Swarm Intelligence (2024) 18:167–185
https://doi.org/10.1007/s11721-023-00230-7

Improved decentralized cooperative multi-agent path finding for robots with limited communication

Abderraouf Maoudj[1] · **Anders Lyhne Christensen**[2]

Received: 29 November 2022 / Accepted: 16 October 2023 / Published online: 12 December 2023

Abstract
Multi-agent path finding (MAPF) holds significant practical relevance in numerous real-world applications involving fleets of mobile robots. The efficiency of such systems is directly determined by the quality of the paths calculated. Accordingly, extensive effort has been directed toward creating effective algorithms to address the MAPF problem. Yet, many existing MAPF algorithms still depend on offline centralized planning, paired with often unrealistic assumptions—such as robots having complete observability of the environment and moving in a deterministic fashion. The resultant plans are typically unsuitable for direct implementation on real robots where these assumptions do not usually apply. Aiming for more effective robot coordination under realistic conditions, we introduce an enhanced decentralized method. In this method, each robot coordinates solely with neighbors within a limited communication radius. Each robot attempts to follow the shortest path from its starting point to its designated target, addressing conflicts with other robots as they occur. Our method also incorporates path replanning, local motion coordination, and mechanisms to avoid robots becoming trapped in livelocks or deadlocks. Simulation-based results from various benchmark scenarios confirm that our enhanced decentralized method is both effective and scalable, accommodating up to at least 1000 robots.

Keywords Multi-agent path finding · Partial observability · Decentralized coordination · Livelock · Deadlock

1 Introduction

Nowadays, fleets of mobile robots are increasingly being adopted in industrial environments, such as logistic distribution systems and automated warehouses, to meet the growing transportation and cargo sorting needs, e.g., transporting goods and materials

✉ Abderraouf Maoudj
abderraouf.maoudj@kfupm.edu.sa

Anders Lyhne Christensen
andc@mmmi.sdu.dk

[1] Interdisciplinary Research Center for Intelligent Manufacturing and Robotics (IRC-IMR), King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia

[2] SDU UAS Center, MMMI, University of Southern Denmark (SDU), Odense, Denmark

between workstations and storage pipes. The increased use of robot fleets has given rise to a number of challenging optimization problems, such as multi-agent path finding (MAPF) (Dingding et al., 2022) and multi-robot routing and scheduling (Bobanac & Bogdan, 2008).

The MAPF problem is of significant practical relevance in several real-world applications of multi-robot systems and receives a great deal of attention as the setup of industrial warehouses matches some of the assumptions commonly used in the MAPF literature, e.g., a large fleet of robots must perform transportation tasks in a known environment represented as a regular square lattice (Varambally et al., 2022). The MAPF problem consists of computing collision-free paths for fleets of agents from their current locations to their respective targets.

MAPF remains a major challenge, and it has been widely studied in the literature (Reijnen et al., 2020; Stern et al., 2019). Numerous algorithms have been developed to achieve efficient coordination of fleets of robots in shared environments, such as CBS (Sharon et al., 2015; Li et al., 2021c), Lazy-CBS (Gange et al., 2019), Multi-label A* (Grenouilleau et al., 2019), M* (Wagner & Choset, 2011), and prioritized planning (Ma et al., 2019). However, most of the proposed algorithms are offline centralized planners, and they cannot be directly applied in the real world as they rely on unrealistic assumptions (Varambally et al., 2022). In real-world scenarios, the number of mobile robots can reach hundreds (Lian et al., 2020; Zhao et al., 2020) and they do not always move at constant speeds. The biggest drawback of centralized approaches is the lack of scalability, as computational complexity tends to grow exponentially with the robot count and map dimensions (Draganjac et al., 2020). Scenarios involving large numbers of robots thus rule out simple adaptation of the offline compute-intensive algorithms, and scalable online approaches are appealing for such settings (Okumura et al., 2022).

In our previous work, we proposed the *decentralized cooperative multi-agent path-finding* (DCMAPF) approach (Maoudj & Christensen, 2022) to solve the MAPF problem. In DCMAPF, each agent autonomously plans its path using A* while initially ignoring the other agents and then, resolves conflicts as they occur. However, DCMAPF proved to have two shortcomings in highly constrained maps: (i) robots could get stuck in *livelock* situations where a pattern of movement is repeated indefinitely, and (ii) in rare cases, robots could end up in *deadlocks* situations where no solution could be found. In this paper, we extend our previous work on DCMAPF and propose *improved decentralized cooperative multi-agent path-finding* (IDCMAPF), where we introduce mechanisms to deal with DCMAPF's shortcomings, which improve robot coordination and enable large-scale fleets of autonomous mobile robots to operate effectively and reliably in highly constrained environments. These mechanisms facilitate the coordination of robots with a high degree of robustness to uncertainties and variations in the speeds of the robots. Moreover, IDCMAPF does not require the robots to have complete information about each other. Instead, we consider that robots operate in a partially observable world, where each robot can only communicate with neighbors within its vicinity. To use IDCMAPF on physical robots, the robots would thus only need a map of the environment, the ability to localize themselves within that environment [e.g., via SLAM (Beinschob & Reinke, 2015), radio (Kammel et al., 2022), or fiducial markers (Varambally et al., 2022)], and the ability to communicate locally with other robots.

The subsequent sections of this paper are structured as follows: Sect. 2 formalizes the MAPF problem. Section 3 discusses state-of-the-art methods for solving MAPF problems. Section 4 presents the proposed decentralized approach. Section 5 reports on the experiments we conducted to evaluate IDCMAPF against state-of-the-art planners and its

robustness to variations in the speeds of the robots in a fleet. Section 7 provides conclusions and lays out directions for future work.

2 Problem formulation

In many practical applications, such as automated warehouses, the layout of industrial environments is fixed, and robots can only move along a predefined roadmap (Dingding et al., 2022). Figure 1 illustrates an example of an automated warehouse, modeled as a 35×15 grid map, in which a set of m mobile robots $\{r_1, ..., r_m\}$ must perform their assigned pickup and delivery tasks. In real-world scenarios, wireless communication can be noisy, in particular over long distances, and all robots may not always be able to communicate with one another (Damani et al., 2021; Stephan et al., 2017). We therefore assume that each robot has a limited communication range and can only access the state of its neighboring robots within a range of two squares, as illustrated in Fig. 2.

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. The robots are assumed to know the graph and their own position and target location in the graph. At every time step t, each

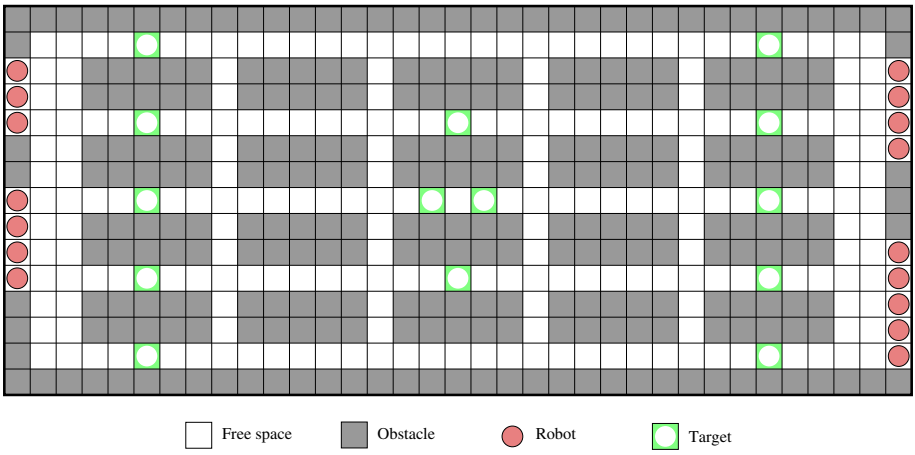


Fig. 1 Example of a warehouse layout modeled as a 35×15 grid map

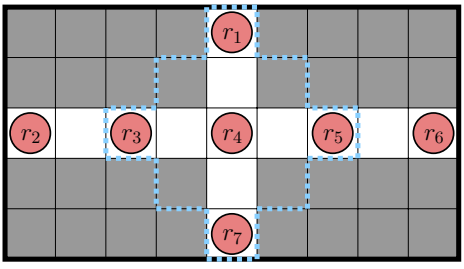


Fig. 2 Communication range of a robot (r_4) in the environment

robot r_i occupies one of the graph nodes n_i^t, referred to as the location of that robot at time-step t and can choose to perform an *action* a_i from one time-step to the next one. The action can be either `wait` in its current node or `move` to an adjacent free node in one of the cardinal directions. At each time step, if robot r_j is in communication range of robot r_i, we say that robot r_j is in robot r_i's neighborhood $j \in N_i^t$.

A *free node* is a location on the map that is neither an obstacle nor occupied by a robot. Robots can traverse free nodes without colliding with other robots or obstacles. A *conflict* is defined as a situation where two or more robots attempt to move to the same node (*intersection conflict*) or to traverse the same edge (*opposite conflict*) in the same discrete time step. A conflict between two agents r_i and r_j thus occurs if and only if:

$$\begin{aligned} &n_i^{t+1} = n_j^{t+1} && \text{(intersection conflict)}, \textit{or} \\ &n_i^{t+1} = n_j^t \wedge n_j^{t+1} = n_i^t, && \text{(opposite conflict)}. \end{aligned}$$

The MAPF problem is concerned with planning a path for each robot from its starting point to its target location in a given environment. The aim is to ensure that no two robots occupy the same node or cross the same edge in the same time step, as this would result in a collision. The objective considered in this work is to minimize the *sum-of-costs*, that is, the sum over all agents of the time steps required to reach their target locations (Surynek et al., 2016).

3 Related work

MAPF is NP-hard to solve optimally (Yu & LaValle, 2013). Numerous approaches to solving this problem have been proposed, and the state-of-the-art algorithms can be categorized into four main groups (Li et al., 2021a):

Systematic search algorithms, which are centralized planning approaches, can identify all possible solutions, including an optimal one. In this category, several algorithms have been proposed, including the branch-and-cut-and-price (BCP) algorithm (Lam & Le Bodic, 2020), pairwise symmetry reasoning (Li et al., 2021b), conflict-based search (CBS) algorithms (Sharon et al., 2015) and their variants (Li et al., 2021b, c) — 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 exponential computational complexity as the problem size grows.

Rule-based algorithms, in which the agents follow ad hoc rules as they move toward their respective targets step-by-step (Okumura et al., 2022). Examples include the graph abstraction approach (Ryan, 2008), the conflict classification-based algorithm (Zhang et al., 2017), biconnected graphs (Surynek, 2009), and parallel-push-and-swap (PPS) (Sajid et al., 2012). 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 (Reijnen et al., 2020). Different learning-based algorithms have been proposed in literature to solve the MAPF problem (Sartoretti et al., 2019; Damani et al., 2021). 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 (Okumura et al., 2022; Sartoretti et al., 2019).

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 (Rathi & Vadali, 2021), searching with consistent prioritization (Ma et al., 2019), the hierarchical cooperative A* approach (HCA), and priority inheritance with backtracking (Okumura et al., 2022). 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 (Van Den Berg & Overmars, 2005).

The algorithms described above mainly focus on offline centralized planning, which may not be viable for all automation systems (Skrynnik et al., 2022; Maoudj et al., 2023), and are based on simplistic assumptions: (i) most of them ignore the robots' kinematic constraints, (ii) assume that robots always move at equal and constant speeds, (iii) assume that robots have full observability of the environment, often requiring expensive external sensors to track the state of all the robots of the fleet in real-world applications and this may increase the cost of implementation and maintenance, and (iv) do not take into account imperfect plan-execution issues (Hönig et al., 2016). In real-world scenarios, a robot may, for instance, 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 variations in the robots' speeds can thus significantly affect the applicability of these offline approaches. Additionally, in real-world applications, such as transporting packages in an automated warehouse, whenever an agent reaches a target, it often receives a new one, and a new path should then be planned in real-time (Okumura et al., 2022). In many practical scenarios where the environment and robot behavior are not completely deterministic, online approaches become desirable.

In this paper, we propose IDCMAPF, an improved decentralized cooperative multi-agent approach to cope with the aforementioned issues. IDCMAPF extends our previous decentralized approach, DCMAPF (Maoudj & Christensen, 2022). In the new version presented in this paper, robots are given the capacity to detect potential deadlocks and livelocks, allowing them to replan their paths on-the-fly to circumvent these issues. Since conflicts are resolved as they occur, delaying robots during plan execution does not affect the applicability of IDCMAPF, and it can thus coordinate large fleets of mobile robots with a high degree of robustness to uncertainties and variations in the robots' speeds. Moreover, IDCMAPF is a deliberative planner suitable for scenarios where agents have a sequence of targets. This feature is especially promising for practical situations where robots are continuously assigned new tasks and must plan paths online (Damani et al., 2021).

4 The proposed approach: IDCMAPF

In this section, we introduce IDCMAPF, our improved decentralized cooperative multi-agent path-finding approach. This approach integrates path planning and motion coordination capabilities with mechanisms to detect and circumvent livelocks and deadlocks. The motion coordination strategy is built upon a set of priority rules, facilitating effective conflict resolution. Moreover, local negotiation within the limited communication range ensures scalability to large fleets of robots. IDCMAPF, as detailed in Algorithm 1, operates using the following variables, each maintained locally by every robot:

– *pathHistory*: is the history of the nodes that the robot has passed through. In this data set, we store the robot's visited nodes at each time step.

- *remainingNodes*: the local list of remaining nodes in the planned path $n_i^0, ..., n_i^T$ 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}, $\forall i \in \{1, ..., m\}$, equal to the robot's n_{id}^t.
- *numberFollowers*: the number of followers of the robot.

In IDCMAPF, we introduce a *leader–follower* concept for adjacent robots moving in the same direction to reduce the complexity of local coordination. Robot r_k is a follower of robot r_i at time step t if $n_k^{t+1} = n_i^t$. With the purpose of achieving effective coordination between robots in conflict, the leader negotiates on behalf of itself and its followers. A *leader* can have an arbitrary number of followers, where the *followers* of robot r_i consist of its immediate follower r_k and its followers. Communication between a *leader* and its *followers* outside of its communication range is facilitated through message passing among interconnected followers.

Algorithm 1 The proposed IDCMAPF approach

input : *map*, *pathHistory*, n_{myID}^0, *target*, *numWaitingSteps* = 0
`phase 1: Path planning`
remainingNodes ← A^*(*map*, n_{myID}^0, *target*)
`phase 2: Execution and motion coordination`
while *robot is active* **do**
  numRepeatedNodes ← **FindRepeatedNodes**(*pathHistory*)
  numRepeatitions ← **ComputeNumRepeatitions**()
  if *(numRepeatedNodes > nodesThreshold)* **and** *(numRepeatitions > repetitionThreshold)* **then**
    remainingNodes ←replan path while considering the last visited node n_{myID}^{t-1} as a virtual obstacle, thereby reversing movement direction
  if *(numWaitingSteps > waitingThreshold)* **then**
    remainingNodes ←replan path while adding a virtual obstacle in each occupied neighboring node
  n_{myID}^t ← *remainingNodes[0]*
  pathLength ← **Length**(*remainingNodes*)
  N_{myID}^t ← **GetNeighbors**()
  send(n_{myID}^{t+1},*pathLength*,*numberFollowers*,*numberRequestsMyNode*)
  for i in N_{myID}^t **do**
    if $(n_{myID}^{t+1} = n_i^{t+1})$ **then**
      criticalNode ← n_{myID}^{t+1}
      a_{myID} ← **SolveIntersectionConflict**(*criticalNode*, N_{myID}^t)
    else if $(n_{myID}^{t+1} = n_i^t)$ **and** $(n_i^{t+1} = n_{myID}^t)$ **then**
      a_{myID} ← **SolveOppositeConflict**(N_{myID}^t)
    else
      //no conflict detected
      a_{myID} ← `move`
      follower← **GetMyFollower**()
      if *(follower is not None)* **and** *(follower.pathLength > pathLength)* **then**
        giveWayNode←**FindFreeNeighboringNode()**
        if *(giveWayNode is not None)* **then**
          Insert the *giveWayNode* into *remainingNodes*
  n_{myID}^{t+1} ← *remainingNodes[0]*
  send(n_{myID}^{t+1}, *plannedAction*)
  a_{myID} ← **PostCoordination**(n_{myID}^{t+1},a_{myID})
  if $(a_{myID}$ = `move`*)* **then**
    `move to` n_{myID}^{t+1}
    Store the node n_{myID}^{t+1} in *pathHistory*
    Remove n_{myID}^{t+1} from *remainingNodes*
    numWaitingSteps = 0
  else
    numWaitingSteps += 1

To avoid deadlocks, each robot tracks the consecutive time steps during which it encounters a conflict, and its resulting action is `wait`. If the number of consecutive `waits` exceeds a configurable threshold (*waitingThreshold*), the robot re-plans its path while adding a virtual obstacle in each occupied neighboring node. If all of a robot's neighboring nodes are occupied and no alternative path is identified (as shown in Fig. 3), the robot queries nearby robots for free nodes and subsequently re-plans its path using a randomly selected free node.

To address livelocks, we adopt the concept of *pathHistory*, wherein the robot's location (node) at each time step is stored in memory. A robot can then detect livelock situations by checking for repeated nodes in its *pathHistory*. Should a robot r_i repeatedly visits the same set of nodes, it re-plans an alternative path treating the last visited node, n_i^{t-1}, as a virtual obstacle.

IDCMAPF operates in two phases: (i) *Path planning* and (ii) *Execution and motion coordination*. In the first phase, each robot independently plans the shortest path from its initial position to its assigned target location using A* and without taking into account other robots or their paths. In the second phase, robots follow their planned paths while resolving local conflicts as they arise. Resolving conflicts between robots as they occur makes IDCMAPF robust to uncertainty in the speeds of robots during the plan execution.

At the beginning, all the robots are located in their initial nodes (n_i^0, $\forall i \in \{1, ..., m\}$). Each robot knows the graph representing the warehouse layout and its target location in the graph. Upon starting the execution, each robot plans its shortest path and then, starts executing its plan. In every time step, each robot first checks its last visited nodes stored in its *pathHistory* and re-plans its path if it visits the same set of *nodesThreshold* nodes more than *repetitionThreshold* times. Second, the robot ensures that it does not stay trapped in a deadlock by re-planning its path if the number of consecutive waits exceeds the *waitingThreshold*.

After that, each robot r_i identifies its neighbors and sends them its local data necessary to identify and resolve potential conflicts, such as its next node n_i^{t+1}, *remainingNodes*, and *numberFollowers*. Upon receiving data from its neighbors, the robot checks for potential conflicts. Conflict detection and handling is thus done exclusively online, and 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. 4.1). Each robot then calculates its action a_i and updates its *remainingNodes* and *pathHistory* accordingly.

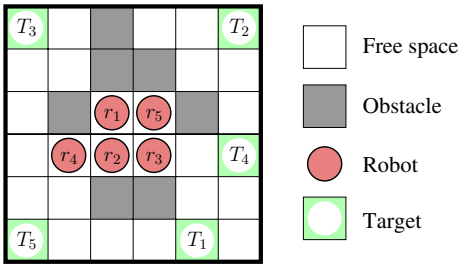


Fig. 3 Example of a situation where the robots are at risk of getting trapped in a deadlock situation. Robots r_3 and r_4 can re-plan their respective paths immediately, because they both have adjacent free nodes. However, since robots r_1, r_2, and r_5 do not have any free neighboring node, they must ask their neighboring robots if they have any free neighboring node, and then, re-plan their respective path through a randomly picked free node received

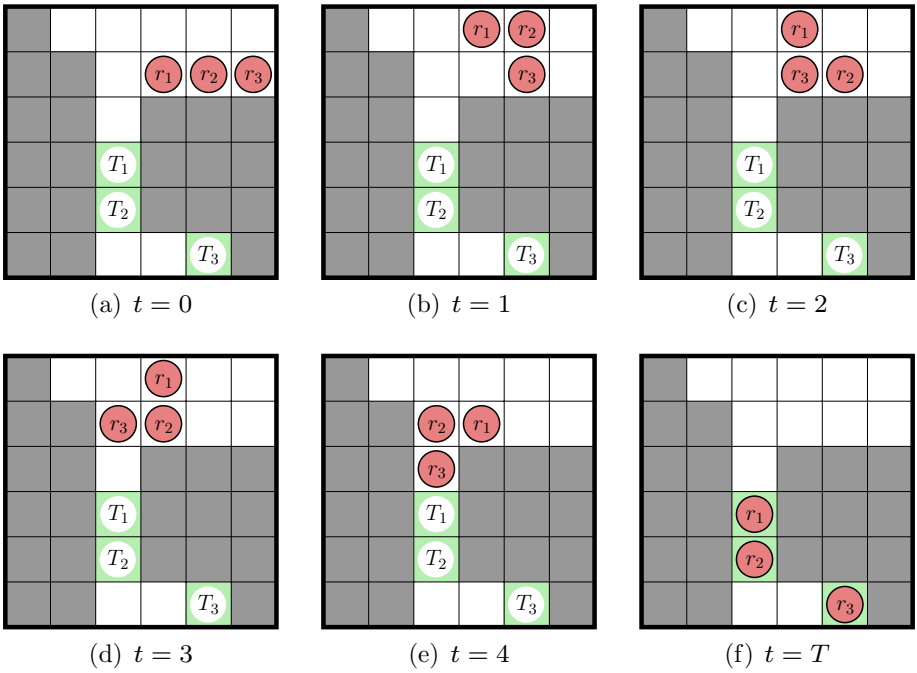


Fig. 4 Example of how deadlocks are prevented in narrow corridors. **a** Initial configuration, **b** robots r_1 and r_2 give way to their immediate follower, **c**–**e** the robots' position after second, third and fourth time step, respectively, and **f** the last step when robots reach their targets

If no conflict is detected and if a robot has 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. The objective of this behavior is to prevent deadlocks in certain regions, such as narrow corridors, see example in Fig. 4. In the example, robots r_1 and r_2 have free neighboring nodes and must give way to their immediate followers (robots r_2 and r_3, respectively), otherwise the robots will enter a deadlock when r_1 reaches its target.

In the subsequent step, the robot's action a_i and its updated *remainingNodes* list will be used in a solution validation process, see Algorithm 2. 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 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 r_i has calculated its a_i and updated its *remainingNodes*, the robot moves to n_i^{t+1} if $a_i =$ `move` and stores n_i^{t+1} in its *pathHistory*, or remains stationary in its current node n_i^t if $a_i =$`wait`. The steps presented in Algorithm 1 are reiterated until each robot has reached its target.

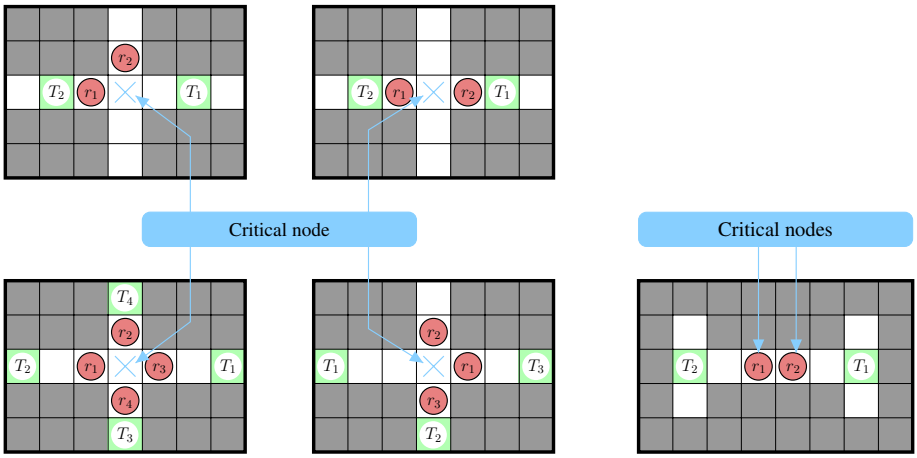
Algorithm 2 PostCoordination

input : n^{t+1}_{myID},a_{myID}
output: a_{myID}

plannedAction ← a_{myID}
`step 1`: Check for further potential conflicts
N ← **GetNeighbors**()
for i in N **do**
  if ($n^{t+1}_{myID} = n^{t+1}_{i}$) **and** *(the action of robot i is* `move`*)* **then**
    criticalNode← n^{t+1}_{myID}
    a_{myID} ←**SolveIntersectionConflict**(*criticalNode*, N^{t}_{myID})
`step 2`: If the robot is a follower, adapt its action to that of its leader
leader=**GetAgentOccupyingNextNode**(N^{t}_{myID})
if (a_{leader} = `wait`) **then**
  a_{myID} ←`wait`
else if (a_{leader} = `move`) **and** ($n^{t+1}_{leader} = n^{t}_{myID}$) **then**
  a_{myID} ←`move`
  giveWayNode← **FindFreeNeighboringNode()**
  Insert the *giveWayNode* into the *remainingNodes* of the agent
else
  a_{myID} ← *plannedAction*
Return(a_{myID})

4.1 Conflict detection and cooperative conflict handling

According to the movement direction of robots during plan execution, conflicts can be divided into two types as illustrated in Fig. 5: (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 at the same time step, see Fig. 5a. 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 moving in opposite



(a) Intersection conflicts (b) Opposite conflict

Fig. 5 Conflict illustrations and critical nodes. **a** Intersection conflicts: two or more robots attempt to move to the same node at the same discrete time step. In this type of conflict, there is only one critical node, which is the shared next node. **b** Opposite conflict: two robots attempt to traverse the same edge at the same discrete time step. In this type of conflict, the robots' current nodes are the critical nodes

directions are located at two adjacent nodes, see Fig. 5b. In this type of conflict, the robots' current nodes are the critical nodes.

The conflict resolution strategy for both types of conflicts consists of two steps. First, the robots negotiate to determine the highest priority robot (see below) that will pass first. In the second step, the other robots calculate their actions to give way to the highest priority robot and to then pass through the critical node one-by-one. In the following, we present the local rules that determine robot priorities and therefore which robot should pass and which robot(s) should give way.

4.1.1 Priority rules

To effectively coordinate the robots' movements when conflicts occur, priority rules are designed to guide the robots' decisions by determining the highest priority robot in resolving conflicts. The procedure for determining the highest priority robot is based on seven priority rules that prevent congestion and allow to reduce the number of additional *giveWayNodes* necessary for the robots to pass through the critical node one-by-one without collision. Giving priority to a robot i means that it will move first, and any robot occupying its next node n_i^{t+1} must give way. The following rules are applied, in order to 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 in conflict with another robot having a free adjacent node is given priority.
- `rule4`: the robot with the largest *numberFollowers* is given priority.
- `rule5`: a robot having its node n_i^{t+2} free is given priority.
- `rule6`: the robot having the largest *numberRequestsMyNode* is given priority.
- `rule7`: the robot with the longest remaining path is given priority.

While the first three rules prevent collisions and ensure progress, the last four rules reduce the number of additional *giveWayNodes* introduced in the robots' path during conflict resolution. In an intersection conflict, all priority rules are applied, in an opposite conflict, only `rule2`, `rule3`, `rule4`, `rule6`, and `rule7` are applied.

4.1.2 Conflict-dependent action selection

Intersection conflict: Algorithm 3 details the action selection process to solve intersection conflicts. Once the highest priority robot (*priorityRobot*) has been determined, $n^{t+2}_{priorityrobot}$ 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 (as illustrated in Fig. 6, top). However, in the second case, the robot occupying the node $n^{t+2}_{priorityRobot}$ must give way to the robot with higher priority to pass and the other robots wait for one time step (Fig. 6,

Algorithm 3 SolveIntersectionConflict

```
input : criticalNode, N_i^t
output: myAction
step 1: Determine the highest priority robot
priorityRobot←CheckPriorityRules()
step 2: Calculate the action
if (myID = priorityRobot) then
    myAction ←move
else
    if (n_myID^t = n_priorityRobot^{t+2}) or I am occupying the criticalNode then
        giveWayNode← FindFreeNeighboringNode()
        myAction ← move
        Insert the giveWayNode into my local remainingNodes
    else
        myAction←wait
Return(myAction)
```

bottom). 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. As shown in Fig. 6(top), robot r_1 has the longest remaining path, so it will have priority. Since n_1^{t+2} is free, robot r_1 will pass first, and robot r_2 must wait in its current node. Whereas in Fig. 6(bottom), the highest priority robot is robot r_2 and its n_2^{t+2} is occupied by robot r_1. As a result, robot r_1 will move to a free adjacent node and robot r_2 will pass. It should be noted that any *giveWayNode* 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*.

Opposite conflict: The approach to solve an opposite conflict is shown in Algorithm 4. The robot with priority passes (i.e., its *action* ← `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.

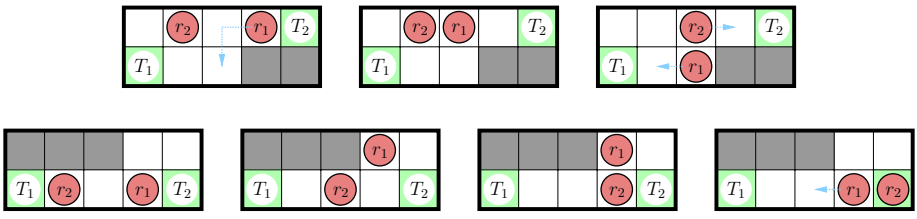


Fig. 6 Intersection conflict examples. *Top row*: node n_1^{t+2} of the highest priority robot (robot r_1) is free, it passes and robot r_2 must wait. *Bottom row*: node n_2^{t+2} of the highest priority robot (robot r_2) is not free, which requires robot r_1 to give way (move to a free neighboring node and then robot r_2 passes)

Algorithm 4 SolveOppositeConflict

```
input  : N_i^t
output: myAction
step 1: Determine the highest priority robot
priorityRobot←CheckPriorityRules()
step 2: Calculate the action
if (myID = priorityRobot) then
    myAction ←move
else
    myAction ←move
    giveWayNode← FindFreeNeighboringNode()
    Insert the giveWayNode into my local remainingNodes
Return(myAction)
```

5 Simulation-based results and performance analysis

This section is devoted to the empirical evaluation of IDCMAPF. We first perform a set of preliminary experiments to tune the parameters of IDCMAPF. We then perform two sets of performance experiments. In the first set of experiments, we compare IDCMAPF's performance to that of state-of-the-art planners. In the second set of experiments, we evaluate IDCMAPF's robustness against uncertainties in the speed of the robots.

5.1 Benchmarks and setup

To evaluate the effectiveness of IDCMAPF, four standard MAPF benchmark maps from Stern et al. (2019) are used with varying sizes, obstacle densities, and numbers of robots. Specifically, we use the maps: *empty-48-48*, *random-32-32-20*, *random-64-64-20*, and *warehouse-20-40-10-2-2*. Table 1 summarizes the characteristics of the benchmark maps. In the experiments, we use the random scenarios from the MAPF benchmark, which yield 25 instances for each combination of map and number of robots. The results presented in this section are thus the means of 25 runs for each combination. We evaluate performance in terms of *success rate* (the percentage of the MAPF instances for which the planner can find a solution), and *sum-of-costs* (the sum over all robots of the time steps required to reach their target locations).

5.2 Parameter tuning

In this section, we conduct preliminary experiments to optimize the parameters in IDCMAPF that safeguard robots against livelocks and deadlocks. These parameters are: *waitingThreshold* (WT), *nodesThreshold* (NT), and *repetitionThreshold* (RT). We evaluate the performance of IDCMAPF in terms of *success rate* and *sum-of-costs* on the highly constrained map *random-64-64-20*, where our previous decentralized MAPF solver,

Table 1 Summary of the characteristics of the benchmark maps

Benchmark	Map size	Number of robots
empty-48-48	48×48	From 50 to 400
random-32-32-20	32×32	From 50 to 200
random-64-64-20	64×64	From 50 to 400
warehouse-20-40-10-2-2	164×340	From 50 to 1000

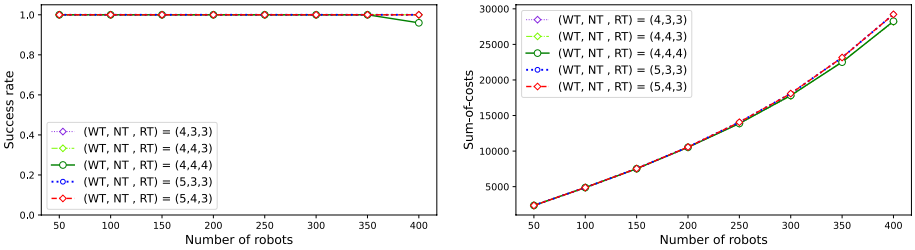


Fig. 7 Performance examples of IDCMAPF in the highly constrained map *random-64-64-20*, with different combinations of values for *waitingThreshold* (WT), *nodesThreshold* (NT), and *repetitionThreshold* (RT). Note that we have only included a selection of the possible combinations values in the plot. Left: *success rate*, right: *sum-of-costs*

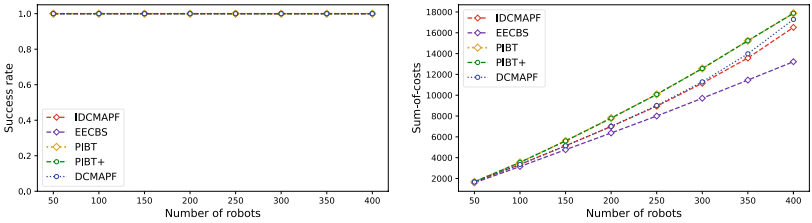
DCMAPF, performed poorly as the number of robots increased. The selection of the WT value necessitates particular care, as it directly influences the behavior of the robots during conflict resolution, a process that may require some robots to wait for others. It is critical that the threshold is not set so low as to be exceeded during a standard conflict resolution process. The same consideration applies to the selection of values for NT and RT. During conflict resolution, robots often must give way to one or more higher-priority robots and therefore need to return to previously visited nodes to continue their paths. To ensure good performance, we conducted systematic testing of these parameter values, focusing specifically on the range of {3, 4, 5}.

We have plotted the results for a selection of the combinations in Fig. 7. Upon examining the results both in terms of *success rate* and *sum-of-costs*, we selected the parameter combination (WT, NT, RT) = (4,4,3), where IDCMAPF attained a *success rate* of 100% and comparatively low *sum-of-costs*.

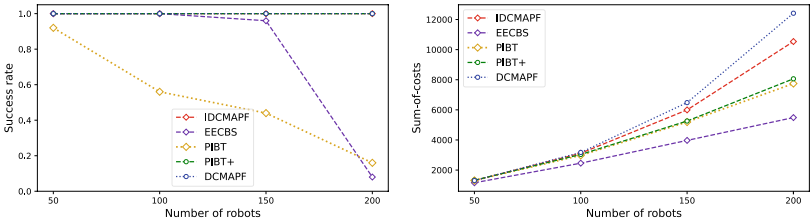
5.3 Comparison against state-of-the-art planners

We compare IDCMAPF to the following state-of-the-art planners: EECBS (Li et al., 2021c), a state-of-the-art bounded sub-optimal search-based planner, DCMAPF (Maoudj & Christensen, 2022), a decentralized planner, and PIBT, and PIBT+ (Okumura et al., 2022) as prioritized planners. These four baselines represent a diverse set of state-of-the-art planners for solving MAPF problems that have been shown to perform well on various MAPF benchmarks, covering both centralized and decentralized approaches as well as different types of prioritization and conflict resolution strategies. The source code for the EECBS, PIBT and PIBT+ planners is available in [1]. EECBS, PIBT and PIBT+ are centralized planners, where robots have full observability of the environment, whereas DCMAPF and our proposed IDCMAPF are decentralized planners, where robots rely exclusively on local observations and cannot access the whole state of the system. For centralized planners, we use the results presented in Okumura et al. (2022), where they were run on a laptop with Intel Core i9 2.3 GHz CPU and 16 GB RAM. The offline planners were given a time limit of 30 s to plan the robots' paths, where 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.

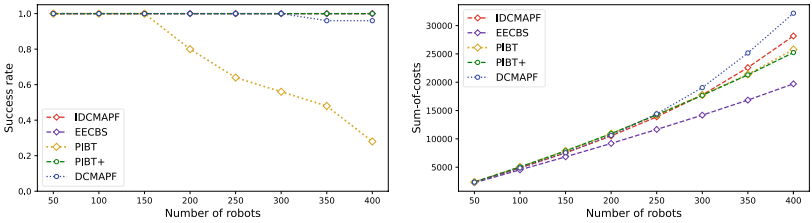
Figure 8 shows the results of the experiments on the benchmark maps considered. Based on these results, we conclude that IDCMAPF provides promising performance as it



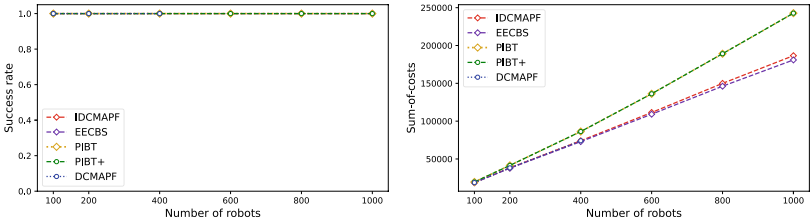
(a) Performance for different planners in *empty-48-48*. Left: *success rate*, right: *sum-of-cost*.



(b) Performance for different planners in *random-32-32-20*. Left: *success rate*, right: *sum-of-cost*.



(c) Performance of different planners in *random-64-64-20*. Left: *success rate*, right: *sum-of-cost*.



(d) Performance of different planners in *warehouse-20-40-10-2-2*. Left: *success rate*, right: *sum-of-cost*. Note that DCMAPF planner only provided results up to a maximum of 400 robots.

Fig. 8 Comparative results in terms of *success rate* and *sum-of-costs* between IDCMAPF, DCMAPF, EECBS, PIBT, and PIBT+ on four benchmark maps for different fleet sizes

can achieve a *success rate* of 100% in all considered maps and thus, is effective for solving MAPF problems in a fully decentralized manner. The same high performance in terms of *success rate* can be observed for the centralized PIBT+, whereas the performance of the decentralized DCMAPF and centralized PIBT decreases in some cases as the obstacle density of the map increases. On the *empty-48-48* map, all planners perform exceedingly well. In contrast, on the maps with high contention due to higher obstacle densities or higher numbers of robots, DCMAPF sometimes fails to achieve a *success rate* of 100% (see *random-64-64-20*) due to its inability to detect and avoid deadlocks. In scenarios involving

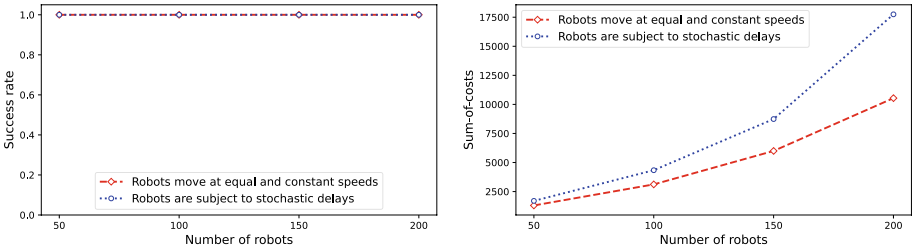
more than 150 robots on the *random-32-32-20* map, EECBS performs significantly worse, achieving a low *success rate*. As the results show, IDCMAPF and PIBT+ outperform the other planners in terms of *success rate* in the highly constrained maps *random-64-64-20* and *random-32-32-20*. Additionally, interesting results can be observed on the large-scale map *warehouse-20-40-10-2-2*, where all planners achieve a *success rate* of 100%.

Regarding the *sum-of-costs* metric, we observed that the performance of IDCMAPF compares well to that of centralized planners, except for a few scenarios with very high robot densities. The exception is the particularly constrained scenarios of the small map *random-32-32-20*, where IDCMAPF's *sum-of-costs* is higher than that of the centralized planners, specifically when the robot count exceeds 150. It is evident from the results that the sub-optimal EECBS planner consistently surpasses the other planners across all maps. However, the *warehouse-20-40-10-2-2* map is an exception, where IDCMAPF yields similar results in terms of *sum-of-costs* to those of the sub-optimal planner EECBS. Although EECBS outperforms other planners in most maps, its performance significantly depends on the map size and robot count (see Fig. 8b where it achieves a low *success rate*). Additionally, even though IDCMAPF employs a strategy of re-planning paths to avoid livelocks and deadlocks, it consistently yields better results in terms of path cost than the decentralized planner DCMAPF across most maps. While IDCMAPF's strategy to avoid livelocks and deadlocks can cause deviations from the initially planned optimal paths, the approach still yielded effective solutions in terms of both *sum-of-costs* and *success rate*.

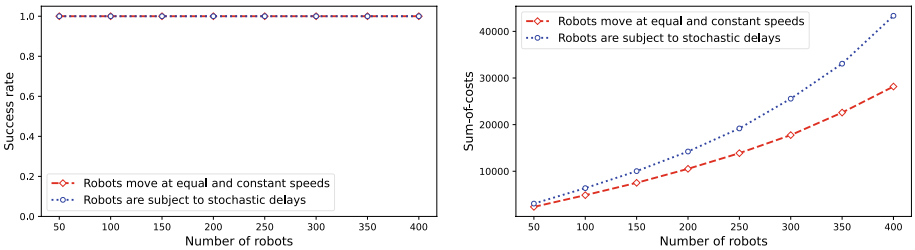
5.4 Sensitivity analysis to the presence of uncertainties in robot speeds

Many state-of-the-art MAPF planners are based on the assumption that the robots move in a deterministic manner. However, in many real-world scenarios, robots cannot precisely follow the offline planned paths due to small variations in their speeds and unexpected events. In the second set of experiments, we therefore evaluate the robustness of IDCMAPF to uncertainties in robot speeds. Specifically, we assess how well IDCMAPF can tolerate random delays in robot movements. For this assessment, we use the challenging maps *random-32-32-20* and *random-64-64-20* and assign each robot a probability of 30% of experiencing a delay at every time step, thus keeping it stationary at its current node for that time step (but still counting the time step toward the *sum-of-costs* if the robot has not yet reached its target).

The results of these experiments are shown in Fig. 9. The results show that IDCMAPF achieves a 100% *success rate* on both maps and thus is able to tolerate stochastic delays. Moreover, as the number of robots increases on both maps, we observe a gradual drop in solution quality. Indeed, introducing stochastic delays inevitably leads to an increase in the *sum-of-costs*, particularly in scenarios with high robot density. As the robot density is increased, robots are more likely to have followers or to be engaged in resolving a conflict, which means each delayed robot is more likely to indirectly delay other robots. On the highly constrained small map *random-32-32-20*, the observed increase in the *sum-of-costs* was from 30% and 46% when the fleet size was increased from 50 robots to 150 robots. For a fleet size of 200 robots, the increase in *sum-of-costs* was 68%. Comparatively, for the map *random-64-64-20*, there is a consistent increase in the cost, reaching 54% when 400 robots are involved. Although delaying robots at each time step inevitably affects the plan cost, IDCMAPF was still able to maintain a 100% *success rate* on both maps, showing that IDCMAPF is able to efficiently coordinate large fleets of mobile robots in highly constrained



(a) Performance in *random-32-32-20*. Left: *success rate*, right: *sum-of-cost*



(b) Performance in *random-64-64-20*. Left: *success rate*, right: *sum-of-cost*

Fig. 9 Robustness of IDCMAPF: comparison between scenarios in which robots move at equal and constant speeds, and scenarios with stochastic delays. Stochastic delays are introduced by randomly deciding, at each time step, whether a robot remains in its current node with a probability of 30%

maps with a high degree of robustness to stochastic delays. Example runs can be found in the supplementary video: https://youtu.be/PrhIZ_mQp5I.

6 Discussion

Our results show that IDCMAPF consistently displays a high performance in terms of *success rate* across all considered benchmark maps and compares very well with the centralized planners in terms of solution quality. IDCMAPF is able to handle stochastic delays because of its decentralized and cooperative approach to conflict handling, and the performance of IDCMAPF degraded gracefully when stochastic robot delays were considered. A potential drawback of entirely decentralized approaches is that deadlocks and livelocks can arise, which we also observed for our previously proposed approach to decentralized cooperative multi-agent path finding, DCMAPF (Maoudj & Christensen, 2022). The two new mechanisms introduced in IDCMAPF were empirically shown to be effective avoiding livelock and deadlock situations. In future work, we plan to investigate if theoretical guarantees for reachability and completeness can be established for IDCMAPF.

Centralized approaches offer the potential advantage of optimality (Sharon et al., 2015) or bounded sub-optimality (Li et al., 2021c). However, aside from potential scalability issues, centralized approaches often require that the environment is static and completely known so that conflicts are avoided at planning time. In our decentralized approach, each robot optimistically plans its path and then, handles conflicts on the fly as they arise. This means that issues, such as stochastic robot delays can be handled seamlessly. The same holds true if a robot breaks down; in our approach, other robots

could consider the malfunctioning robot as a new obstacle similarly to how virtual obstacles are introduced to handle livelocks. A decentralized approach, such as IDCMAPF, thus also offers robustness and flexibility.

In this paper, we focused on the MAPF problem where each agent must move from its initial position to a given goal location. However, given that IDCMAPF is based on on-the-fly local conflict resolution, it would be straightforward to extend our approach to a lifelong version of the MAPF problem, called the *multi-agent pickup and delivery* (MAPD) problem (Ma et al., 2017), where agents are constantly engaged with new tasks.

7 Conclusions

We have contributed to the MAPF literature by proposing a fully decentralized approach to coordination that integrates effective path planning and motion coordination capabilities. The proposed IDCMAPF approach, which relies solely on short-range local communication, is less susceptible to communication issues compared to methods dependent on long-range communication. The motion coordination uses priority rules to ensure reliable and effective conflict resolution. Moreover, IDCMAPF includes specific mechanisms to detect and prevent livelocks and deadlocks. To assess the performance of IDCMAPF, we conducted extensive simulation-based experiments on MAPF benchmark maps. Our results show that IDCMAPF is effective, that it exhibits a consistent high performance across all maps, and that in many cases, it can match or even exceed the performance of the state-of-the-art centralized planners. IDCMAPF scales well to large numbers of robots, and it is robust to stochastic robot delays during execution. Future work will focus on expanding the application domain of IDCMAPF to non-highly controlled dynamic environments, where obstacles stochastically appear and disappear.

Another avenue for future research is to incorporate a probabilistic model of communication noise into our simulations and assess the reliability of our system under different noise levels. Finally, it is necessary to conduct experiments with physical robots in a real-world scenario to fully validate the proposed approach.

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

Author contributions AM contributed to conceptualization, methodology, software, validation, writing, review, and editing. ALC contributed to conceptualization, methodology, software, validation, writing, review, editing, and supervision. All authors read and approved the final manuscript.

Funding This work was supported by the Independent Research Fund Denmark under Grant 0136-00251B.

Data availability Not applicable

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable.

Consent for publication Not applicable.

References

Beinschob, P., & Reinke, C. (2015). Graph SLAM based mapping for AGV localization in large-scale warehouses. In *2015 IEEE International conference on intelligent computer communication and processing (ICCP)*, (pp. 245–248). IEEE.

Bobanac, V., & Bogdan, S. (2008). 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.

CBS, EECBS and PIBT. https://github.com/Jiaoyang-Li/CBSH2-RTC, https://github.com/Jiaoyang-Li/EECBS, and https://github.com/Kei18/pibt2

Damani, M., Luo, Z., Wenzel, E., & Sartoretti, G. (2021). PRIMAL$_2$: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters, 6*(2), 2666–2673.

Draganjac, I., Petrović, T., Miklić, D., Kovačić, Z., & Oršulić, J. (2020). Highly-scalable traffic management of autonomous industrial transportation systems. *Robotics and Computer-Integrated Manufacturing, 63*, 101915.

Gange, G., Harabor, D., & Stuckey, Peter J. (2019). Lazy CBS: Implicit conflict-based search using lazy clause generation. In *Proceedings of the international conference on automated planning and scheduling*, (Vol. 29, pp. 155–162). AAAI Press.

Grenouilleau, F., van Hoeve, W.-J., & Hooker, J. N. (2019). A multi-label A* algorithm for multi-agent pathfinding. In *Proceedings of the international conference on automated planning and scheduling*, (Vol. 29, pp. 181–185). AAAI Press.

Hönig, W., Kumar, T.K., Cohen, L., Ma, H., Xu, H., Ayanian, N., & Koenig, S. (2016). 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.

Kammel, C., Kögel, T., Gareis, M., & Vossiek, M. (2022). A cost-efficient hybrid UHF RFID and odometry-based mobile robot self-localization technique with centimeter precision. *IEEE Journal of Radio Frequency Identification, 6*, 467–480.

Lam, E., & Le Bodic, P. (2020). 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.

Li, J., Chen, Z., Harabor, D., Stuckey, P. J., & Koenig, S.(2021a). Anytime multi-agent path finding via large neighborhood search. In *International joint conference on artificial intelligence*, (pp. 4127–4135). IJCAI.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., Gange, G., & Koenig, S. (2021). Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence, 301*, 103574.

Li, J., Ruml, W., & Koenig, S. (2021c). EECBS: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, (pp. 12353–12362). AAAI Press.

Lian, Y., Xie, W., & Zhang, L. (2020). A probabilistic time-constrained based heuristic path planning algorithm in warehouse multi-AGV systems. *IFAC-PapersOnLine, 53*(2), 2538–2543.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., & Koenig, S. (2019). Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, (pp. 7643–7650). AAAI Press.

Ma, H., Li, J., Kumar, T. K., & Koenig, S. (2017). 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.

Maoudj, A., & Christensen, A. L. (2022). Decentralized multi-agent path finding in warehouse environments for fleets of mobile robots with limited communication range. In *13th international conference on swarm intelligence (ANTS2022)*, (pp. 104–116). Springer.

Maoudj, A., Kouider, A., & Christensen, A. L. (2023). The capacitated multi-AGV scheduling problem with conflicting products: Model and a decentralized multi-agent approach. *Robotics and Computer-Integrated Manufacturing, 81*, 102514.

Okumura, K., Machida, M., Défago, X., & Tamura, Y. (2022). Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence, 310*, 103752.

Rathi, A., & Vadali, M. (2021). Dynamic prioritization for conflict-free path planning of multi-robot systems. arXiv preprint arXiv:2101.01978

Reijnen, R., Zhang, Y., Nuijten, W., Senaras, C., & Goldak-Altgassen, M. (2020). 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.

Ryan, M. R. K. (2008). Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research, 31*, 497–542.

Sajid, Q., Luna, R., & Bekris, K. (2012). Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *International symposium on combinatorial search*, (pp. 88–96). AAAI Press.

Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T. S., Koenig, S., & Choset, H. (2019). Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters, 4*(3), 2378–2385.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence, 219*, 40–66.

Skrynnik, A., Andreychuk, A., Yakovlev, K., & Panov, A. I. (2022). POGEMA: partially observable grid environment for multiple agents. arXiv preprint arXiv:2206.10944

Stephan, J., Fink, J., Kumar, V., & Ribeiro, A. (2017). Concurrent control of mobility and communication in multirobot systems. *IEEE Transactions on Robotics, 33*(5), 1248–1254.

Stern, R., Sturtevant, N. R, Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Satish Kumar, T.K. et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Symposium on combinatorial search (SoCS)*, (pp. 151–158). AAAI Press.

Surynek, P. (2009). 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.

Surynek, P., Felner, A., Stern, R., Boyarski, E. (2016). 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.

Van Den B., Jur P., & Overmars, M. H. (2005). Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ international conference on intelligent robots and systems*, pp 430–435. IEEE.

Varambally, S., Li, J., & Koenig, S. (2022). Which MAPF model works best for automated warehousing? In *Proceedings of the international symposium on combinatorial search*, (Vol. 15, pp. 190–198). IOS Press.

Wagner, G., & Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, (pp. 3260–3267). IEEE.

Dingding, Yu., Xianliang, H., Liang, K., & Ying, J. (2022). A parallel algorithm for multi-AGV systems. *Journal of Ambient Intelligence and Humanized Computing, 13*(4), 2309–2323.

Yu, J., & LaValle, S.M. (2013). 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.

Zhang, Z., Guo, Q., & Yuan, Peijiang. (2017). 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.

Zhao, Y., Liu, X., Wang, G., Shaobo, W., & Han, S. (2020). Dynamic resource reservation based collision and deadlock prevention for multi-AGV. *IEEE Access, 8*, 82120–82130.