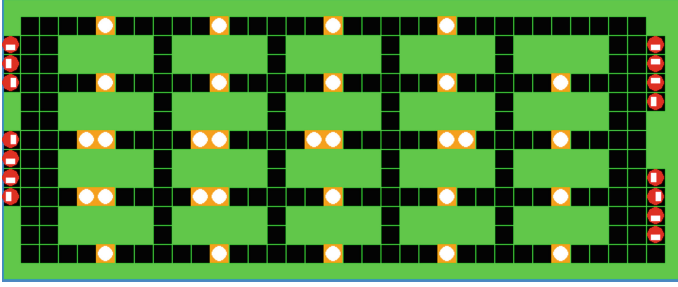
**Rule-based algorithms**, in which the agents move step-by-step following ad-hoc rules [13]. For instance, the graph abstraction approach [16], the conflict classification-based algorithm [26], biconnected graphs [21], and parallel-push-and-swap (PPS) [17]. These algorithms are polynomial-time but can still fail to find solutions within a reasonable amount of time for large instances.

**Learning-based algorithms** use reinforcement learning techniques for finding cooperative and competitive behaviors for solving conflicts [15]. Different learning-based algorithms have been proposed in literature for solving MAPF, see for instance [2, 18]. Even though learning-based approaches have proven to be more robust to uncertainties in practical applications than the algorithms discussed above, they do not provide guarantees on solution quality [13, 18].

**Priority-based algorithms**, in which the MAPF problem is decomposed into a series of single-agent path planning problems, where the agents plan their paths sequentially according to a priority scheme. Popular algorithms include the prioritized planning algorithm [14], searching with consistent prioritization [10], the hierarchical cooperative A\* approach (HCA), and priority inheritance with backtracking [13]. The prioritized planning algorithm provides a practical solution to applications with large numbers of robots. However, the quality of the resulting solutions depends on the choice of the prioritization scheme, especially in dense environments with limited path choices [23].

The algorithms described above rely on simplistic assumptions and have different objectives. Most of them assume that robots always move at their nominal speed, ignore kinematic constraints, and do not take into account imperfect plan-execution capabilities [4]: in practical scenarios, a robot may need to slow down or come to a complete halt when facing a challenging situation, such as entering a narrow corridor or turning on the spot. The execution will therefore deviate from the plan found offline, and variation in the robots' speeds can thus significantly affect the applicability of these approaches.

To overcome the aforementioned challenges, we propose a decentralized approach based on online conflict resolution, wherein each agent autonomously plans its path using A\* while initially ignoring the other agents. Our approach does thus not require the robots to have complete information about the state of the environment. Instead, we consider that robots operate in a partially-observable



**Fig. 1.** Example of a warehouse layout. (Color figure online)

world, where each robot can only communicate with neighbors within its vicinity. Additionally, the proposed approach can be used in scenarios where agents have a sequence of goals, which makes it promising for practical scenarios, where agents are continually assigned new goal locations and are required to compute paths online [2].

## 2 Environment Model and Assumptions

In many practical applications, the layout of a warehouse is fixed, and robots can only move along a predefined roadmap [24]. Accordingly, in this study, we consider automated warehouses with predefined roadmaps, in which a set of  $m$  mobile robots  $\{r_1, \dots, r_m\}$  perform their assigned transportation tasks. The robots are assumed to know the roadmap and their own position and orientation in the map. Figure 1 illustrates an example of a warehouse layout modelled as a  $36 \times 15$  grid map: the red circles and yellow circles represent the robots and their designated targets, respectively, the green squares represent obstacles, and the black squares represent free space where the robot can move.

In real-world scenarios, wireless communication can be noisy and the robots often have a limited field-of-view [2, 19]. Therefore, to reduce the gap between simulation and real-world scenarios, we assume that each robot can only access the state of its neighbors within limited communication range (2 squares). At each time step, if robot  $j$  is in communication range of robot  $i$ , we say that robot  $j$  is in robot  $i$ 's neighborhood  $j \in N_i^t$ .

A warehouse layout can be abstracted into an undirected graph  $G = (V, E)$ , where nodes  $V$  correspond to locations arranged in the grid and the edges  $E$  correspond to straight lines between locations that can be traversed by the robots. At every time step  $t$ , each robot  $i$  occupies one of the graph nodes  $n_i^t$ , referred to as the *location* of that robot at time  $t$ , and can choose to perform an *action*  $a_i$ . The action can be either `move` to an adjacent node or `wait` in its current node. The multi-agent path finding problem consists of computing collision-free paths for the team of agents from their current locations to their respective targets. The objective is to minimize the *sum-of-costs* (or *flow time*), that is, the sum over all agents of the time steps required to reach their target locations [22].

### 3 Proposed Approach

In this section, we present our decentralized cooperative multi-agent path finding approach (DCMAPF) enabling large-scale systems of autonomous mobile robots to operate effectively in shared warehouse environments.

The proposed DCMAPF is presented in Algorithm 1. DCMAPF has two phases: (i) *Path planning* and (ii) *Execution and motion coordination*. In the first phase, the robots individually plan the shortest paths from their initial location to their targets using A\*. In the second phase, robots follow their planned paths while detecting and resolving local conflicts at each time step. To reduce the complexity of local coordination, we introduce a *leader-follower* concept for adjacent robots moving in the same direction. At time step  $t$ , robot  $k$  is a follower of robot  $i$  if  $n_k^{t+1} = n_i^t$ . Since followers relay messages, a leader can have an arbitrary number of followers, and the *followers* of robot  $i$  consist of robot  $k$  and its followers. If a conflict occurs, the leader negotiates on behalf of itself and its followers. Moreover, to achieve effective decentralized conflict resolution, manually designed local rules are adopted that determine which robot should be given priority. Giving priority to a robot means that it will move first, and a robot occupying the next node in higher priority robot's path must give way.

Hereinafter, the following concepts are used:

- *remainingNodes*: the local list of remaining nodes  $n_i^0, \dots, n_i^T$  in the planned path for robot  $i$ . The list is updated at each time step (a node is removed) and during conflict resolution (nodes are added if a robot needs to give way).
- *giveWayNode*: a free neighboring node that can be used by a robot to move out of the way and allow another, higher priority robot to pass.
- *numberRequestsMyNode*: the number of robots having their  $n_i^{t+1}$  or  $n_i^{t+2}$ ,  $\forall i \in \{1, \dots, m\}$ , equal to the robot's  $n_{id}^t$ .
- *numberFollowers*: the number of followers of the robot.

Upon starting the execution, all the robots are located in their initial nodes. In every time step, each robot  $i$  identifies all neighbors within communication range and sends them its local data, such as its next node  $n_i^{t+1}$ , *remainingNodes*, and *numberFollowers*. After receiving data from its neighbors, the robot checks for potential conflicts with its neighbors. Since conflict detection and handling is done online, only the robot's next node  $n_i^{t+1}$  is used for conflict detection. If a conflict is detected, the robots coordinate to solve the conflict as described in Algorithms 2 and 3 (details can be found in Sect. 3.1), then each robot calculates its action  $a_i$  and updates its *remainingNodes* accordingly. If no conflict is detected and if a robot has any followers, it checks if its immediate follower's path is longer than its own. If so, the robot gives way to its follower if it has a free neighboring node. This step is essential to avoid deadlocks in certain regions, such as narrow corridors.

In the subsequent step, the robot's action  $a_i$  and its updated *remainingNodes* list will be used in a post coordination process, see Algorithm 4. This process is executed by the robots involved in resolving conflicts in the previous steps to check for further potential conflicts resulting from their previous decisions. In

**Algorithm 1:** Proposed DCMAPF approach

---

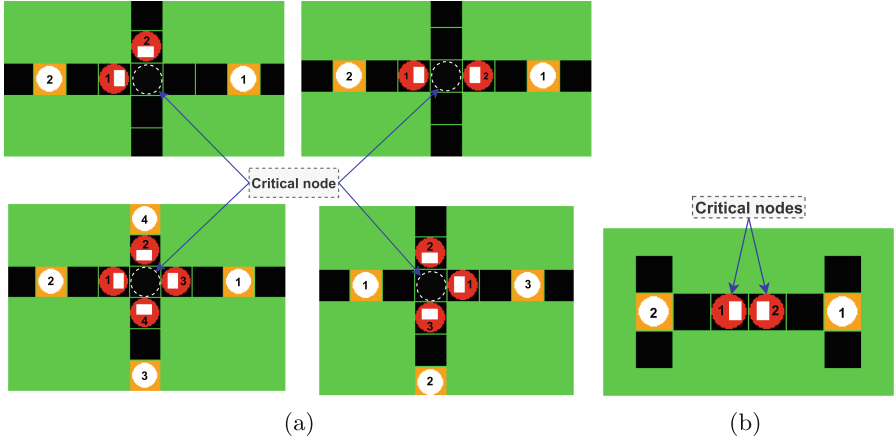
```

input: map,  $n_{myID}^0$ , Targets[]
phase 1: Path planning
remainingNodes  $\leftarrow A^*(\text{map}, n_{myID}^0, \text{Targets}[])$ 
phase 2: Execution and motion coordination
while remainingNodes  $\neq \phi$  or neighbors.remainingNodes  $\neq \phi$  do
   $n_{myID}^t \leftarrow \text{remainingNodes}[0]$ 
  pathLength  $\leftarrow \text{Length}(\text{remainingNodes})$ 
   $N_{myID}^t \leftarrow \text{GetNeighbors}()$ 
  send( $n_{myID}^{t+1}, \text{pathLength}, \text{numberFollowers}, \text{numberRequestsMyNode}$ )
  for  $i$  in  $N_{myID}^t$  do
    if ( $n_{myID}^{t+1} = n_i^t$ ) and ( $n_i^{t+1} = n_{myID}^t$ ) then
       $\text{criticalNode} \leftarrow \{n_{myID}^{t+1}, n_i^t\}$ 
       $a_{myID} \leftarrow \text{Algorithm3}(\text{criticalNode}, N_{myID}^t)$ 
    else if ( $n_{myID}^{t+1} = n_i^{t+1}$ ) then
       $\text{criticalNode} \leftarrow n_{myID}^{t+1}$ 
       $a_{myID} \leftarrow \text{Algorithm2}(\text{criticalNode}, N_{myID}^t)$ 
    else
      //no conflict detected
      nextAction  $\leftarrow \text{move}$ 
      follower  $\leftarrow \text{GetMyFollower}()$ 
      if (follower.pathLength  $>$  pathLength) then
         $\text{giveWayNode} \leftarrow \text{GetFreeNeighboringNode}()$ 
        if (giveWayNode is not None) then
          Insert the giveWayNode into remainingNodes
   $n_{myID}^{t+1} \leftarrow \text{remainingNodes}[0]$ 
  send( $n_{myID}^{t+1}, \text{plannedAction}$ )
   $a_{myID} \leftarrow \text{PostCoordination}(n_{myID}^{t+1}, a_{myID})$ 
  if ( $a_{myID} = \text{move}$ ) then
    move to  $n_{myID}^{t+1}$ 
    Remove  $n_{myID}^{t+1}$  from remainingNodes

```

---

this process, detected conflicts are resolved using the same steps and algorithms as described above. Afterward, the robots involved in the negotiation process send their calculated action  $a_i$  and next node  $n_i^{t+1}$  ( $\forall i \in N_i^t$ ) to their neighbors. Accordingly, leaders ensure that their followers adapt their actions to the outcome of the negotiation process. Once a robot  $i$  has calculated its  $a_i$  and updated its *remainingNodes*, the robot moves to  $n_i^{t+1}$  if  $a_i = \text{move}$ , or remains stationary in its current node  $n_i^t$  if  $a_i = \text{wait}$ . The steps presented in Algorithm 1 are reiterated until all robots have reached their target.



**Fig. 2.** Conflict illustrations and critical nodes. (a) Intersection conflicts, and (b) Opposite conflict.

### 3.1 Cooperative Conflict Resolution Strategy

In this work, we divide potential conflicts into the two types illustrated in Fig. 2: (i) *intersection conflict* and (ii) *opposite conflict (swapping conflict)*. The intersection conflict occurs when two or more robots have planned to pass through the same node in the same time step. In this type of conflict, there is only one critical node, which is the shared next node in the robots' paths. On the other hand, an opposite conflict occurs when two robots are located on two adjacent nodes and need to move in opposite directions. In this type of conflict, the robots' current nodes are the critical nodes.

The conflict resolution strategy has two steps. First, the robots negotiate to determine the highest priority robot (see below). In the second step, the robots calculate their actions to give way to the highest priority robot and to then pass through the critical node one by one.

**Priorities:** The procedure for defining the highest priority robot is based on six rules that prevent congestion and reduce the number of additional *giveWayNodes* necessary for the robots to pass through the critical node without collision. The following rules are applied in order and determine priority:

- rule1: a robot occupying a critical node is given priority.
- rule2: a robot moving out of another robot's way is given priority.
- rule3: the robot with the largest *numberFollowers* is given priority.
- rule4: a robot having a free neighboring node is given priority.
- rule5: the robot having the largest *numberRequestsMyNode* is given priority.
- rule6: the robot with the longest remaining path is given priority.

While the first three rules prevent deadlocks, the last three rules reduce the number of additional *giveWayNodes* introduced in the robots' path and thus enable the robots to avoid one another in fewer time steps.

### Conflict-Dependent Action Selection:

**Intersection conflict:** Algorithm 2 details the action selection process. Once the highest priority robot (*PriorityAgent*) has been determined, the node  $n_{PriorityAgent}^{t+2}$  is either free or occupied by another robot. In the first case, the robot with higher priority passes through the critical node first and the other robots have to wait in their current nodes for one time step. However, in the second case, the robot occupying the node  $n_{PriorityAgent}^{t+2}$  must give way to the robot with higher priority to pass and the other robots wait for one time step. The robot requested to move out of the way chooses a free neighboring node. If no free neighboring node is found, the robot chooses the node of another robot from its neighbors and informs the concerned neighbor to move out of the way, and so on.

**Opposite conflict:** The approach to solve an opposite conflict is shown in Algorithm 3. The robot with priority passes (i.e. its *action*  $\leftarrow$  *move*) and the other robot moves out of the way to a free neighboring node. If no free neighboring node is found, the robot with lower priority chooses the node of its follower robot (move backward) and informs the follower to move out of the way.

Note that any neighboring node calculated during the conflict resolution process will be inserted as the first elements in the *remainingNodes* list of the robot. Accordingly, if the robot’s action is *move*, then the robot selects the first node in its *remainingNodes*.

## 4 Experimental Results and Performances Analysis

In this section, we present the results of an extensive set of experiments conducted to assess the performance of *DCMAPF*. These tests were performed using benchmark maps with varying sizes, obstacles densities, and number of robots. We implemented *DCMAPF* in Python and the experiments were conducted on a workstation equipped with an AMD Ryzen 9 5950X 16-core CPU @3.40 GHz and 32 GB RAM.

### 4.1 Benchmarks and Setup

For our experiments, we chose three types of maps, *empty*, *random* and *warehouse* from the MAPF benchmark maps [20]. Specifically, we used the following maps: *empty-48-48*, *random-32-32-20*, *random-64-64-20*, and *warehouse-20-40-10-2-2*. For each combination of map and number of agents, we selected 25 scenarios from the MAPF benchmark.

We compared our *DCMAPF* approach to four state-of-the-art planners, namely: CBS with its improvement technique [8] as an optimal planner, EECBS [9] as a state-of-the-art bounded sub-optimal search-based planner, and PIBT and PIBT+ [13] as prioritized planners. Note that, for all planners, the implementations coded by their respective authors were used with default parameter settings [13]. The source code for these planners is available in [6].

---

**Algorithm 2:** Solve intersection conflict

---

**input** : criticalNode,  $N_i^t$   
**output**:  $a_{myID}$   
**step 1: Determine the highest priority agent**  
 $PriorityAgent \leftarrow \mathbf{CheckPriorityRules}()$   
Action  $\leftarrow$  Empty list  
**step 2: Calculate the action**  
Action[PriorityAgent]  $\leftarrow$  move  
**if** ( $n_{PriorityAgent}^{t+2}$  is Free) **then**  
    **for**  $i$  in  $N_i^t$  and  $i \neq PriorityAgent$  **do**  
        Action[ $i$ ]  $\leftarrow$  wait  
        **if** criticalNode is not Free **then**  
            giveWayNode  $\leftarrow$  **GetFreeNeighboringNode**()  
            Action[Agent occupying the criticalNode]  $\leftarrow$  move  
            Insert the giveWayNode into the remainingNodes of the agent  
    **else**  
        **for**  $i$  in  $N_i^t$  **do**  
            **if** ( $n_i^t = n_{PriorityAgent}^{t+2}$ ) **then**  
                Action[ $i$ ]  $\leftarrow$  move  
                giveWayNode  $\leftarrow$  **GetFreeNeighboringNode**()  
                Insert the criticalNode into the remainingNodes set of the agent  
            **else**  
                Action[ $i$ ]  $\leftarrow$  wait  
    **Return**(Actions[myID])

---



---

**Algorithm 3:** Solve opposite conflict

---

**input** : criticalNode,  $N_i^t$   
**output**:  $a_{myID}$   
**step 1: Determine the highest priority agent**  
 $PriorityAgent \leftarrow \mathbf{CheckPriorityRules}()$   
Action  $\leftarrow$  Empty list  
**step 2: Calculate the action**  
Action[PriorityAgent]  $\leftarrow$  move  
**for**  $i$  in  $N_i^t$  **do**  
    **if** ( $i \neq PriorityAgent$ ) **then**  
        Action[ $i$ ]  $\leftarrow$  move  
        giveWayNode  $\leftarrow$  **GetFreeNeighboringNode**()  
        Insert the giveWayNode into the remainingNodes of the agent  
**Return**(Actions[myID])

---

Our comparison metrics are *sum-of-costs* and *success rate*, which is the percentage of the MAPF instances solved within a runtime limit. It is important to note that *CBS*, *EECBS*, *PIBT* and *PIBT+* are centralized planners and have



---

**Algorithm 4:** Post coordination

---

```

input :  $n_{myID}^{t+1}, a_{myID}$ 
output:  $a_{myID}$ 

 $plannedAction \leftarrow a_{myID}$ 
step 1: Check for further potential conflicts
 $N \leftarrow \mathbf{GetNeighbors}()$ 
for  $i$  in  $N$  do
  if ( $n_{myID}^{t+1} = n_i^{t+1}$ ) and (the action of robot  $i$  is move) then
     $criticalNode \leftarrow n_{myID}^{t+1}$ 
     $a_{myID} \leftarrow \mathbf{Algorithm2}(criticalNode, N_{myID}^t)$ 
step 2: Followers adapt their actions to those of their leader
Leader= $\mathbf{GetAgentOccupyingNextNode}(N_{myID}^t)$ 
if ( $a_{leader} = \mathit{wait}$ ) then
   $a_{myID} \leftarrow \mathit{wait}$ 
else if ( $a_{leader} = \mathit{move}$ ) and ( $n_{leader}^{t+1} = n_{myID}^t$ ) then
   $a_{myID} \leftarrow \mathit{move}$ 
   $giveWayNode \leftarrow \mathbf{GetFreeNeighboringNode}()$ 
  Insert the  $giveWayNode$  into the  $remainingNodes$  of the agent
else
   $a_{myID} \leftarrow plannedAction$ 
Return( $a_{myID}$ )

```

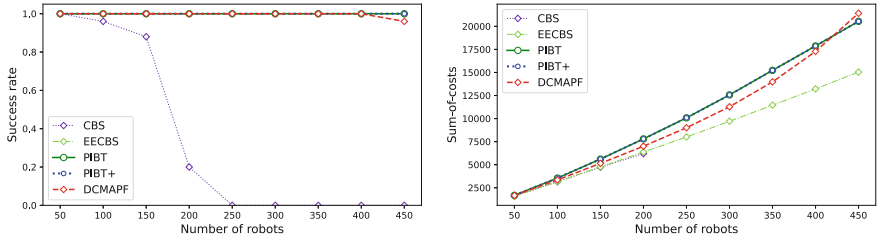
---

access to the whole state of the system, whereas *DCMAPF* is a decentralized approach where the robots' decisions are based only on their local observation and messages shared between robots within a limited communication range. Since *DCMAPF* resolves conflicts online, we allowed a maximum of 300 time steps for 32 and 48-sized maps, and 600 time steps for the other maps. The other offline planners were given a time limit of 30s to plan the paths for all robots as is commonly used [13,20]. An execution was considered unsuccessful if the robots failed to resolve a conflict or a planner failed to provide a solution within the time limit.

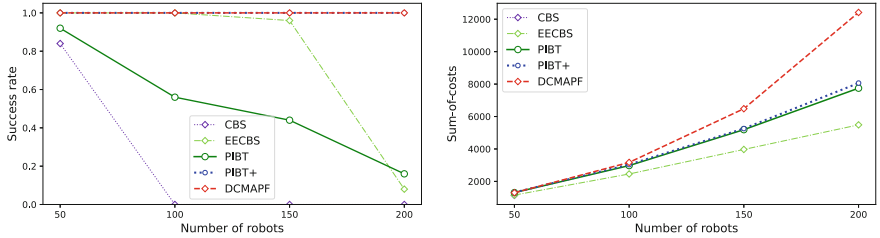
## 4.2 Results

The obtained results are presented in Fig. 3. The first clear trend is that the *DCMAPF* performs well in terms of success rate in all maps no matter the map size or the number of robots. Secondly, a prominent trend observed in all plots of the metric *sum-of-costs* is that *DCMAPF* outperforms the prioritized planners *PIBT* and *PIBT+* for small fleet sizes. Additionally, it is evident that the *sum-of-costs* of *DCMAPF* tends to increase relative to the other planners as the maps become more challenging with higher numbers of robots.

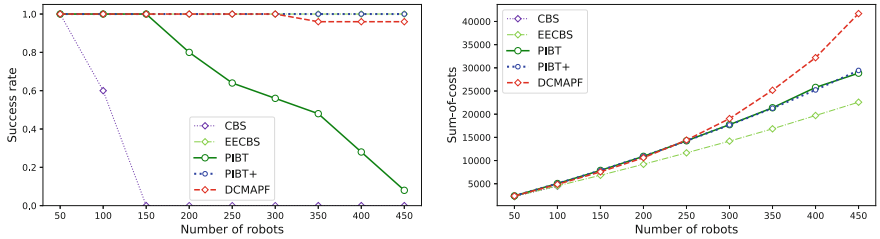
In maps with low obstacle densities, such as *empty-48-48*, all planners have very high success rates, except CBS that has lower success rate in most experiments involving more than 100 robots due to its computational complexity. In terms of solution cost, *DCMAPF* outperforms *PIBT* and *PIBT+* when the



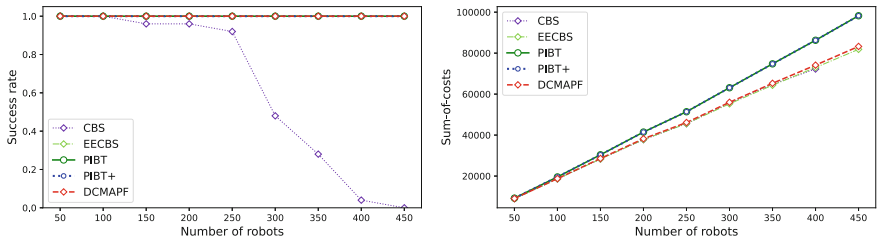
(a) *empty-48-48*



(b) *random-32-32-20*



(c) *random-64-64-20*



(d) *warehouse-20-40-10-2-2*

**Fig. 3.** Comparative results in terms of *success rate* and *sum-of-costs* of successful runs on four benchmark maps.

number of robots is less than 450 since in conflict resolution, robots with low priority have enough space to quickly give way to higher priority robots. However, when the robot count hits 450, we notice that the performance of *DCMAPF* is slightly worse than *PIBT* and *PIBT+*. On the random-32-32-20 map and on the random-64-64-20 map, a similar, but more pronounced trend is observed. This is due to the decentralized nature of *DCMAPF* where each robot relies only on a partial observation of the environment. This can make it difficult to resolve conflicts that involve large numbers of robots and can result in robots getting stuck in undesirable looping behavior. The looping behavior could potentially be corrected by introducing a mechanism to detect and avoid this undesirable behavior. Importantly, the success rate on the random-32-32-20 map reached 100% and is high for both the empty-48-48 map and the random-64-64-20 map. Interesting results can be observed for the warehouse map, where *DCMAPF* shows high performance and outperforms *PIBT* and *PIBT+*, and yields similar results to those of the sub-optimal planner *EECBS* and the optimal planner *CBS*, further substantiating the efficacy of *DCMAPF*.

In summary, the performance of *DCMAPF* compares well to that of centralized planners, except for scenarios with very high robot densities. Its solution quality is better than that of the prioritized planners *PIBT* and *PIBT+* with a high success rate in multiple scenarios. The one exception is the particularly constrained scenario of the small map *random-32-32-20*, where *DCMAPF*'s performance is worse than the other planners, specifically when the robot count exceeds 150. Notwithstanding the increase in sum-of-costs in this map, the 100% success rates demonstrate the robustness of *DCMAPF* in demanding circumstances. In a nutshell, the obtained results highlight the effectiveness of *DCMAPF* and that decentralized coordination is a promising approach to solve *MAPF* problems. Example runs can be found in the supplementary video: [https://youtu.be/5\\_5TdVuM8kI](https://youtu.be/5_5TdVuM8kI).

## 5 Conclusions

In this work, we presented a decentralized multi-agent path finding approach for mobile robots with a limited communication range. In the proposed approach, each robot plans its shortest path offline and then autonomously coordinates with its neighbors to solve potential conflicts as they occur during task execution. Through an extensive set of experiments, we showed that the *DCMAPF* produces competitive results compared to state-of-the-art centralized planners, and therefore can be considered a promising decentralized approach to solve *MAPF* problems. Future work will focus on implementing a strategy to avoid robots getting stuck in undesirable looping behavior in highly constrained maps.

**Acknowledgements.** This work was supported by the Independent Research Fund Denmark under grant 0136-00251B.

## References

1. Bobanac, V., Bogdan, S.: Routing and scheduling in multi-AGV systems based on dynamic banker algorithm. In: Proceedings of the 16th Mediterranean Conference on Control and Automation, pp. 1168–1173. IEEE (2008)
2. Damani, M., Luo, Z., Wenzel, E., Sartoretto, G.: PRIMAL<sub>2</sub>: pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robot. Autom. Lett.* **6**(2), 2666–2673 (2021)
3. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.* **31**, 591–656 (2008)
4. Hönig, W., et al.: Multi-agent path finding with kinematic constraints. In: Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS), pp. 477–485. AAAI Press (2016)
5. Lam, E., Le Bodic, P.: New valid inequalities in branch-and-cut-and-price for multi-agent path finding. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pp. 184–192. AAAI Press (2020)
6. Li, J.: Source code for CBS, EECBS and PIBT. <https://github.com/Jiaoyang-Li/CBSH2-RTC>. <https://github.com/Jiaoyang-Li/EECBS> and <https://github.com/Kei18/pibt2>
7. Li, J., Chen, Z., Harabor, D., Stuckey, P., Koenig, S.: Anytime multi-agent path finding via large neighborhood search. In: International Joint Conference on Artificial Intelligence, pp. 4127–4135. IJCAI (2021)
8. Li, J., Harabor, D., Stuckey, P.J., Ma, H., Gange, G., Koenig, S.: Pairwise symmetry reasoning for multi-agent path finding search. *Artif. Intell.* **301**, 103574 (2021)
9. Li, J., Ruml, W., Koenig, S.: EECBS: a bounded-suboptimal search for multi-agent path finding. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 12353–12362. AAAI Press (2021)
10. Ma, H., Harabor, D., Stuckey, P.J., Li, J., Koenig, S.: Searching with consistent prioritization for multi-agent path finding. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 7643–7650. AAAI Press (2019)
11. Ma, H., Li, J., Kumar, T., Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 837–845. IFAAMAS (2017)
12. Okoso, A., Otaki, K., Nishi, T.: Multi-agent path finding with priority for cooperative automated valet parking. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 2135–2140. IEEE (2019)
13. Okumura, K., Machida, M., Défago, X., Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-2019), pp. 535–542. IJCAI Organization (2019)
14. Rathi, A., Vadali, M., et al.: Dynamic prioritization for conflict-free path planning of multi-robot systems. *arXiv preprint [arXiv:2101.01978](https://arxiv.org/abs/2101.01978)* (2021)
15. Reijnen, R., Zhang, Y., Nuijten, W., Senaras, C., Goldak-Altgassen, M.: Combining deep reinforcement learning with search heuristics for solving multi-agent path finding in segment-based layouts. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2647–2654. IEEE (2020)
16. Ryan, M.R.K.: Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res.* **31**, 497–542 (2008)

17. Sajid, Q., Luna, R., Bekris, K.: Multi-agent pathfinding with simultaneous execution of single-agent primitives. In: International Symposium on Combinatorial Search, vol. 3, no. 1, pp. 88–96. AAAI Press (2012)
18. Sartoretti, G., et al.: Primal: pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robot. Autom. Lett.* **4**(3), 2378–2385 (2019)
19. Stephan, J., Fink, J., Kumar, V., Ribeiro, A.: Concurrent control of mobility and communication in multirobot systems. *IEEE Trans. Rob.* **33**(5), 1248–1254 (2017)
20. Stern, R., et al.: Multi-agent pathfinding: definitions, variants, and benchmarks. In: Symposium on Combinatorial Search (SoCS), pp. 151–158. AAAI Press (2019)
21. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: 2009 IEEE International Conference on Robotics and Automation, pp. 3613–3619. IEEE (2009)
22. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: Proceedings of the Twenty-second European Conference on Artificial Intelligence, ECAI, pp. 810–818. IOS Press (2016)
23. Van Den Berg, J.P., Overmars, M.H.: Prioritized motion planning for multiple robots. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 430–435. IEEE (2005)
24. Yu, D., Hu, X., Liang, K., Ying, J.: A parallel algorithm for multi-AGV systems. *J. Ambient. Intell. Humaniz. Comput.* **13**(4), 2309–2323 (2022)
25. Yu, J., LaValle, S.M.: Structure and intractability of optimal multi-robot path planning on graphs. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, pp. 1443–1449. AAAI Press (2013)
26. Zhang, Z., Guo, Q., Yuan, P.: Conflict-free route planning of automated guided vehicles based on conflict classification. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1459–1464. IEEE (2017)
27. Zhao, Y., Liu, X., Wang, G., Wu, S., Han, S.: Dynamic resource reservation based collision and deadlock prevention for multi-AGV. *IEEE Access* **8**, 82120–82130 (2020)