Preventing Deadlocks for Multi-Agent Pickup and Delivery in Dynamic Environments

Benedetta Flammini
Politecnico di Milano
Milan, Italy
benedetta.flammini@polimi.it

Davide Azzalini
Politecnico di Milano
Milan, Italy
davide.azzalini@polimi.it

Francesco Amigoni
Politecnico di Milano
Milan, Italy
francesco.amigoni@polimi.it

ABSTRACT

The Multi-Agent Pickup and Delivery (MAPD) problem, in which a team of agents has to plan paths to accomplish incoming pickup and delivery tasks without collisions, has recently attracted significant attention both from academia and industry. In this paper, we consider a MAPD setting in which the environment is *dynamic*, namely it is populated by other moving agents, beyond those belonging to the team. For instance, in a warehouse, moving agents could be humans or cleaning robots. We assume that the team agents cannot communicate with the moving agents and cannot interfere with their tasks and paths, which are a priori unknown and cannot be modified. As a consequence, team agents have to reactively try to solve potential collisions when they appear. However, it can happen that some conflicts are not solvable without affecting the moving agents, resulting in *deadlocks*. Since deadlocks can become rather frequent, especially in crowded environments, in this paper we propose an approach that, by imposing minor constraints on the environment and the movements of the agents, solves potential collisions and prevents the formation of deadlocks by design. Experimental results show that our approach prevents deadlocks, even in very crowded environments, with negligible impact on the performance of task completion.

KEYWORDS

Multi-Agent Pickup and Delivery; Multi-Agent Systems; Deadlock Prevention

ACM Reference Format:
Benedetta Flammini, Davide Azzalini, and Francesco Amigoni. 2024. Preventing Deadlocks for Multi-Agent Pickup and Delivery in Dynamic Environments. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

The Multi-Agent Pickup and Delivery (MAPD) problem, in which multiple agents plan collision-free paths to accomplish incoming tasks, has gained significant attention within both academia and industry due to its several real-world applications. One of the most relevant applications is in automated warehouses [30]. Beyond logistics, other MAPD applications are in aircraft-towing vehicles [23], office robots [29], and automated control of non-player characters in video games [27]. Usually, the agents addressing the MAPD problem are considered the only moving entities in the environment, and several algorithms have been proposed to plan coordinated paths that avoid collisions [19]. However, in some application settings, other moving agents could be present in the environment, either humans or robots [9]. For example, in a warehouse, cleaning robots can operate in the same environment as logistics robots, but the two groups of agents can belong to different companies and operate independently.

In this paper, we consider the MAPD problem in a *dynamic* environment, in which other autonomous moving agents are present, beyond those belonging to the team and performing MAPD. We assume that *team agents* cannot communicate with the other *external agents* and cannot interfere with their tasks and paths, which are unknown and cannot be modified. As a result, collisions between team agents and external agents may happen. When there is a potential collision, the team agents have to reactively avoid the collision and replan their paths. However, it can happen that some of these potential collisions are not solvable by team agents alone, resulting in *deadlocks*. Since deadlocks can become quite frequent, especially in crowded environments, in this paper we propose a new approach to solve potential collisions with the guarantee that deadlocks will not form. We do so by partitioning the environment in *tiles*, and imposing constraints on the number of agents that can be inside each tile at each time step. Through an extensive experimental campaign, we show that our approach prevents deadlocks, even in very crowded environments, with negligible impact on the time required for the completion of tasks.

The main contributions of our paper are: (α) the definition of MAPD in dynamic environments and (β) a new approach that, by tiling grid environments and imposing some constraints on the occupancy of tiles, allows solving the potential conflicts and prevents the formation of deadlocks by design.

2 BACKGROUND

In this section, we review MAPD and the Token Passing algorithm, which is the MAPD solver employed in our proposed solution.

2.1 MAPD

The Multi-Agent Pickup and Delivery (MAPD) problem [20] involves n agents in an environment represented by an undirected connected graph $G = (V, E)$, where the vertices in V represent the locations of the environment, and the edges in E the connections between them. Time is discrete, and at each time step each agent performs an action. Two types of actions are allowed: if at time step t an agent is in $v \in V$, at time step $t + 1$ it can either remain in

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

v or move to an adjacent vertex $v' \in V$ (such that $(v, v') \in E$). All actions are assumed to cost one time step.

A task set $\mathcal{T}$ contains all the tasks that have not been yet assigned and, due to the dynamic nature of the problem, new tasks can be added at any time. Each task $\tau_j \in \mathcal{T}$ is composed of a pickup location $s_j \in V$ and a delivery location $d_j \in V$. When an agent has an assigned task, it is *occupied*, otherwise it is *free*, and it can be assigned any task in $\mathcal{T}$, with the constraint that a task can be assigned to only one agent. An occupied agent becomes free when it completes the assigned task. To complete a task $\tau_j = (s_j, d_j)$, an agent has to go from its current location to the delivery location d_j, passing through the pickup location s_j. So, to solve an assigned task $\tau_j = (s_j, d_j)$, an agent a_i has to plan and perform a sequence of actions (path) $\pi_i = (\alpha_1, \ldots, \alpha_n)$ that brings it from its current location to the pickup location s_j and then to the delivery location d_j. Paths of the agents must not collide, that is: two different agents cannot be in the same location at the same time (*vertex conflict*), and they cannot traverse the same edge in opposite directions at the same time (*swapping conflict*).

The aim of a MAPD problem is to plan paths that complete all the tasks in the shortest time: the quality of a solution is evaluated using either the *service time*, that is the average number of time steps required to complete a task since its appearance in $\mathcal{T}$, or the *makespan*, that is the number of time steps necessary to complete all the tasks (which are assumed to be finite).

Not all MAPD problems are solvable. A sufficient condition to assure that a MAPD instance is solvable is that of being *well-formed* [20]. To illustrate this condition, it is necessary to introduce the concept of non-task endpoints, which are a sort of parking locations in the environment in which agents can ideally stay forever without blocking other agents. The task endpoints, instead, are all the possible pickup and delivery locations. A MAPD instance is well-formed iff (1) there is a finite number of tasks, (2) the number of non-task endpoints is at least equal to the number of the agents, and (3) for every pair of endpoints there exists at least a path that connects them without traversing any other endpoint.

2.2 Token Passing

Ma et al. [20] present different MAPD algorithms that solve well-formed MAPD instances, both centralized and decentralized. Centralized algorithms have better performance in terms of service time and makespan, but require higher computational costs, while decentralized algorithms generally perform worse but are more suitable for real-time tasks. Token Passing (TP, Algorithm 1) [16] is a decentralized MAPD algorithm in which each agent assigns itself to tasks and plans its collision-free paths exploiting some global information on the environment and other agents. This global information is contained in the *token*, that is a synchronized block of memory shared among the agents that includes the task set $\mathcal{T}$, tasks' assignments, and current agents' paths $\{\pi_i\}$.

In the TP algorithm, an agent a_i with an assigned task τ_j uses a path planner (like A* or Dijkstra) to find minimum-cost collision-free paths in a space whose states are pairs composed of a location and a timestamp. There exists an edge between state (v, t) and state $(v', t+1)$ with $v, v' \in V$, if $(v, v') \in E$ or $v = v'$. A state $(v', t+1)$ is removed from the state space when an agent a_i in state v' at time

Algorithm 1 TP

```
1:  /* system executes now */;
2:  initialize token with the (trivial) path ⟨loc(a_i)⟩ for each agent a_i
    (loc(a_i) is the current location of a_i);
3:  while true do
4:      add new tasks, if any, to the task set T;
5:      while agent a_i exists that requests token do
6:          /* system sends token to a_i and a_i executes now */;
7:          T' ← {τ_j ∈ T | no path in token ends in s_j or in d_j};
8:          if T' ≠ {} then
9:              τ ← argmin_{τ_j ∈ T'} h(loc(a_i), s_j);
10:             assign a_i to τ;
11:             remove τ from T;
12:             update a_i's path in token with the path returned by
                PathPlanner(a_i, τ, token);
13:         else if no task τ_j ∈ T exists with s_j = loc(a_i) then
14:             update a_i's path in token with the path ⟨loc(a_i)⟩;
15:         else
16:             update a_i's path in token with Idle(a_i, token);
17:         end if
18:         /* a_i returns token to system, which executes now */;
19:     end while
20:     agents move along their paths in token for one time step;
21:     /* system advances to the next time step */;
22: end while
```

$t+1$ results in a vertex collision with other agents according to their paths in the token. Similarly, the edge between state (v, t) and state $(v', t+1)$ is removed if the traversal of the edge by agent a_i results in a swapping conflict.

At the beginning, the token is initialized considering that each agent trivially remains at its starting location (line 2). At each time step, the system can add new tasks to the task set (line 4). Each free agent requires the token once per time step: the system sends the token to each agent that has requested it, one after the other (line 5). The agent with the token assigns itself to the task closest to its current position (line 10) according to a heuristic function h (line 9, in our experiments h is the Manhattan distance), if there is not any agent currently assigned to a task that ends at same pickup or delivery locations (line 7). After the assignment of the task, a minimum-cost path is planned from the current location of the agent to the delivery location passing through the pickup location, with the condition of being free of collisions with respect to other agents' paths in the token (line 12). Finally, the agent gives back the token to the system and moves according to the path in the token (lines 18 and 20). If no task satisfies the above condition or there is not any feasible path, the agent updates its path in the token with the position in which it is staying (line 14) or calls the *Idle* function to compute a path to a non-task endpoint (line 16).

3 RELATED WORK

We review the literature on the problem of avoiding collisions and deadlocks in multi-robot systems, that could happen due to poorly coordinated plans or to delays or errors at execution time.

In the context of MAPD and MAPF (Multi-Agent Path Finding [19], a simpler "one shot" version of MAPD), Fujitani et al. [11] extend the priority inheritance with backtracking approach [25] for

iterative MAPF to environments containing dead-ends, in which agents plan following a priority order and decide their next actions according to their neighbors, by introducing temporary priorities and limiting agents' moves in dead-ends. Yamauchi et al. [32] tackle the problem of deadlocks for MAPD in non-well-formed environments: they introduce the concept of standby nodes, that are locations in which agents can stay for a long period without affecting the connectivity of the graph, and that change according to the paths of the agents. Liu et al. [16] introduce a MAPD deadlock avoidance method called reserving dummy paths, which consists of reserving paths with minimum travel time to the parking location of each agent. Also TP [20] can be considered as a MAPD deadlock-free method under the assumption of well-formedness. In [24], a conflict-free and deadlock-free more efficient version of TP is studied. In [6], authors present the Push and Rotate algorithm, that overcomes some shortcomings of Push and Swap [18] and is able to guarantee completeness for MAPF instances with at least two unoccupied locations in a connected graph. In [22], it is presented a MAPD deadlock-free distributed planning method where it is assumed that agents can travel asynchronously at different speeds. However, all the above solutions are not directly applicable to our case, since in our setting the team of agents is not alone but has to deal with the presence of independent external agents that cannot be directly controlled (and they cannot be involved in a common planning process with team agents).

In [28] an approach to the n-reciprocal collision avoidance problem is proposed that, by considering the velocities of the agents, reduces the problem to the resolution of a linear program. Other techniques to tackle this problem range from velocity modeling [26, 28] and Model Predictive Control [21] to Reinforcement Learning [2, 4, 8, 15]. In these cases, agents act independently without communication, but our problem is different since team agents are coordinated and have to prioritize the paths of external agents.

Our problem shares some similarities with dynamic obstacle avoidance, usually addressed by methods based on the velocity of the agents [14, 33]. For example, [12] proposes a Probabilistic Velocity Obstacle approach in a dynamic occupancy grid to estimate the velocity and position of moving objects and to avoid collisions. Other methods solve collisions in a reactive way [5, 10, 31], while in [3, 7] a stochastic robotic planning approach is employed. However, our problem differs from the dynamic obstacle avoidance problem in some aspects: in our dynamic environments we can reasonably know what the next actions of external agents will be(see next section) and we focus our attention on preventing the formation of deadlocks, not just on avoiding collisions.

4 PROBLEM FORMULATION

We first introduce our MAPD variant. Then, we present a simple variation of the TP algorithm that avoids collisions without guaranteeing that deadlocks will not form. This method will be used as a baseline in our experiments.

4.1 MAPD in Dynamic Environments

We consider a team of n agents $\mathcal{A} = \{a_1, ..., a_n\}$, called *team agents*, that move in an environment represented as an undirected connected graph $G = (V, E)$, and perform a MAPD instance. We restrict

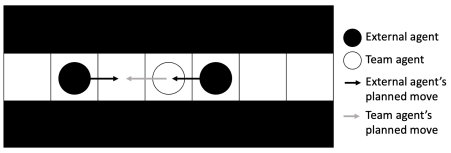


Figure 1: Example of deadlock.

G to be a 4-connected *grid*, which can represent most of the environments of practical interest (extending results of Section 5 to more general graphs is not immediate and will be investigated in future work). The team agents are able to communicate with each other and know the environment G. The team agents aim to complete all the pickup and delivery tasks in $\mathcal{T}$ minimizing a cost measure, such as the service time or the makespan.

In the same environment, there is a set of k *external agents* $\mathcal{M} = \{m_1, ..., m_k\}$, that are not necessarily a team of coordinated agents, moving to perform some tasks. We assume that the team agents and the external agents are neither collaborative nor adversarial and that they do not know each other's tasks and paths. We also make a *no-interference assumption*, namely, we assume external agents' tasks and plans to be immutable, and hence only team agents are in charge of implementing behaviors to avoid collisions. Hence, if a team agent a_i, while following its path π_i detects a potential collision (i.e., a conflict) with an external agent m_j, a_i will have to make a move and modify its planned path in order not to collide with m_j that, from its side, does not do anything to avoid the collision.

We assume that each team agent $a_i \in \mathcal{A}$ can detect external agents within a field of view $FOV(a_i) = \{l \in V \mid \exists \pi = (loc(a_i), \ldots, l) \text{ with } |\pi| \leq 2\}$, which covers all locations l of the environment that are reachable from the current location of a_i with paths π of length 2 or less (excluding wait actions). We assume that team agents know the locations and the next actions of external agents within their field of view. For example, in a warehouse, this amounts to assuming that team robots can detect the current heading of external robots in their field of view and that external robots are equipped with turn signals, which is the case for several warehouse robots. The team agent a_i observes the next action of the external agents in its field of view, and if there is no risk of conflict it continues on its path π_i, while if the agents will collide given the current path π_i and the next moves of external agents in $FOV(a_i)$, the team agent a_i has to change its planned action to avoid the conflict (see Section 4.2 for details).

Since each team agent is only aware of the actions of the external agents in its local field of view, and external agents do not change their paths, there could be potential collisions that cannot be solved, which we call *deadlocks*. For example, let us suppose to be in a narrow corridor, as the one shown in Figure 1. The team agent cannot move in any direction without incurring in a collision: it cannot go left or stand still, otherwise, there would be a vertex conflict, and it cannot go right, since it would cause an edge conflict.

To formally define a deadlock, assume that, performing their intended actions, there will be a collision at the next time step between a moving agent $m_j \in \mathcal{M}$ and a team agent $a_i \in \mathcal{A}$. It can be a vertex conflict, meaning that the location that agent m_j

Algorithm 2 TP with collision avoidance and replanning

1: /* system executes now */;
2: initialize *token* with the (trivial) path $\langle loc(a_i) \rangle$ for each team agent a_i ($loc(a_i)$ is the current location of a_i);
3: **while** true **do**
4: proceed like in Algorithm 1 (lines 4 - 19);
5: **for all** team agents $a_i \in \mathcal{A}$ **do**
6: $C \leftarrow CheckCollisions(loc(a_i))$;
7: **if** $C = \emptyset$ **then**
8: a_i moves along its path in *token* for one time step;
9: **else**
10: a_i moves to $BestLegalMove(a_i, loc(a_i), C, token)$;
11: $\pi_i \leftarrow PathPlanner(a_i, \tau_i, token)$ ▷ replanning for task τ_i assigned to a_i
12: **end if**
13: **end for**
14: /* system advances to the next time step */;
15: **end while**

intends to occupy at time $t+1$ is equal to the location that a_i intends to occupy at time $t+1$, that is $loc^{t+1}(m_j) = loc^{t+1}(a_i)$, or a swap conflict, in which $loc^{t+1}(a_i) = loc^t(m_j)$ and $loc^{t+1}(m_j) = loc^t(a_i)$. To avoid the collision, the team agent a_i has to change its action and choose another one from the set $A_{a_i}^{t+1}$ of actions that would not result in a conflict with any moving or team agent at time $t+1$ (see Section 4.2 for details). We have a deadlock when the set $A_{a_i}^{t+1}$ is empty since there is no legal action for the team agent a_i. The idea can be extended to multiple team and external agents. In general, when the set $A_{\mathcal{A}}^{t+1} = \langle loc^{t+1}(a_1), \ldots, loc^{t+1}(a_n) \rangle$, which includes all the possible legal combinations of actions of the team agents at time $t+1$, is empty, it does not exist a configuration at time $t+1$ for the team agents such that they all can perform a legal action. This means that, even if all team agents cooperate to solve the potential conflict, it is still not possible to avoid collisions with external agents. This situation is considered a deadlock since it cannot be solved only by the team agents (violating the no-interference assumption). In practice, deadlocks could require human intervention to manually move the agents (see Section 6.3).

The problem we address in this paper is that of solving potential collisions and guaranteeing that deadlocks cannot form. Before introducing it, we detail a deadlock-prone method that team agents can use to try to solve potential collisions.

4.2 TP with Collision Avoidance and Replanning

As we have seen, being external agents' tasks and paths unknown to the team agents, collisions may happen when team agents follow their paths. We show a simple variant of the TP algorithm to be run by team agents in which, before performing an action, a team agent checks if such an action would result in a collision and, if so, it modifies its plan.

Algorithm 2 reports the *TP with collision avoidance and replanning* algorithm (*TP-CA*), which is equal to the TP algorithm introduced in Section 2.2 (see Algorithm 1) except for how actions are executed. In TP-CA, before executing an action, a team agent checks if that action would result in a collision with surrounding external agents (line 6). A collision happens when the next move of an external agent inside $FOV(a_i)$ results in the two agents occupying the same location or in swapping their locations. Note that, since team agents use TP(-CA), their planned paths are guaranteed to be collision-free. In this work, we do not consider conflicts that could arise from errors in execution [13, 17].

In case collisions are detected, team agent a_i performs the best (i.e., the one that gets it closer to its goal) legal action from $loc(a_i)$. Legal actions are those that would not result in a collision with any external agent, another team agent, nor an obstacle (line 10). Once moved and prevented the collision, a_i updates the token with a new path starting from its new location (line 11). Note that undoing the last move performed to avoid the collision and resuming the original path (to save the cost of replanning) is not generally feasible as there would be no guarantee that the (now delayed) path would not conflict with those of other team agents.

Using TP-CA, there may be cases in which there is no legal move to be performed by a team agent to prevent a collision with an external agent, which is a deadlock. The frequency of deadlocks depends both on the configuration of the environment and on the number of agents, as we will see in Section 6.

5 PROPOSED APPROACH

In this section, we propose a solution that solves potential collisions and prevents the formation of deadlocks by design. The idea is to impose some (minor) constraints on the configuration of the environment and the possible movements of the team and external agents (Section 5.1). If these conditions hold, it is always possible to solve conflicts, thus avoiding deadlocks (Section 5.2), using a modification of the TP-CA algorithm (Section 5.3).

5.1 Assumptions

We impose that the free area of the grid environment in which agents move is tileable by 2×2 non-overlapping tiles. Considering grids that can be tiled in this way is not too strict, since regular environments like warehouses can easily satisfy the constraint. An algorithm that checks whether a given environment is tileable by 2×2 squares and, if it is, returns the corresponding tiling, is proposed in [1].

It is not necessary that the tiling covers the whole environment, but we impose that the team agents can only move on the portions of the environment that are tiled. Thus, our approach works also with environments for which we can find a 2×2 tiling such that all the pickup and delivery locations of the team agents are covered by the tiles, and it is always possible to find a path between any pair of these locations that traverses only cells in the tiles. The environment tiling is unique and is known to both team and external agents.

Besides tiling of environment, we constrain the movements of team and external agents. In particular, we impose that:

(1) team agents can only move on locations that are covered by the tiling, while external agents can move anywhere in the environment;
(2) there can be at most three team agents in each tile at the same time;
(3) there can be at most one external agent in each tile at the same time.

5.2 Deadlock Prevention

In this section, we show that, given the above constraints, it is always possible to solve potential conflicts between team and external agents, and thus deadlocks are avoided.

Proposition 1. *Given a MAPD in dynamic environment (with the no-interference assumption, Section 4.1) and assuming (1), (2), and (3) of Section 5.1, deadlocks can be avoided.*

Proof sketch. We need to prove that potential collisions between external and team agents can always be avoided by actions of team agents (Section 4.1). We distinguish two cases: conflicts that happen inside a tile and conflicts that happen when external agents move from one tile to an adjacent one.

Conflicts inside a tile. Assume that the potential conflict happens between a team agent and an external agent in the same tile: we show that is always possible to solve the conflict by moving team agents only inside the tile. Looking at Figure 2, even in the worst case, i.e., when there are 3 team agents in the tile (which is the maximum number of allowed team agents in one tile), they can rotate (counterclockwise, in the example) such that the conflict is solved and all the team agents remain inside the tile. Note that the moves that allow the team agents to rotate and avoid the conflict are not those prescribed by their planned paths, otherwise there would have been no potential collision. The situation in Figure 2 with three team agents inside one tile can be generalized. If there are fewer than three team agents inside the tile, they can still rotate or move to free the destination cell of the external agent and also avoid edge conflicts. The situation in which the external agent wants to perform the other allowed move is symmetric with respect to the one depicted (team agents would have to rotate clockwise). Since, as shown, it is always possible to solve conflicts that happen inside a tile by moving team agents inside the same tile, we can treat potential conflicts in different tiles as independent.

Conflicts involving adjacent tiles. If a potential conflict happens between an external agent and a team agent that are in two different tiles, as in Figure 3, our assumptions imply that the external agent is moving towards a tile where there are no external agents, or where there is an external agent that is leaving that tile at the same time step, since there can be at most one external agent in each tile. This means that there is at least one empty location in the tile towards which the external agent is moving. The presence of this empty location always allows team agents to rotate or move to avoid the conflict (similar to the previous case), thus preventing the deadlock. Also in this case, the resolution of the conflict happens entirely in the destination tile of the external agent, implying that each tile can be considered independently.

We can conclude that, since it is always possible to solve all conflicts locally, i.e., by not moving team agents outside the tile in which the conflict happens, conflicts can be considered independent from one another. This allows us to say that there will be no deadlocks over the whole environment since team agents can always move inside tiles. □

Overall, we have thus proved that under the assumptions of Section 5.1, it is possible for the team agents to always avoid deadlocks by making appropriate moves to solve potential collisions. In the next section, we introduce an algorithm that allows the team agents to perform these appropriate moves.

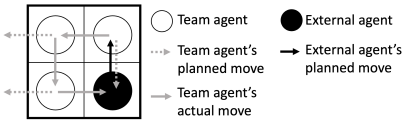


Figure 2: Conflict between agents inside a tile.

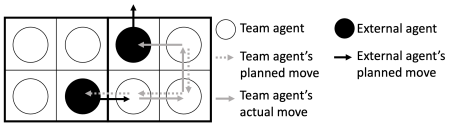


Figure 3: Conflict between agents in two different tiles.

5.3 TP with Collision Avoidance and Replanning + Tiling

In Algorithm 3, called *TP with collision avoidance and replanning + tiling (TP-CA-T)*, we add to TP-CA the tiling of the environment, the constraints on movements for the team agents, and the local decisions of Section 5.2 to avoid potential collisions and guarantee the absence of deadlocks. We remark that external agents can use any planning algorithm, as long as there is only one external agent in each tile at any given time step, as specified in Section 5.1.

In the base TP algorithm (Section 2.2), an agent a_i uses a path planner to find minimum-cost collision-free paths in a space whose states are pairs composed of a location and a timestamp. Those states that result in a vertex or edge conflict with the paths already stored in the token are removed. In TP-CA-T, also those states that do not respect the condition relative to the maximum number of team agents in each tile are removed. This is implemented in a modified *PathPlanner* algorithm that considers also the tiling of the environment (lines 12 and 32). In this way, the constructed paths are both collision-free and compliant with our assumptions.

When a team agent has to reactively avoid collisions with external agents, it calls the *BestMove* function (lines 23 and 31), which at first tries to select that action that does not result in a conflict with an external agent, another team agent, nor an obstacle, and is in the same tile as the agent's location. If this action cannot be found, as it happens when deadlocks could form, it chooses an action that does not result in a conflict with an external agent or an obstacle and is in the same tile as the agent's location. This implies that the chosen action could result in a collision with another team agent. After the agent has chosen an action, its choice is considered fixed, meaning that it cannot be changed, and the agent is added to the F_{ag} set (line 24), which contains team agents that have already changed their paths to avoid collisions: this prevents loops of team agents that continuously change their actions. After this, it is checked if the new action of the team agent results in a conflict with those of the others: if so, the other team agents change their actions in the same way and are added to F_{ag}. This process (lines 20-28) is iterated until all team agents perform an action that does not result in a conflict with others: it is always possible to reach this equilibrium point since, as shown in Section 5.2, all conflicts can be solved remaining inside the tile in which they happen. Then, team agents that have

Algorithm 3 TP with collision avoidance and replanning + tiling

```
1:  /* system executes now */;
2:  initialize token with the (trivial) path ⟨loc(a_i)⟩ for each team agent a_i
    (loc(a_i) is the current location of a_i);
3:  while true do
4:      add new tasks, if any, to the task set 𝒯;
5:      while team agent a_i exists that requests token do
6:          /* system sends token to a_i and a_i executes now */;
7:          𝒯' ← {τ_j ∈ 𝒯 | no path in token ends in s_j or in d_j};
8:          if 𝒯' ≠ {} then
9:              τ ← argmin_{τ_j ∈ 𝒯'} h(loc(a_i), s_j);
10:             assign a_i to τ;
11:             remove τ from 𝒯;
12:             update a_i's path in token with the path returned by
                PathPlanner(a_i, τ, token, tiling);
13:         else if no task τ_j ∈ 𝒯 exists with s_j = loc(a_i) then
14:             update a_i's path in token with the path ⟨loc(a_i)⟩⟩;
15:         else
16:             update a_i's path in token with Idle(a_i, token);
17:         end if
18:         /* a_i returns token to system, which executes now */;
19:     end while
20:     while all team agents a_i ∈ 𝒜 are free from collisions do
21:         C_i ← CheckCollisions(loc(a_i), F_ag);
22:         if C_i ≠ ∅ then
23:             a_i chooses BestMove(a_i, loc(a_i), C, token, tiling, F_ag);
24:             add a_i to F_ag;
25:         else
26:             continue
27:         end if
28:     end while
29:     for all team agents a_i ∈ 𝒜 do
30:         if a_i ∈ F_ag then
31:             a_i moves to BestMove(a_i, loc(a_i), C, token, tiling, F_ag)
32:             π_i ← PathPlanner(a_i, τ, token, tiling)        ▷ replanning
33:         else
34:             a_i moves along its path in token for one time step;
35:         end if
36:     end for
37:     /* system advances to the next time step */;
38: end while
```

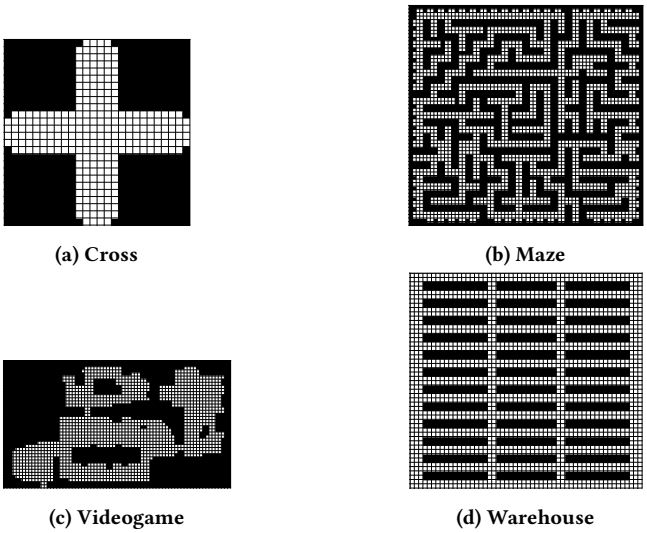
to perform an action to avoid collisions replan their paths, and all agents move one step forward (lines 29-35).

6 EXPERIMENTS

6.1 Experimental Setting

To evaluate the performance of our approach, we test it in four different fully-tileable environments. We compare against TP-CA, namely a method that solves potential conflicts without guaranteeing that deadlocks will not form, since we are not aware of any other method that guarantees the absence of deadlocks in our MAPD setting (see Section 3). We also consider an ideal fully controlled (FC) approach in which team and external agents use TP sharing a unique token. While practically unrealistic, FC provides the best way in which external and team agents can coordinate.

The environments are a 26×26 cells cross environment (Figure 4a), a 66×62 cells maze environment (Figure 4b), a 73×41 cells



(a) Cross **(b) Maze** **(c) Videogame** **(d) Warehouse**

Figure 4: Experimental environments.

video game map (Figure 4c), and a 54×50 cells warehouse map (Figure 4d). We assume that also the external agents are performing pickup and delivery tasks and use the TP algorithm when the team agents use TP-CA and, when the team agents use TP-CA-T, the external agents use a modified version of TP (denoted TP-mod) to comply with the tiling constraint. (External agents can do anything, as long as they satisfy the constraints of Section 5.1. We assume that they solve a MAPD problem and use TP and TP-mod just for simplicity.) In all cases, the path-finding algorithm is A*. Tasks are created by choosing pickup and delivery locations uniformly at random among a set of predefined vertices, and tasks' arrival times are sampled uniformly at random from predefined time intervals. Non-task endpoints and delivery locations are usually located at the border of the environments, while pickup locations are sparsely diffused in the environments. Table 1 summarizes the experimental parameters: for each environment, it is specified its size, the number of team and external agents, the number of tasks that team and external agents have to accomplish, the time interval in which team and external agents' tasks appear in the environment, the number of possible pickup locations (that are the same for team and external agents), and the number of possible delivery locations for team and external agents (which are different for team and external agents). These parameters have been set manually to make the environments crowded and create a challenging test bed for our approach. Slight variations in the values of parameters produce qualitatively similar results.

We measure the number of deadlocks over the runs and, considering only those runs that do not end in a deadlock, we measure the makespan of both the team agents and the external agents, that is the time required to complete all the assigned tasks. The number of deadlocks provides the most important information about the effectiveness of our method, indicating whether all tasks of a run will be completed from both sides, whereas the makespan is a measure of efficiency. In particular, the comparison of the team and external agents' makespan with and without our approach

	Cross	Maze	Videogame	Warehouse
size	26 × 26	66 × 62	73 × 41	54 × 50
#team agents	22	45	48	22
#external agents	22	30	41	26
#team tasks	100	110	160	110
#external tasks	155	100	150	140
team task interval	[50,100]	[30,60]	[30,100]	[30,60]
external task interval	[0,100]	[0,50]	[0,80]	[0,50]
#pickups	52	1948	100	1056
#team deliveries	24	52	52	23
#external deliveries	24	39	50	29

Table 1: Experimental parameters.

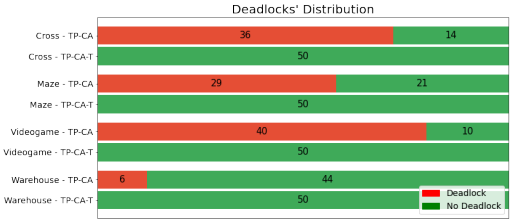


Figure 5: Number of deadlocks over 50 runs.

allows us to understand the effect of the constraints on the time of completion of the tasks. Results are averaged over 50 runs, where each run is different from the others for the arrangement and time of arrival of the tasks. Experiments have been conducted on a 2.10 GHz Intel(R) Xeon(R) Silver 4116 CPU with 32 GB of RAM.

6.2 Experimental Results

Figure 5 reports the number of deadlocks experienced in each environment averaged over 50 runs when team agents use TP-CA and when they use TP-CA-T. With FC there are no deadlocks by design since all agents behave as a single team. When team agents use TP-CA and external agents use TP, the environments that are affected the most by the presence of deadlocks are the cross, the maze, and the videogame maps. In the case of the cross and the maze maps, deadlocks happen frequently because of narrow portions of the environments with dead-ends, in which it is easy for team agents to get trapped by external agents. The videogame map features several open spaces connected by narrow corridors: it is in these corridors that most deadlocks happen. In the warehouse environment, only 6 runs over 50 end in deadlock: the presence of three detached shelves in each aisle allows team agents to easily change their paths without remaining trapped in long aisles or dead-ends, and the fact that aisles are 2 cells wide prevents the formation of deadlocks by giving alternative possibilities for collision avoidance. From Figure 5, the effectiveness of the proposed method appears clearly: as expected from results of Section 5, no deadlocks happen even in crowded or narrow environments, in which the possibility of ending in a deadlock is high, allowing all agents to complete their tasks by imposing few constraints.

To evaluate the impact of these constraints, Figure 6 and Figure 7 report the makespan of the team agents and the makespan of the external agents, respectively, comparing TP-CA, TP-CA-T, and FC for team agents, and TP, TP-mod, and FC for external agents. For

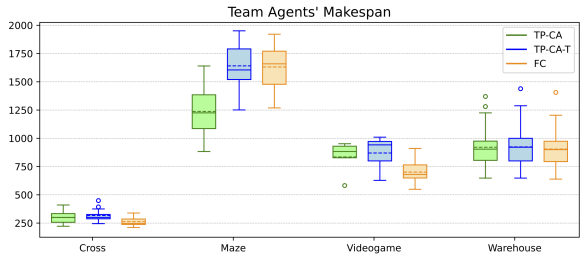


Figure 6: Makespan of the team agents.

	Cross	Maze	Videogame	Warehouse
TP-CA-T wrt TP-CA	+5.52%	+32.62%	+4.32%	+0.52%
TP-CA-T wrt FC	+16.67%	+0.52%	+19.00%	+1.72%

Table 2: Average increase of team agents' makespan.

each environment, we only consider those runs that do not end in a deadlock for all three methods, to have a fair comparison. For the team agents, in the cross, videogame, and warehouse maps, the makespan is similar when employing TP-CA and TP-CA-T, as also shown in Table 2. This means that the constraints imposed on team agents are not too limiting. Only in the maze environment, there is a big difference between the TP-CA and the TP-CA-T case, with an average makespan 32.62% higher for the latter. This difference comes from the structure of the environment and the tiling constraint: since the maze has narrower corridors and many dead-ends, team agents are forced to replan more often than in other environments to avoid collisions with external agents. In fact, looking at Figure 8, which reports the number of replans that team agents have to perform in order to avoid collisions with external agents, in the maze map replans are more for TP-CA-T than for TP-CA, causing an increase in the makespan due to the frequent changes of paths. Besides, the tiling constraints on both the team and external agents force them to take alternative paths to satisfy the maximum number of agents in each tile, making them more sparse on the map. For the same reason, the number of team agents' replans is slightly larger in the TP-CA-T case than in the TP-CA case also for other environments.

Makespan of the external agents (Figure 7), is slightly larger with TP-mod than with TP in all the environments (see also Table 3). Although the constraints imposed on external agents are stricter than those imposed on the team agents, they are able to complete their tasks without big impacts on the quality of their activities.

The makespan of team and external agents when using TP-CA-T and TP-mod (respectively) is larger than when using FC, due to weaker coordination. For external agents the maximum increase (over the environments) is +5.5%, showing that our approach does not penalize much external agents. For team agents, the increase is +1.7%, +0.5%, +16.7%, +19.0% (warehouse, maze, videogame, cross): team agents are more penalized by our approach in open environments in which replans are frequent.

Experimental results show that our method is effective in preventing deadlocks even in narrow and crowded environments, by imposing minor constraints on the environment and the movements

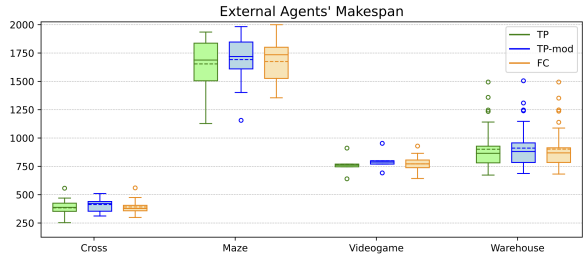


Figure 7: Makespan of the external agents.

	Cross	Maze	Videogame	Warehouse
TP-mod wrt TP	+7.08%	+2.51%	+4.54%	+1.20%
TP-mod wrt FC	+5.46%	+1.14%	+3.57%	+1.15%

Table 3: Average increase of external agents' makespan.

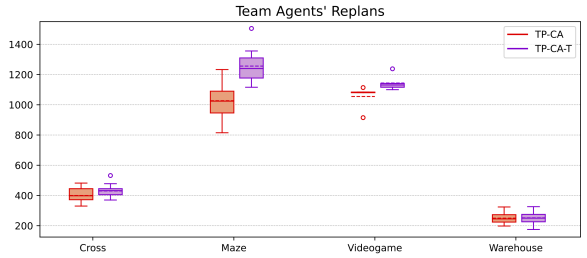


Figure 8: Number of replans of the team agents.

of the agents. These constraints do not generally have a strong effect on the efficiency of the agents, whose makespan increases only slightly. In narrower environments with many dead-ends, as in mazes, the constraints increase in the makespan for the team agents due to the structure of the environment, but our method is able to guarantee that all agents complete all their tasks without incurring in deadlocks.

6.3 Convenience Analysis

Solving deadlocks in real settings could require manual interventions. For example, a human operator could move the robots involved in the deadlock to their non-task endpoints, or rearrange their locations to force the resolution of the deadlock. To quantitatively evaluate whether this approach is more convenient than ours (i.e., whether it is better to use our approach that prevents deadlocks or to let deadlocks happen and then solve them manually), we define the cost c (in time units) of a deadlock as:

$$c = p \cdot T, \tag{1}$$

where p is the probability of formation of a deadlock and T is the mean time needed for a human operator to solve a deadlock. Given an environment, the probability p that a deadlock will form can be computed empirically from our data as the number of runs that end in deadlock divided by the number of total runs: $p = \frac{\#deadlock\_runs}{\#total\_runs}$. We can compute p for each environment from Figure 5, as reported in the first row of Table 4. In an environment, our approach is more

	Cross	Maze	Videogame	Warehouse
p	0.72	0.58	0.80	0.12
T	31.08	226.07	40.75	75.00

Table 4: Convenience analysis' parameters and results.

convenient than solving deadlocks manually when:

$$M^{\text{TP-CA-T}} - M^{\text{TP-CA}} < c, \tag{2}$$

where $M^{\text{TP-CA-T}}$ and $M^{\text{TP-CA}}$ is the mean number of time steps such that all the team and external agents finish their tasks relative to TP-CA-T and to TP-CA, respectively, which is as an upper bound to the makespan of team agents. From (1) and (2), we derive that our approach is more convenient when:

$$T > \frac{M^{\text{TP-CA-T}} - M^{\text{TP-CA}}}{p}. \tag{3}$$

Note that, in (3), we are implicitly assuming that in each run only a single deadlock could happen, meaning that, after a deadlock has been manually solved, the run will proceed to the end without the formation of other deadlocks. Thus, (3) provides a lower bound on the time required by the human to solve a deadlock, and is an optimistic estimate for the manual resolution of deadlocks. Table 4 (second row) reports the values of the smallest T that satisfies (3). Note that T is expressed in terms of time steps, and a time step is the time required by an agent to move from a cell to an adjacent cell. Just for illustration purposes, let us assume that one time step lasts 4 seconds. This means that, in the case of the warehouse, our approach is better than human intervention when manually solving a deadlock requires more than $75.00 \times 4 = 300$ seconds, namely 5 minutes. Considering that a human operator has to control remotely the robots or enter the environment, an intervention time of more than 5 minutes could be likely in many settings, making our approach appealing for employment in real applications.

7 CONCLUSION

We presented an approach that solves potential conflicts in a MAPD setting in which team agents operate in a dynamic environment with external agents that do not change their tasks and paths. By imposing simple constraints on the environment and on the agents' movements, our approach prevents deadlocks by design. Experiments have been performed on four different maps, confirming the effectiveness of our approach even in crowded and narrow environments. Moreover, the constraints that our approach imposes only slightly increase the makespan.

Future research directions include the extension of our approach to more general environments, beyond grids, the improvement in efficiency of the proposed solution, and the investigation of a game-theoretical formulation of the problem.

ACKNOWLEDGMENTS

Benedetta Flammini is financially supported by the ABB – Politecnico di Milano Joint Research Center. This paper is supported by PNRR-PE-AI FAIR project funded by the NextGeneration EU program.

REFERENCES

[1] Anders Aamand, Mikkel Abrahamsen, Peter MR Rasmussen, and Thomas D Ahle. 2023. Tiling with squares and packing dominos in polynomial time. *ACM Transactions on Algorithms* 19, 3 (2023), 1–28.

[2] Hyansu Bae, Gidong Kim, Jonguk Kim, Dianwei Qian, and Sukgyu Lee. 2019. Multi-robot path planning method using reinforcement learning. *Applied sciences* 9, 15 (2019), 3057.

[3] Lars Blackmore, Masahiro Ono, and Brian Williams. 2011. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics* 27, 6 (2011), 1080–1094.

[4] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan How. 2017. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *Proc. ICRA*. 285–292.

[5] Aurélie Clodic, Vincent Montreuil, Rachid Alami, and Raja Chatila. 2005. A decisional framework for autonomous robots interacting with humans. In *Proc. ROMAN*. 543–548.

[6] Boris De Wilde, Adriaan Ter Mors, and Cees Witteveen. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research* 51 (2014), 443–492.

[7] Noel Du Toit and Joel Burdick. 2011. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics* 28, 1 (2011), 101–115.

[8] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. 2020. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research* 39, 7 (2020), 856–892.

[9] Benedetta Flammini, Davide Azzalini, and Francesco Amigoni. 2023. Multi-Agent Pickup and Delivery in Presence of Another Team of Robots. In *Proc. AAMAS*. 2562–2564.

[10] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* 4, 1 (1997), 23–33.

[11] Yukita Fujitani, Tomoki Yamauchi, Yuki Miyashita, and Toshiharu Sugawara. 2022. Deadlock-Free Method for Multi-Agent Pickup and Delivery Problem Using Priority Inheritance with Temporary Priority. *Procedia Computer Science* 207 (2022), 1552–1561.

[12] Chiara Fulgenzi, Anne Spalanzani, and Christian Laugier. 2007. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *Proc. ICRA*. 1610–1616.

[13] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. 2016. Multi-agent path finding with kinematic constraints. In *Proc. ICAPS*. 477–485.

[14] Frédéric Large, Dizan Vasquez, Thierry Fraichard, and Christian Laugier. 2004. Avoiding cars and pedestrians using velocity obstacles and motion prediction. In *Proc. IV*. 375–379.

[15] Juntong Lin, Xuyun Yang, Peiwei Zheng, and Hui Cheng. 2019. End-to-end decentralized multi-robot navigation in unknown complex environments via deep reinforcement learning. In *Proc. ICMA*. 2493–2500.

[16] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proc. AAMAS*. 1152–1160.

[17] Giacomo Lodigiani, Nicola Basilico, and Francesco Amigoni. 2023. Robust Multi-Agent Pickup and Delivery with Delays. In *Proc. ECMR*. 1–8.

[18] Ryan Luna and Kostas Bekris. 2011. Push and swap: Fast cooperative path-finding with completeness guarantees. In *Proc. IJCAI*. 294–300.

[19] Hang Ma. 2022. Graph-Based Multi-Robot Path Finding and Planning. *Current Robotics Reports* 3, 3 (2022), 77–84.

[20] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proc. AAMAS*. 837–845.

[21] Run Mao, Hongli Gao, and Liang Guo. 2020. A novel collision-free navigation approach for multiple nonholonomic robots based on orca and linear mpc. *Mathematical Problems in Engineering* 2020 (2020), 1–16.

[22] Yuki Miyashita, Tomoki Yamauchi, and Toshiharu Sugawara. 2023. Distributed Planning with Asynchronous Execution with Local Navigation for Multi-agent Pickup and Delivery Problem. In *Proc. AAMAS*. 914–922.

[23] Robert Morris, Corina Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. 2016. Planning, scheduling and monitoring for airport surface operations. In *Proc. AAAI Workshop on Planning for Hybrid Systems*. 608–614.

[24] Zhenbang Nie, Peng Zeng, and Haibin Yu. 2020. Effective decoupled planning for continuous multi-agent pickup and delivery. In *Proc. CCDC*. 2667–2672.

[25] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence* 310 (2022), 1–22.

[26] Erick Rodríguez-Seda and Dušan Stipanović. 2019. Cooperative avoidance control with velocity-based detection regions. *IEEE Control Systems Letters* 4, 2 (2019), 432–437.

[27] David Silver. 2005. Cooperative pathfinding. In *Proc. AIIDE*. 117–122.

[28] Jur Van Den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In *Proc. ISRR*. 3–19.

[29] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. Cobots: Robust symbiotic autonomous mobile service robots. In *Proc. IJCAI*. 4423–4429.

[30] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29 (2008), 9–20.

[31] Bin Xu, Daniel Stilwell, and Andrew Kurdila. 2010. A receding horizon controller for motion planning in the presence of moving obstacles. In *Proc. ICRA*. 974–980.

[32] Tomoki Yamauchi, Yuki Miyashita, and Toshiharu Sugawara. 2022. Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks. In *Proc. AAMAS*. 1427–1435.

[33] Qiuming Zhu. 1991. Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation. *IEEE Transactions on Robotics and Automation* 7, 3 (1991), 390–397.