Privacy Preserving Multi Agent Path Finding

Rotem Lev Lehman
Ben Gurion University of the Negev
Be'er Sheva, Israel
levlerot@post.bgu.ac.il

Roni Stern
Ben Gurion University of the Negev
Be'er Sheva, Israel
sternron@bgu.ac.il

Guy Shani
Ben Gurion University of the Negev
Be'er Sheva, Israel
shanigu@bgu.ac.il

arXiv:2605.14119v1 [cs.MA] 13 May 2026

ABSTRACT

In the multi-agent path finding (MAPF) problem, a group of agents search in a graph for a path for each agent where no two paths collide. While in all applications of MAPF the agents must not collide with each other, in some of them the agents may not wish to share their paths due to privacy constraints. In this work, we formulate two types of privacy constraints for MAPF and propose algorithms that preserve them. The first type of privacy we consider is *planning-level privacy*, which means that during planning, the agents cannot identify exactly the planned location of the other agents. We propose a general framework for obtaining planning-level privacy, which works by adding *mock agents* to the planning process. The second type of privacy we consider is *execution-level privacy*, which is relevant when agents have limited sensing capabilities. Execution-level privacy is preserved if none of the agents is allowed to sense the location of the other agents during execution. We show how to adapt two popular MAPF algorithms, namely PIBT and LaCAM, such that they preserve execution-level privacy. Lastly, we propose a post-processing technique that allows the agents to reduce the sum of costs of the returned solution without losing any privacy. We also implemented our algorithms and evaluated them empirically, showing that the proposed post-processing technique indeed improved cost significantly.

KEYWORDS

Multi Agent Path Finding, MAPF, Privacy

ACM Reference Format:
Rotem Lev Lehman, Roni Stern, and Guy Shani. 2026. Privacy Preserving Multi Agent Path Finding. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 10 pages. https://doi.org/10.65109/

1 INTRODUCTION

The Multi-Agent Path Finding (MAPF) problem arises when multiple mobile agents must each find a path from their respective start locations to their goal locations on a shared graph. The primary requirement is that these paths are collision-free, meaning that no two agents occupy the same location or swap locations simultaneously. MAPF is motivated by a wide range of real-world applications, such as automated warehouse robotics, airport ground traffic management, and digital entertainment, where efficient and safe coordination of multiple agents is essential.

Most work on MAPF has assumed centralized control and complete information sharing between the agents. This is suitable for many use cases, such as warehouse robots controlled by a single operating company. In this work, we consider a different setting, where the agents are controlled by different entities that must collaborate to avoid collisions and optimize a shared objective function, yet still wish to keep some information private from each other. A primary motivation for such a setting is *compartmentalization* due to security concerns. For example, consider an emergency response scenario in a large city, where multiple organizations—such as fire departments, police, and private security firms—deploy their own autonomous vehicles or drones to respond to incidents. While these organizations must coordinate to avoid traffic congestion and ensure rapid response, they may not wish to reveal the exact locations, routes, or priorities of their units due to operational security or privacy regulations. More use-cases for privacy-preserving MAPF include coordinated trucks and drones of different logistics companies, coordinating routes of human taxi drivers, and coordinating the movement of robots in shared environments. In this context, MAPF algorithms must enable safe and efficient multi-agent coordination while preserving the confidentiality of each organization's sensitive information.

This form of privacy-preserving collaborative multi-agent planning has been studied in the context of other types of multi-agent planning problems, including Multi-Agent STRIPS [2, 12, 14, 16], Distributed Constraints Optimization (DCOP) [4, 6] and Distributed Constraint Satisfaction Problem (DisCSP) [29, 30]. In this work we explore different types of privacy requirements one may consider in the context of MAPF and how one can achieve them.

We distinguish between two types of privacy in MAPF: *planning-level privacy* and *execution-level privacy*. Planning-level privacy means that the agents cannot infer from the information they share during planning process any location and time any of the other agents are planning to visit. Execution-level privacy means that even if the agents are equipped with some sensing capabilities, they would still be unable to infer the location of the other agents during execution. Formally defining these types of privacy is the first contribution of this work.

To obtain planning-level privacy, we propose a general framework that works by adding *mock agents* to the planning process. Each real agent is associated with several mock agents, each has a fictitious start and goal location. Any MAPF algorithm can then be used to plan for all agents, including both real and mock agents. We discuss the type of strong privacy one obtains with this mechanism and its weaknesses. Then, we show how one can obtain execution-level privacy in two popular MAPF algorithms, namely PIBT [20] and LaCAM* [19], given prior knowledge about the sensing capabilities of the other agents.

To preserve privacy, the real agent cannot directly manipulate the planning process to optimize its own cost over the costs of the paths of the mock agents, as this may leak information about its

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/

true path. However, we show that the cost of the real agent can be improved in a post-processing step that does not affect the privacy guaranties. In this post-processing step, we identify *safe zones* in the plan, which are locations and time steps where the real agent can deviate from its planned path without risking a collision or detection with other agents. We then use these safe zones to re-plan locally for the real agent, improving its cost while maintaining the overall privacy guaranties.

Finally, we implemented our privacy-preserving algorithms and evaluated them empirically, showing the relation between the amount of privacy we aim for and overall efficiency. Also, we show when our post-processing can be very effective and when it does not add significant gains. Overall, this work paves the way for future research on privacy-preserving MAPF, highlighting the trade-offs between privacy, efficiency, and solution quality in MAPF.

2 BACKGROUND AND PROBLEM SETUP

A classical MAPF problem with N agents is defined by a tuple $\langle G, s, t\rangle$ where $G = (V, E)$ is an undirected graph whose vertices are the possible locations agents may occupy, and every edge $(v, u) \in E$ represents that an agent can move from v to u without passing through any other vertex. The functions s and t map each agent to its initial and desired destination locations, respectively. Time is discretized into time steps. In every time step, each agent can either *wait* in its current location or *move* to a location adjacent to its current location. A *single-agent plan* for an agent i is a sequence of actions (wait or move) that if performed starting from $s(i)$ will end up in $t(i)$. A *joint plan* is a set of single-agent plans, one for each agent. For a joint plan Π, we denote by Π_i its constituent single-agent plan for agent i, and denote by $\Pi_i(t)$ as the vertex agent i is planned to occupy at timestep t according to Π_i.

A pair of agents i and j have a *vertex conflict* in a joint plan Π if, according to their respective single-agent plans Π_i and Π_j, both agents are planned to occupy the same vertex at the same time. Similarly, agents have a *swapping conflict* in a joint plan if they are planned to swap locations over the same edge at the same time. A *valid* solution to a MAPF problem is a joint plan without any conflict. Several solution cost functions have been proposed for MAPF. The two most common are sum of costs (SOC) and makespan, which are the sum and max, respectively, over the lengths of the single-agent plans in the solution. Finding cost-optimal solution for either cost function is NP-Hard [25, 31], and in directed graphs even finding a solution is NP-Hard [15].

Nevertheless, many algorithms have been proposed for solving classical MAPF [5, 26]. A MAPF algorithm is *complete* if it will eventually find a valid solution if one exists, and is *optimal* if the found solution minimizes the cost function. Some MAPF planners are complete and optimal, such as CBS [23], others only complete, such as LaCAM [18], and others neither complete nor optimal yet are known to be very fast, such as PIBT [20]. Anytime MAPF algorithms, such as LaCAM* [19], quickly find an initial solution and then continuously improve it as long as additional runtime is available. In this work, we build on PIBT and LaCAM* and therefore describe them briefly below.

PIBT [20] is a sequential *configuration* generator, where a configuration here is an N-sized vector representing the location of each agent. In each time step, PIBT accepts a configuration of the agents Q^{from} and generates a new configuration Q^{to} corresponding to a possible valid move of all agents. An initial priority is set at each timestep for all of the agents based on their distance to their goal, where the closest agent that hasn't already reached its goal gains the highest priority. In order to determine Q^{to}, PIBT sequentially assigns a vertex to each agent while avoiding collisions. The order of assignment depends on the priority given to each agent, but before assigning a vertex v to agent a_i, PIBT first checks if $\exists a_j \neq a_i : Q^{from}[a_j] = v$. If such a_j exists, then it must apply that $Q^{to}[a_j] \neq v$ because of the collision avoidance. If $Q^{to}[a_j]$ was not assigned yet, PIBT is run on a_j to try and move it from v, and a_j gains the priority of a_i to enable it to move lower priority agents in order to clear the way for a_i. If it was unable to move to another location, PIBT backtracks and tries to assign the next best vertex from the neighbors of $Q^{from}[a_i]$ for a_i.

LaCAM* [19] is a two level-search algorithm, where each node in the high level search holds the configuration and a *constraint tree*, and search throughout the nodes in a depth-first manner. The low-level search gradually grows the constraint tree, and picks the next constraint node to explore, using a breadth-first manner. Then, after picking both high-level and low-level nodes, LaCAM* generates a new configuration using a *configuration generator*. The configuration generator must satisfy the constraints from the low-level node. The authors of LaCAM* chose to use PIBT as a configuration generator, and so we also use it as such in our work.

In this work we also build on DisCSP [29, 30], and so describe it briefly together with privacy preserving solvers.

A DisCSP is a tuple $\langle A, X, D, C, \alpha\rangle$, where

- $A = \{a_1, a_2, \ldots, a_m\}$ is a finite set of autonomous agents.
- $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables.
- $D = \{D_1, D_2, \ldots, D_n\}$ where each D_i is a finite domain of possible values for variable x_i.
- $C = \{c_1, c_2, \ldots, c_k\}$ where each constraint c_j is defined over a subset of variables and specifies the set of allowed tuples for those variables. Constraints can be defined over variables owned by the same agent or by different agents.
- $\alpha : X \rightarrow A$. The function α assigns each variable to exactly one agent, meaning (1) Agent $\alpha(x_i)$ is the owner of variable x_i. (2) Each agent controls the value assignment of its own variables.

The work in [30] proposes a secure privacy-preserving DisCSP algorithm that: (1) Uses public-key cryptography and multiple cooperating servers to perform the search process on encrypted information. (2) Ensures the distributed search (similar to chronological backtracking) does not leak private information of any agent. (3) Guarantees that neither other agents nor the servers can infer additional details about private variable values beyond the final agreed solution. In other words, agents can collaboratively solve the constraint problem and reach agreement on a solution while keeping each agent's private information completely confidential.

Problem setup and assumptions. In this work, we consider a group of agents faced with a classical MAPF problem $\langle G, s, t\rangle$. That is, each agent i is initially located in an initial location $s(i)$ and aims to reach a target location $t(i)$ while avoiding collisions with the other agents. The environment is static and the agents' actions are deterministic.

Each agent is fully aware of the underlying graph G, yet it does not know the initial and target location of the other agents. To coordinate, the agents may send messages to each other directly and immediately. Thus, without any privacy considerations the agents could solve the problem by sending their initial and target locations to one of the agents; have that agent reconstruct the MAPF problem and use any MAPF algorithm to solve it; share the solution with all agents; and finally have all agents safely execute it. The fundamental challenge we focus on in this work is that each agent does not wish the agents to know where it plans to visit in every timestep on its way to its target. That is, for a MAPF solution Π, agent i, and time step t, the *private information* agent i does not wish to disclose is the location $\Pi_i(t)$. Next, we consider two stages where such information can be revealed or inferred — during planning and during execution — and propose methods to preserve privacy in these stages.

Our setting somewhat similar to prior work on distributed methods for solving MAPF [8, 9, 22, 27, 28]. One such approach [3, 9] has each agent plan independently, coordinating and replanning online with other agents that enter its field of view. In that scheme, and in general in existing decentralized MAPF algorithms, agents may know the partial path of other agents if they are close to each other during the runtime. Thus, they do not provide any privacy guarantees, which is our primary objective.

3 PLANNING-LEVEL PRIVACY

During planning, agents may be able to infer some knowledge about the private information of other agents by analyzing the messages sent by each agent. We denote by μ the set of messages passed between the agents during the planning process. The *agents' belief* about agent i at time t, denote $b_i(\mu, t)$, is the set of locations agent i may occupy at time t according to μ. In other words, the agents' belief about agent i captures the uncertainty of the other agents regarding the true location of i at timestamp t. The *belief state* of the agents is the vector $b = (b_1, \ldots, b_N)$ containing all the agents' beliefs.

Definition 1 (k-Privacy). *A belief state b is said to preserve* k-Privacy *if for every agent i and time step t, it holds that $|b_i^t(\mu)| \geq k$.*

Intuitively, this means that for every agent i and time step t, the messages sent during planning are not sufficient to allow the other agents to narrow down the possible location of agent i to fewer than k possible locations. Next, we consider the problem of finding a solution to a given MAPF problem while preserving k-privacy during the planning process. We call this problem the k-Privacy Preserving MAPF (kPPMAPF) problem. Note that MAPF is a special case of a kPPMAPF problem, where $k = 1$.

3.1 k-Privacy Preserving MAPF Planner

Now we describe the k-Privacy Preserving MAPF Planner (kPP), which is a general algorithm for solving the kPPMAPF problem. kPP works by having each agent i create a set of $k-1$ *mock agents*, each associated with a unique pair of initial and target locations. Then, the agents collaboratively solve a larger MAPF problem that includes non-conflicting single-agent plans for both real and mock agents. Each (real) agent then follows the single-agent plan created for it, discarding the plans created for the mock agents. Next, we describe kPP in more details.

In kPP, the agents start by designating randomly one of the agents to be the *planning agent*. Then, each agent i performs the steps described in Algorithm 1. First, it creates a set of k pairs of initial and target locations, denoted $AgGroup_i$. This set includes the pair $(s(i), t(i))$, i.e., the agent's real initial and target locations, as well as $k-1$ additional pairs. These pairs must be unique, i.e., different from $(s(i), t(i))$, from each other, and from the pairs used by the other (real) agents. Then, the agent shuffles these pairs randomly and shares them with all other agents (line 5 in Alg. 1). Next, agent i waits until all agents have broadcast will receive the agents groups of all agents. At this point, if i is the designated planning agent, then it creates a MAPF problem with the initial and target locations in all agent groups (a total of $k \cdot N$ "agents"). Then, it calls any off-the-shelf MAPF solver to solve this MAPF problem, and broadcasts the solution to all the agents. If i is not the planning agent, then it waits for the solution to be broadcasted by the planning agent. Finally, agent i extracts from the solution created by the planning agent only the single-agent plan starting from $s(i)$ and ending in $t(i)$ (line 12).

Figure 1a(left) illustrates a kPPMAPF problem with two agents and the solution created for it by kPP for $k = 2$. The mock agents are marked in green and orange and the dashed lines mark the plans created for them.

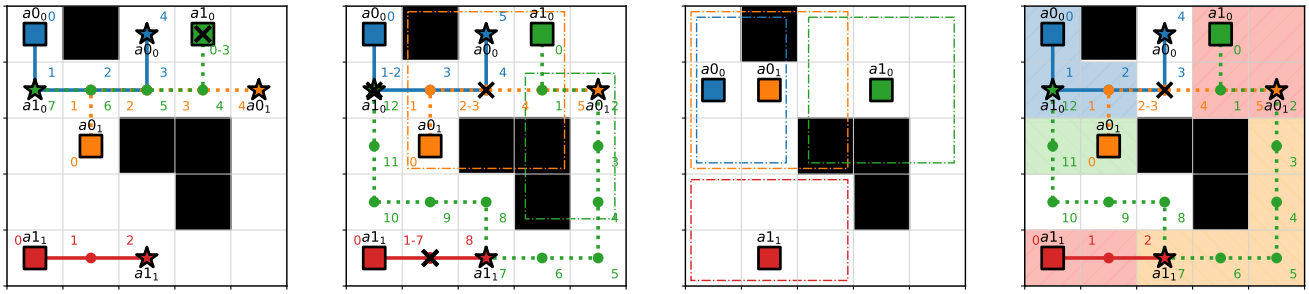
3.2 Choosing the Mock Agents

A critical step in kPP is how each agent chooses the initial and target locations of its $k-1$ mock agents (line 3).

We consider an assignment of mock agents to have a collision between two agents i, j, if there exists a tuple (real or mock) $\langle s_{i_m}, g_{i_m} \rangle$ in the group of locations of agent i, and a tuple (real or mock) $\langle s_{j_n}, g_{j_n} \rangle$ in the group of agent j, where $s_{i_m} = s_{j_n} \vee g_{i_m} = g_{j_n}$. Note that if any collision is found between two assignments, we cannot find a valid plan for the overall kPPMAPF. Also note that if two agents collide in their published groups of locations, and one of them (i) changes its previously published set S by replacing a subset of locations $L \subset S$ to be another set of size $|L|$, then all other agents can understand that its real tuple $\langle s(i), g(i) \rangle \in S \setminus L$. This reduces i's privacy to $k - |L|$ privacy instead of the wanted k privacy. So,

Algorithm 1: kPP for agent i

```
1  AgGroup_i ← (s(i), t(i))
2  for j = 1 to k do
3      s(i_j), t(i_j) ← choose unique initial and target locations
4      Add (s(i_j), t(i_j)) to AgGroup_i
5  Shuffle AgGroup_i and broadcast it to all agents
6  Wait for all agents to broadcast their agents group
7  if i is the designated planning agent then
8      Π ← solve for all agent groups
9      Broadcast Π to all agents
10 else
11     Π ← Wait for a solution from the planning agent
12 Extract single-agent plan for (s(i), t(i))
```



(a) κPPMAPF problem and the solution created for it by κPP.

(b) εκPPMAPF problem. Dashed squares represent the FoV of a_{0_1}, a_{1_0} at time step 3 of the plan.

(c) FoV of each agent at time step 1 of the plan from figure 1b.

(d) PPFPP solution with initial and extended safe zones at time step 2 of the FPP plan.

Figure 1: Examples of κPPMAPF (1a), εκPPMAPF (1b), FoV (1c), and of PPFPP (1d) with 2 agents (a_0 and a_1) and $k = 2$. Squares represents the initial location of each agent and stars represent the desired destination. The real agent of each group has a full line in the path and the mock agent has a dotted line. An X on the path of an agent marks that the agent waits in place in that spot for at least 1 timestep. The timestep of vertices in the plan appears near the vertex. In figure 1d the blue and red vertices correspond to the initial safe zones for a_0 and a_1, respectively, while green and orange correspond to the extended safe zones.

in order to keep the needed k privacy criteria, the algorithm used to pick mock agents need to publish the actual groups of locations only once, and to not have collisions. We will now consider different options for selecting mock agents and discuss their pros and cons.

3.2.1 Choosing Mock Agents Randomly. One approach to do so is to have each agent choose its group of locations randomly. A limitation for this approach is that two agents may collide in their assignments, as seen above. We will now calculate the probability of such collision to occur. The probability of no collision in the assignment of mock agents for a specific pair of agents in a graph with $|V|$ vertices, and a desire for k-privacy, when each agent chooses mock agents randomly is calculated in the following way: Each agent i has a predefined real tuple $\langle s(i), g(i)\rangle$, which do not collide with other agents real tuple. Then each agent chooses $k-1$ mock tuple, i.e. $2(k-1)$ vertices of the $k-1$ mock tuples from the remaining $|V|-1$ vertices, resulting in the following probability: $\frac{\binom{|V|-1-2k}{2(k-1)}}{\binom{|V|-1}{2(k-1)}}$. Therefore, The probability of no collisions in the assignment of N agents is approximately:

$$\left(\frac{\binom{|V|-1-2k}{2(k-1)}}{\binom{|V|-1}{2(k-1)}}\right)^{\binom{N}{2}} \xrightarrow{N\to\infty,k>1} 0$$

As can be seen, this approach can lead to collisions with higher probability as the number of agents planning grows. Next, we explain some other methods for choosing mock agents.

3.2.2 Choosing Mock Agents using a DisCSP. We suggest another approach for choosing the mock agents in a way that will both not collide with other agents and will preserve the privacy of each agent by not revealing the real (start, goal) tuple of each agent. We can model the problem of choosing the mock agents in a privacy-preserving manner as a DisCSP. Each agent will have $2(k-1)$ variables, representing the start and goal of its mock agents, and two additional variables for the real start and goal. The domain of each variable is all possible locations in the graph (V). The public constraints are for agents to avoid conflicts. The private constraints of each agent are to ensure that its real start and goal are assigned to it. Solving this DisCSP will ensure that every agent will have its real start and goal in its k-sized group, and there will be no conflicts, while making sure that this assignment does not reveal any private information.

The wanted privacy model here is to hide the private constraints of the agents, and make sure other agents cannot distinguish between the real tuple of each agent and its mock tuples. This can be solved securely using the method in [30], since they make the search process in remote servers that do not know the values of the variables since they are encrypted, and the agents do not know any knowledge about the failed assignments that happens during the search process (which could reveal the collision between two agents groups), since they are not part of the search and only receive the final assignments for their groups. Hence, we can conclude that no agent can distinguish the other agents real start and goal vertices from their set of disclosed locations.

Solving this DisCSP problem can be very expensive in large graphs, with many agents and large values of k, so one can think of using a different approach that we explain now.

3.2.3 Mock Agents Dispatcher. We suggest yet another approach of choosing the mock agents, by using an *external dispatcher* agent whose sole role is to centralize the process of generating a unique initial and target location pairs when requested to do so by an agent (line 3). This external dispatcher agent knows the real initial and target locations of all agents and makes sure they do not collide.

This can be viewed as some loss of privacy. However, it is external to the path planning agents, and so it does not know the plans the agents end up selecting. Thus, unless the dispatcher collaborates with one of the agents, the amount of privacy lost is limited to the dispatcher knowing the start and goal locations of the agents.

In order to motivate the use of the external dispatcher, consider a different problem setting, where the dispatcher maintains a large

pool of tasks and assigns k candidate tasks to each agent, without knowing which of the k tasks the agent will actually commit to. This provides a natural way to generate indistinguishable alternatives without requiring plan-level information.

The kPP algorithm's behavior is independent of the mock agents selection method in use, and one can use whichever method it prefers. We chose to implement our experiments in section 6 using the external dispatcher method because of its simplicity.

3.3 Theoretical Analysis of kPP

Due to space constraints, we prove informally, the main theoretical property of kPP, namely, that it outputs valid solutions that preserves k privacy.

Theorem 1 (kPP Solves kPPMAPF problems). *If the designated planning agent in kPP returns a valid MAPF solution, then kPP returns a valid kPPMAPF solution.*

Proof outline: A valid MAPF solution ensures there are no conflicts between all agents. This includes conflicts between the real agents and also conflicts between a real agent and the other (mock) agents in its agent group. This means that at any point in time the agents in an agent group occupy k different locations. All messages sent between the agents are agnostic to who is the main agent and who is the mock agents within an agent group. Thus, the belief state will be the same regardless of who is the real agent. Thus, a conflict-free solution has been found and k-privacy preserved.

Unfortunately, kPP is not complete, since there could be a randomized agent group initial locations and targets that would make the original MAPF problem solvable but the new problem unsolvable. Similarly, kPP is not optimal even if the designated planning agent uses an optimal MAPF algorithm.

4 EXECUTION-LEVEL PRIVACY

The agents in MAPF may be equipped with sensors that allow them to sense the location of nearby agents. In such cases, the agents risk revealing their locations during execution, even if they execute a MAPF solution generated by a planning-level privacy preserving MAPF algorithm. Next, we formally describe the problem that arises in these cases and discuss how to ensure privacy is still preserved, even during execution.

4.1 Conflicting Field of Views

We formalize the sensing capability of an agent i by a *Field of View* (FoV) function F_i that maps every possibly location v of agent i to the locations agent i can sense when situated in v. If an agent can sense the location of another agent during execution, i.e., it enters if FoV, then that agent's privacy has been compromised. We refer to this as as FoV conflict and define it formally as follows.

Definition 2 (FoV Conflict). *A pair of single agent plans Π_i and Π_j assigned to agents i and j, respectively, are said to have a FoV conflict if there exists t such that either $\Pi_i(t) \in FoV_j(\Pi_j(t))$ or $\Pi_j(t) \in FoV_i(\Pi_i(t))$.*

An ePPMAPF problem is defined by a tuple $\langle P^*, F \rangle$ where P^* is a classical MAPF problem, and $\{F_1, \ldots, F_N\}$ is the agents' FoV functions. A solution Π for an ePPMAPF problem is called valid if it is a valid solution for the underlying MAPF problem (P^*) and it has no FoV conflict. It is hard to motivate preserving privacy during execution but not during planning. Thus, for the rest of this work we require a valid ePPMAPF solution to also preserve k-privacy.

Definition 3 (Runtime k-Privacy). *An ePPMAPF solution Π preserves* Runtime k-Privacy *if the belief state generated from finding Π preserves k-Privacy and there are no FoV conflicts in Π.*

The ekPPMAPF problem is defined as the problem of finding a runtime k-privacy solution for a given MAPF problem, desired planning-level privacy k, and FoV functions.

Definition 3 is important, since it makes the runtime stage, where agents are aware of the joint plan for all agents, k-privacy preserving, since if the agent a_i saw a_j at vertex v during the runtime, at time t, it ruins the entire belief state of a_j not only for b_j^t, but also for all $b_j^{t'} | t' \neq t$, since a_i can now understand in hindsight that the path related to the v that a_j is traversing in the joint plan, is its real path. In a different case, if agent a_i traverses near vertex v at time t where $v \in b_j^t(\Pi)$ but a_i can see that a_j is not in v at time t, then a_i can reduce the belief state of a_j to be $b_j^{t'} \leftarrow b_j^t \setminus \{v\}$, and, again, it can do so for the entire path that is related to v in the joint path, and by doing so, it reduces the privacy of a_j from $k - Privacy$ to $(k-1) - Privacy$. This is why it is important to preserve $Runtime\ k - Privacy$ when there are sensors in the agents and $k - Privacy$ is needed.

Figure 1b illustrates an example of a ekPPMAPF problem and a valid solution for it for $N = 2, k = 2, \forall v : F_0(v) = F_1(v) = \{u || u.x - v.x| \leq 1 \ \ \& \ \ |u.y - v.y| \leq 1\}$. As can be seen, The plan from figure 1a is not valid for the ekPPMAPF problem, since the paths in that plan has FoV conflict. As a result a different path for a_{1_0} is chosen, but in timestep 2, agent a_{0_1} must wait-in-place, since otherwise it will collide with agent a_{1_0}. As can be seen, the FoV of the two agents do not contain the other agent, and so the plan is now valid. Figure 1c shows the FoV of the agents at time step 1. Notice that the FoV of agents a_{0_0} and a_{0_1} contain the other agent, but it is fine since they are both in the same agent-group.

Notice that kPPMAPF problem is a sub-problem of an ekPPMAPF problem, where $\forall a_i \forall v \ F_i(v) = \{v\}$.

A straightforward ekPPMAPF planner can be obtained by extending the kPP, resulting in the Field-of-View k-Privacy Preserving Planner (fPP). fPP follows the standard kPP framework but modifies the underlying MAPF planner to incorporate additional FoV conflicts that prevent agents from entering each other's fields of view. These FoV conflicts are enforced only between different agent groups, and not between sub-agents within the same group, since sub-agents represent hypothetical alternatives of a single real agent. Consequently, visibility among sub-agents of the same group does not compromise privacy during execution.

Note that when selecting the agent groups in line 3 of algorithm 1, as seen in section 3.2, we need to consider a broader collision definition, which also takes into account that in each tuple of start, goal vertices the vertices are not in the FoV of other agents tuples.

4.2 Theoretical Analysis of fPP

By construction, fPP preserves runtime k privacy. As discussed in section 3.3, kPP is neither complete not optimal, and so fPP is also neither complete nor optimal. Section 5 proposes a post-processing plan improvement step that can be performed on top of fPP to reduce the cost of the solution returned without compromising the required runtime k privacy requirement.

4.3 Implementing fPP

fPP requires the *Sol* used by it to avoid FoV conflicts with agents of different agent groups. In order to support it, one can alter available MAPF planners to support the $FoV\ conflict$ avoidance. Of course, there will be no conflict if a_{i_k} and a_{i_q} which are in the same agent group a_i will be in each other's FoV, and so the sub-solver we use should avoid considering this as a conflict. Next, we describe how to do so in two state-of-the-art MAPF planners: *PIBT* [20] and *LaCAM** [17–19].

PIBT is built to have a priority inheritance stage and a backtracking stage, inside a loop. The priority inheritance means that if an agent a_i with higher priority wants to go to a vertex v where an agent with lower priority a_j resides currently, then it will start a PIBT session for a_j in order for it to move from v to some other vertex and let a_i go to v, and in the process move other agents, even with higher priority than a_j, but with lower priority than a_i. If it cannot move to another location then it will backtrack and a_i will have to find a different location to go to. In order to support FoV conflicts resolution, we don't move only the agent a_j from its location, but all agents in the set $\{a_k|\Pi_k(t) \in F_i(v) \vee v \in F_k(\Pi_k(t))\}$ must move to clear the way for a_i, since otherwise there will be a FoV conflict at time t. To do so we run PIBT on all of the set of agents in the field of view of v, and if one of them fail (cannot move from its current location), we backtrack all of the set (since they do not longer have the priority of a_i).

LaCAM* builds on PIBT to generate configuration. Thus, adapting it to support execution-level privacy is straightforward - simply use the adapted PIBT described above. We note that LaCAM* also includes a slight change to PIBT to support a *swap* operation which increases performance. We did not implement this in our work as it requires additional modifications that we leave to future work.

5 IMPROVING SOLUTION QUALITY

A key property of a runtime k-privacy solution is that for every timestep t and agent i, all the locations planned for agent i and associated $k-1$ mock agents will not be in the FoV of any other agent. We refer to such vertices, i.e., the vertices that are guaranteed to not be sensed by any other agents, as the *safe zone* of agent i at timestep t. Each agent can choose a different plan for itself without coordinating with the other agents as long is the new plan only occupies vertices in corresponding safe zones. This can be beneficial, as the agent may now prioritize its single-agent plan over those of its $k-1$ mock agents, potentially obtaining lower SOC for itself, without compromising privacy. Next, we discuss how this can be done.

5.1 Safe zones in ekPPMAPF plans

In this section we define notions to be used in the post process for improving the plans originated from running fPP.

Definition 4 (Agent Group's FoV). *An agent group a_i's FoV in timestep t on a given plan Π generated by fPP on an ekPPMAPF problem P, denoted $FoV_i^t(\Pi)$, is the set $\{v|\exists a_{i_j} v \in F_i(\Pi_j(t))\}$.*

In other words, $FoV_i^t(\Pi)$ is the set of vertices that are in the FoV of any agent in the group of $Mock_i$ in time t using the plan Π.

Definition 5 (Initial Safe-Zone). *The initial safe-zone of an agent group a_i in a specific timestep t on a given plan Π that was generated from running fPP on an ekPPMAPF problem P, denoted $IS_i^t(\Pi)$, is the set $\{v|F_i(v) \subseteq FoV_i^t(\Pi)\}$.*

In other words, $IS_i^t(\Pi)$ is the set of vertices that all of the vertices in their FoV are in the agent group's FoV.

Definition 6 (Separated Safe Zones). *A set of safe-zones $S(\Pi) = \{S_i(\Pi)|\forall i \in A\}$, is called separated, if and only if:*
$\forall t \forall i \in A \forall v \in S_i^t(\Pi) \forall j \neq i \nexists u \in S_j^t(\Pi) : u \in F_i(v) || v \in F_j(u)$.

In other words, A set of safe-zones $S(\Pi)$ is separated if all of the vertices in each $S_i(\Pi)$ are out of the field of view of each other in each timestep.

Definition 7 (Symmetric Field of View function). *A Field of View function F is considered symmetric, if and only if $\forall v \in V \forall u \in F(v) : v \in F(u)$.*

Definition 8 (PathCost). *A single agent i path cost, denoted $PathCost(\Pi_i)$, is the amount of actions performed by agent i in Π_i until it reaches $g(i)$ and stays there.*

Definition 9 (Real Agent SoC). *A real-agent SoC of an ekPPMAPF plan Π, denoted $RSoC(\Pi)$, is calculated as follows: $RSoC(\Pi) \leftarrow \sum_{a_i \in A} PathCost(\Pi[a_i.real\_agent])$.*

In other words, $RSoC(\Pi)$ is the sum of costs of the real agents' paths.

5.2 Post-Processing fPP algorithm

Algorithm 2 is run by a shared planner. It first calculates the *IS*, as defined in definition 5. Then using it, it calculates the *ES*, which is the extended safe zone. It does so by using $a_i.ExtendSafeZone(ES^t)$. $a_i.ExtendSafeZone(ES^t)$ is a function for each agent, that defines the policy of the agent on picking the next vertex to add to its extended safe zone ES_i^t. Then, each agent group a_i can use ES_i to calculate the best single agent plan for its real agent that passes only inside of the ES_i (this is the *FindPathForRealAgent* method in line 7 in Alg. 2). In our implementation, it is done by using *SIPP* [21], since it handles well planning with safe intervals, which in our case where the safe intervals from ES_i.

Definition 10 (Rules for Extending Safe Zone). *The function $a_i.ExtendSafeZone(ES^t)$ is a function that a_i uses for extending the safe zone ES_i^t. It must pick one vertex v to add to ES_i^t. It returns $TRUE$ if found v that was added to ES_i^t, $FALSE$ otherwise, and it must follow the following rules:*

(1) *v is a neighbor of some $u \in ES_i^t$.*
(2) *$\forall a_j, v \notin ES_j^t$.*

(3) $\forall a_j \neq a_i, \nexists u \in ES_j^t : u \in F_i(v) \vee v \in F_j(u)$.
(4) *Using it must preserve the runtime k-Privacy of* μ, Π *after running it.*

For example, a function that qualifies for all of the rules above is the function *ExtendSafeZoneRandomly* defined in algorithm 3. Algorithm 3 can be implemented distributively, by setting the same random seed and the order of agents to pass by in the for loop at line 13 to each agent for ensuring that they all result in the same *ES*.

Figure 1d shows an example for the initial and extended safe zones, and for the plan generated from the extended safe zones using the PPFPP algorithm. As can be seen, the FPP plan in figure 1b is not the same as the PPFPP plan in figure 1d.

For example, a_{0_0} waited at time step 1, in the FPP plan, since otherwise it would have a conflict with a_{0_1} which must wait at time step 2. Although, in the PPFPP we can see that the initial safe zone of a_0 contains the steps needed for a_{0_0} to continue without waiting until it reaches its goal. This reduces the path cost of a_0, which its real agent is a_{0_0}, to 4 instead of 5.

Algorithm 2: Post-Processing (FPP) Solutions (PPFPP)

1 **Input:** Π*: An EKPPMAPF solution*
2 **Output:** Π' *refined for all agent groups*
3 **Function** *PPfPP*:
4 $IS(\Pi) \leftarrow$ initial safe zone as defined in definition 5 ;
5 $ES(\Pi) \leftarrow ExtendSafeZone(IS(\Pi))$;
6 **foreach** $a_i \in A$ **do**
7 $a_i.FindPathForRealAgent(ES_i(\Pi))$;
8 **Function** *ExtendSafeZone(IS)*:
9 ES $\leftarrow$ IS ;
10 **foreach** $t \in T$ **do**
11 **do**
12 $done \leftarrow TRUE$;
13 **foreach** $a_i \in A$ **do**
14 $picked \leftarrow a_i.ExtendSafeZone(ES^t)$;
15 $done \leftarrow done \wedge \neg picked$;
16 **while** $\neg done$;
17 **return** ES ;

Algorithm 3: Extend safe zone randomly.

1 **Function** $a_i.ExtendSafeZoneRandomly(ES^t)$:
2 $X \leftarrow \{v | \exists u \in ES_i^t : (u,v) \in E \vee (v,u) \in E\} \setminus ES_i^t$;
3 **foreach** $a_j \neq a_i$ **do**
4 $X \leftarrow X \setminus ES_j^t \setminus ES_j^{t-1}$;
5 $X \leftarrow X \setminus \{v | \exists u \in ES_j^t : u \in F_i(v) \vee v \in F_j(u)\}$;
6 **if** $|X| > 0$ **then**
7 $v \leftarrow RandomPick(X)$;
8 $ES_i^t \leftarrow ES_i^t \cup \{v\}$;
9 **return** $TRUE$;
10 **return** $FALSE$;

a_{1_1} on the other hand, cannot continue to its goal on time step 2 based on its initial safe zone, since it contains only the vertex it steps on in the FPP plan. But, the extended safe zone does contain its goal at time step 2, and so it can continue to it. This reduces the plan cost of a_1, which its real agent is a_{1_1}, to 2 instead of 8.

5.3 Theoretical Analysis of PPFPP

In this section we will show that the extension of safe zones is safe, that PPFPP returns valid EKPPMAPF plans, and that the resulting plan cost is less or equal to the original plan.

THEOREM 2 (INITIAL SAFE-ZONE SAFENESS). *An initial safe-zone* $IS(\Pi)$ *calculated from definition 5, when* $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i *is symmetric, is safe.*

THEOREM 3 (EXTENDED SAFE-ZONE SAFENESS). *An extended safe-zone* $ES(\Pi)$ *calculated from running ExtendSafeZone in line 8 in algorithm 2 on IS calculated from definition 5, when the function* $a_i.ExtendSafeZone(ES^t)$ *from line 14 follows the rules from definition 10, when* $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i *is symmetric, is safe.*

THEOREM 4 (PPFPP SOLVES EKPPMAPF PROBLEMS). *If ES was calculated using a* $a_i.ExtendSafeZone(ES^t)$ *function that follows the rules from definition 10, and* $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i *is symmetric* & $\forall u \in v.neighbors : u \in F_i(v)$*, then PPFPP returns valid EKPPMAPF plans.*

The proofs for theorems 2, theorem 3, and theorem 4 are pretty straightforward, and hence are attached in the supplementary material of this paper.

Even though KPPMAPF is a sub-problem of EKPPMAPF, we cannot run KPPMAPF plans inside the PPFPP algorithm to improve their plan cost, since in Theorem 4 we must have the neighbors of each vertex inside the FoV function in order for it to return valid plans (to avoid *swapping* conflicts).

THEOREM 5 (PPFPP IMPROVES $RSoC(\Pi)$). *If ES was calculated using a* $a_i.ExtendSafeZone(ES^t)$ *function that follows the rules from definition 10, and* $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i *is symmetric, and* Π' *is the result of running PPFPP on a valid EKPPMAPF plan* Π*, then* $RSoC(\Pi') \leq RSoC(\Pi)$.

The proof of theorem 5 is trivial as the *ES* will always contain the previous plan and SIPP is an optimal algorithm.

6 EXPERIMENTAL RESULTS

In this section, we evaluate our algorithms on a standard MAPF benchmark [24] using the modified PIBT [20] and LaCAM* [19] algorithms as described in Section 4.3.

We ran our experiments 4-neighborhood grids from the standard grid-based MAPF benchmark [24]. Twelve grids were selected from this benchmark with different size and sparseness. The maps chosen to run experiments on, with their corresponding amount of vertices, are: (1) brc202d ($|V|$ = 43,151), (2) lt_gallowstemplar_n ($|V|$ = 10,021), (3) maze-32-32-2 ($|V|$ = 666), (4) orz900d ($|V|$ = 96,603), (5) ost003d ($|V|$ = 13,214), (6) random-32-32-20 ($|V|$ = 819), (7) random-64-64-20 ($|V|$ = 3,270), (8) room-32-32-4 ($|V|$ = 682), (9) room-64-64-16 ($|V|$ = 3,646), (10) room-64-64-8 ($|V|$ = 3,232), (11)

warehouse-20-40-10-2-1 ($|V| = 22{,}599$), and (12) warehouse-20-40-10-2-2 ($|V| = 38{,}756$). In terms of sensing, we assumed an agent can sense nearby agent based on the following parametric and symmetric FoV function:

$$F_i^r(v) = \{u || u.x - v.x| \leq r \ \ \& \ \ |u.y - v.y| \leq r\} \quad (1)$$

6.1 Results

In the first set of experiment, we varied the number of agents $N = 10, 20, \ldots, 50$, desired k-privacy values of $k = 1, 2, 3$, and FoV sizes $r = 1, 2$, and 3. Each setting was run 3 times with different random seeds, and was given 1 minute to finish. The experiments ran on a cluster of computers, each with 20 CPUs and 40GB of RAM, and in parallel with 4 processes running different instances.

Figure 2 shows a cactus chart of the RSoC of the problem against the #solved problems for using both sub solvers LaCAM* and PIBT in FPP with different k values. It was analyzed over the entire experiment results, including all different agent amounts and *FoV r* values. The x axis represents the amount of solved instances, and the y axis represents the RSoC of the solved instance. As expected, increasing k decreases the amount of solved instances, for example, in figure 2c, we saw that using LaCAM* with $k = 1$ solved 56 instances, $k = 2$ solved 37 and $k = 3$ solved 29. In most cases, using LaCAM* as the sub-solver of FPP solved more instances, for example in figure 2a with $k = 2$, we saw that LaCAM* solved 38 while PIBT solved only 28 instances. Notice that the RSoC of the instances that were solved by LaCAM* and not by PIBT (the harder instances) was extremely high. This is due to the fact that LaCAM* is an any-time algorithm that improves the SoC of the found plan when more time is given, so on harder problems it found an initial plan and improved it only a little in comparison to easier problems. This trend can be seen for example in figures 2a and 2c.

Figure 3 shows a cactus chart similar to figure 2, but using different FoV radius values. Increasing the FoV radius drastically impacts both the amount of solved instances and the solution quality. For example in the map maze-32-32-2 in figure 3b the only solved instances were of FoV radius = 0. We also see that in figure 3d it was not so drastically affected from the increase in FoV. We assume it is because of the many narrow corridors within the map that make the instances with FoV radius > 0 have a very little amount of *FoV* conflicts, since they can just follow each other in the corridor in a safe distance from each other and they will be able to pass without a conflict.

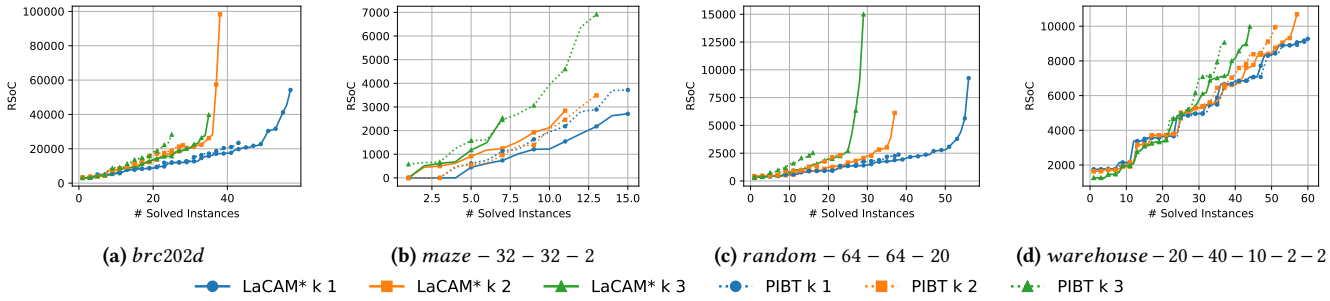
Similar trends were observed in the other maps. We show these results for both figures 2 and 3 in the supplementary material.

In the second set of experiments, we aimed to explore the impact of higher values of k. Thus, we limited the range of values for the other experiment parameters, namely we ran FPP on all maps with $k \in [2, 10]$, $N = 10$, $r = 1$, and using PIBT and LaCAM* as sub-solvers. Every experiment was performed 30 times for each configuration of parameters with different random seeds. For each instance solved by FPP under 1 minute, we ran the PPFPP algorithm with a timeout of 5 minutes. We ran this entire experiment on the cluster, with each map running in a different node. Each node ran with 40 CPUs for FPP and 50 CPUs for PPFPP and 60GB of RAM, in parallel, with 8 processes running different instances and 10 threads running in each process for calculating the *ES* in parallel over different time steps. The maps *orz900d* and *maze-32-32-2* are not shown in the results because of different reasons. *orz900d* does not appear since it is a very large map ($|V| = 96{,}603$) and when running it in PPFPP in our implementation it generates an *out of memory* exception on the node for many problems. *maze-32-32-2* does not appear since no instance of it was solved by FPP when using $r > 0$.

Table 1: Average *RSoC improvement%* $\left(\frac{RSoC(fPP) - RSoC(PPfPP)}{RSoC(fPP)}\right) * 100\%$ (mean ± std, max and median (med)) of PPFPP over FPP, across domains with different k values. #S is the amount of solved instances by FPP (all instances also were also solved by PPFPP). * and ** marks significant improvement of PPFPP over FPP (Wilcoxon, p < 0.05 and p < 0.01 accordingly). FPP MS is the makespan of the original plan created by FPP. Each map shows the amount of vertices in it ($|V|$).

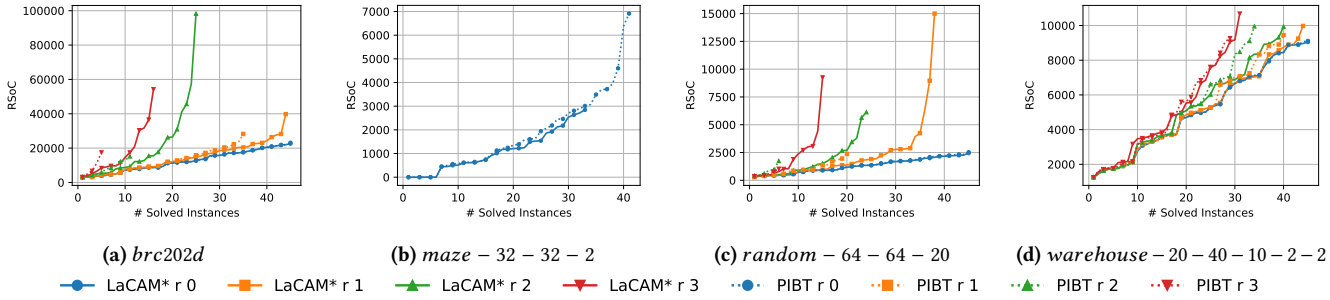
Map	k	*RSoC Improvement%* mean ± std	max	med	#S	time [s]	FPP MS
brc202d $\|V\| = 43{,}151$	2	$0.77 \pm 2.47^{**}$	11.50	0.02	56	113.6	943.9
	3	$0.64 \pm 1.65^{**}$	7.25	0.02	55	116.0	975.4
	4	$1.04 \pm 2.69^{**}$	11.30	0.02	47	121.2	1022.4
	5	$1.06 \pm 2.66^{**}$	11.27	0.00	40	125.3	1041.3
	6	$1.20 \pm 2.68^{**}$	10.72	0.02	39	124.0	1040.9
	7	$1.80 \pm 3.90^{**}$	12.90	0.03	42	131.0	1094.5
	8	$2.04 \pm 4.45^{**}$	16.91	0.07	30	137.6	1170.1
	9	$3.21 \pm 6.40^{**}$	22.83	0.11	25	143.4	1203.6
	10	$1.01 \pm 2.69^{**}$	13.21	0.07	25	143.8	1312.6
random-64-64-20 $\|V\| = 3{,}270$	2	$0.20 \pm 1.05^{**}$	7.75	0.00	56	0.3	90.5
	3	$0.62 \pm 1.89^{**}$	10.34	0.00	48	0.3	96.7
	4	$0.73 \pm 1.84^{**}$	7.65	0.00	38	0.3	100.3
	5	$1.33 \pm 2.60^{**}$	10.00	0.23	29	0.4	106.8
	6	$1.04 \pm 2.30^{**}$	8.33	0.20	21	0.4	120.4
	7	$0.70 \pm 1.24^{**}$	4.85	0.19	19	0.4	146.4
	8	$0.93 \pm 1.88^{**}$	5.72	0.23	15	0.5	151.8
	9	$2.97 \pm 5.36^{**}$	18.32	0.16	12	0.7	244.6
	10	$0.86 \pm 1.55^{*}$	3.64	0.16	5	0.9	294.6
Mean	2	$0.50 \pm 1.61^{**}$	11.50	0.00	518	19.5	293.3
	3	$0.68 \pm 2.22^{**}$	25.74	0.00	440	22.3	336.9
	4	$0.49 \pm 1.55^{**}$	11.30	0.00	360	24.6	374.7
	5	$0.93 \pm 2.35^{**}$	19.11	0.02	289	26.4	408.3
	6	$0.83 \pm 2.25^{**}$	17.06	0.00	244	30.2	451.8
	7	$1.04 \pm 2.86^{**}$	18.33	0.03	202	37.7	507.6
	8	$1.06 \pm 2.78^{**}$	16.91	0.04	176	36.2	518.6
	9	$1.66 \pm 3.90^{**}$	22.83	0.05	137	38.6	548.7
	10	$0.71 \pm 1.93^{**}$	13.21	0.05	119	41.5	630.2

Table 1 shows the results of the experiment for two interesting maps *brc202d*, *random-64-64-20* and the mean over all maps. The table contains results for both MAPF algorithms PIBT and LaCAM* together since there were similar trends for each of them separately. A table like this is shown in the supplementary material for the rest of the maps. In almost all k value for each map, the RSoC improvement is statistically significant. In some cases, such as in $k = 9$ in both maps, the maximal RSoC improvement was very



(a) *brc202d* **(b)** *maze* − 32 − 32 − 2 **(c)** *random* − 64 − 64 − 20 **(d)** *warehouse* − 20 − 40 − 10 − 2 − 2

Figure 2: Cactus chart of *RSoC* for each configuration of k in each sub-solver (*LaCAM*/*PIBT*) over # of solved instances.



(a) *brc202d* **(b)** *maze* − 32 − 32 − 2 **(c)** *random* − 64 − 64 − 20 **(d)** *warehouse* − 20 − 40 − 10 − 2 − 2

Figure 3: Cactus chart of *RSoC* for each configuration of *FoV* in each sub-solver (*LaCAM*/*PIBT*) over # of solved instances.

high (22.83% for *brc202d* and 18.32% for *random-64-64-20*). As the k value grows, the mean improvement grows as well, as can be seen in both maps, and in the mean of all maps, notice that the median value also grows with k, although it is very small. This consists with our hypothesis where larger k values will result in larger RSoC improvement. The runtime of PPFPP depends directly on the map size, where larger maps take longer to generate the *ES*. This can be seen for example in map *brc202d* where $|V| = 43,151$, the runtime is extremely high, in comparison to map *random-64-64-20* where $|V| = 3,270$, and the runtime there is below one second for all k values. The runtime also depends on the makespan of the original plan generated by FPP, where the higher the makespan, the higher the runtime of PPFPP. This is because we need to calculate ES for each time step in the original plan, and go over all of the vertices. It can be seen in both maps and in the mean over all maps.

7 RELATED WORK

In the field of Collaborative Privacy Preserving Planning (CPPP) there has been a lot of work regarding privacy in multi-agent planning scenarios [2, 11–14, 16]. It is a similar field, but privacy preserving MAPF is a special case of the CPPP general field. It is important to study privacy in MAPF as well, since it allows us to handle the MAPF case with better suited algorithms than the general purpose algorithms used in CPPP. In the CPPP field the privacy is measured by not disclosing the variables and actions of the world of each agent to the other agents (or disclose part of them keeping the other part private) - in privacy preserving MAPF— which we properly define in this paper for the first time, the actions are known to all agents (move to any neighbor or stay in place). However, the vertices that each agent is in at any given time is a private knowledge that needs to be kept from other agents while not conflicting with other agents in the same time - which is a hard task to achieve. Brafman [2] suggests a strong type of privacy where a value of a variable is strongly private if the other agents cannot deduce its existence from the information available to them. In this paper we suggest a privacy preserving MAPF problem where we do not seek a form of fully strong privacy on the path of each agent, but we seek that the other agents cannot deduce the location of the agent in a specific time step exactly but from within a minimum of k possible values.

Another well-studied privacy preserving field is the privacy preserving distributed constraint optimization problem (DCOP) [4, 6, 7, 10], where the goal is to solve distributed combinatorial problems in which the variables of the problem are owned by different agents while keeping privacy over the constraints and variables of the different agents. In this paper, we focus on the privacy preserving MAPF problem we suggested, where, unlike privacy preserving DCOP, the goal is to plan paths for a set of agents, instead of optimizing some utility function.

Privacy has rarely been mentioned in the context of MAPF. One exception is recent work on finding all optimal solutions in MAPF [1]. They state that finding all solutions can help keep privacy about the chosen paths. However, the privacy considered there is with respect to an external agent, and not of hiding the paths of agents from each other as we do in this work.

8 CONCLUSION AND FUTURE WORK

This paper introduces a new field in MAPF where privacy is considered. We defined two novel privacy preserving problems, the

kPPMAPF which is a problem where agents must find paths that do not collide, without sharing their real path during planning, and the ekPPMAPF which is an extension of the kPPMAPF with the addition of also avoiding the paths of agents to be disclosed during the execution stage when there are limited sensors on the agents by keeping out of the field of view of other agents during planning. We also provide two novel algorithms to solve the new problems, kPP, which solves the kPPMAPF problem by introducing *mock agents* to confuse other agents during planning and the fPP, which extends the kPP in order to solve the ekPPMAPF problem by also adapting the sub-solver used to keep out of the field of view of agents in different groups.

We also introduce the PPfPP algorithm, which improves the cost of plans generated for the ekPPMAPF problem by defining a concept of *safe-zones* where each agent group can walk freely in and re-plan for the single real agent inside of it.

We also provide extensive theoretical and experimental evaluation of our algorithms, showing they preserve k-privacy and Runtime k-privacy and that the PPfPP algorithm improves the cost of the ekPPMAPF plans resulted by fPP.

In PPfPP, one can further research more complicated functions other than the random function suggested in algorithm 3, that follows the rules from definition 10, for improving the extension of safe zones, and maybe improve the $RSoC(\Pi)$ even further.

Another future research direction is to explore other sub-planners to use in the fPP algorithm, such as optimal MAPF planners, and see whether the PPfPP improves their cost significantly, and makes it closer to the optimal RSoC.

In our work we used a vanilla version of PIBT which does not use the *swap* improvement introduced at [17]. A future research direction could be to support *FoV* in the *swap* operation of PIBT to improve the planning time of fPP while using LaCAM*, and solving more instances while using PIBT.

REFERENCES

[1] Shahar Bardugo, Daniel Koyfman, and Dor Atzmon. 2025. Finding All Optimal Solutions in Multi-Agent Path Finding. In *International Symposium on Combinatorial Search*. 20–28.

[2] Ronen I Brafman. 2015. A privacy preserving algorithm for multi-agent planning and search. In *24th International Joint Conference on Artificial Intelligence, IJCAI 2015*. International Joint Conferences on Artificial Intelligence, 1530–1536.

[3] Stepan Dergachev and Konstantin Yakovlev. 2021. Distributed multi-agent navigation based on reciprocal collision avoidance and locally confined multi-agent path finding. In *IEEE International Conference on Automation Science and Engineering (CASE)*. 1489–1494.

[4] Boi Faltings, Thomas Léauté, and Adrian Petcu. 2008. Privacy guarantees through distributed constraint satisfaction. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 2. IEEE, 350–358.

[5] Ariel Felner, Roni Stern, Solomon Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *International Symposium on Combinatorial Search*, Vol. 8. 29–37.

[6] Rachel Greenstadt, Barbara Grosz, and Michael D Smith. 2007. SSDPOP: improving the privacy of DCOP with secret sharing. In *International joint conference on Autonomous agents and multiagent systems (AAMAS)*. 1–3.

[7] Tal Grinshpoun and Tamir Tassa. 2016. P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research* 57 (2016), 621–660.

[8] Florence Ho, Rúben Geraldes, Artur Gonçalves, Bastien Rigault, Benjamin Sportich, Daisuke Kubo, Marc Cavazza, and Helmut Prendinger. 2020. Decentralized multi-agent path finding for UAV traffic management. *IEEE Transactions on Intelligent Transportation Systems* 23, 2 (2020), 997–1008.

[9] M Onur Keskin, Furkan Cantürk, Cihan Eran, and Reyhan Aydoğan. 2024. Decentralized multi-agent path finding framework and strategies based on automated negotiation. *Autonomous Agents and Multi-Agent Systems* 38, 1 (2024), 10.

[10] Pablo Kogan, Tamir Tassa, and Tal Grinshpoun. 2022. Privacy preserving DCOP solving by mediation. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*. Springer, 487–498.

[11] Rotem Lev Lehman, Guy Shani, and Roni Stern. 2021. Partial disclosure of private dependencies in privacy preserving planning. In *20th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2021*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 1563–1565.

[12] Rotem Lev Lehman, Guy Shani, and Roni Stern. 2022. Reducing disclosed dependencies in privacy preserving planning. *Autonomous Agents and Multi-Agent Systems* 36, 2 (2022), 52.

[13] Shlomi Maliah, Guy Shani, and Roni Stern. 2016. Stronger privacy preserving projections for multi-agent planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 26. 221–229.

[14] Shlomi Maliah, Guy Shani, and Roni Stern. 2017. Collaborative privacy preserving multi-agent planning: Planners and heuristics. *Autonomous agents and multi-agent systems* 31, 3 (2017), 493–530.

[15] Bernhard Nebel. 2024. The computational complexity of multi-agent pathfinding on directed graphs. *Artificial Intelligence* 328 (2024).

[16] Raz Nissim and Ronen Brafman. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research* 51 (2014), 293–332.

[17] Keisuke Okumura. 2023. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*.

[18] Keisuke Okumura. 2023. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

[19] Keisuke Okumura. 2024. Engineering LaCAM*: Towards Real-Time, Large-Scale, and Near-Optimal Multi-Agent Pathfinding. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[20] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2022. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *Artificial Intelligence* (2022), 103752. https://doi.org/10.1016/j.artint.2022.103752

[21] Mike Phillips and Maxim Likhachev. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE international conference on robotics and automation*. IEEE, 5628–5635.

[22] Poom Pianpak, Tran Cao Son, Phoebe O Toups Dugas, and William Yeoh. 2019. A distributed solver for multi-agent path finding problems. In *International Conference on Distributed Artificial Intelligence (DAI)*. 1–7.

[23] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence* 219 (2015), 40–66.

[24] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*. 151–158.

[25] Pavel Surynek. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*.

[26] Pavel Surynek. 2022. Problem Compilation for Multi-Agent Path Finding: a Survey.. In *IJCAI*. 5615–5622.

[27] Prasanna Velagapudi, Katia Sycara, and Paul Scerri. 2010. Decentralized prioritized planning in large multirobot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4603–4609.

[28] Hanlin Wang and Michael Rubenstein. 2020. Walk, stop, count, and swap: decentralized multi-agent path finding with theoretical guarantees. *IEEE Robotics and Automation Letters* 5, 2 (2020), 1119–1126.

[29] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10, 5 (1998), 673–685. https://doi.org/10.1109/69.729707

[30] Makoto Yokoo, Koutarou Suzuki, and Katsutoshi Hirayama. 2002. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 387–401.

[31] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*.

Privacy Preserving Multi Agent Path Finding - Supplementary Material

Rotem Lev Lehman
Ben Gurion University of the Negev
Be'er Sheva, Israel
levlerot@post.bgu.ac.il

Roni Stern
Ben Gurion University of the Negev
Be'er Sheva, Israel
sternron@bgu.ac.il

Guy Shani
Ben Gurion University of the Negev
Be'er Sheva, Israel
shanigu@bgu.ac.il

ACM Reference Format:
Rotem Lev Lehman, Roni Stern, and Guy Shani. 2026. Privacy Preserving Multi Agent Path Finding - Supplementary Material. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 6 pages. https://doi.org/10.65109/

1 THEOREMS AND PROOFS FOR PPFPP

Definition 1 (Initial Safe-Zone). *The initial safe-zone of an agent group a_i in a specific timestep t on a given plan Π that was generated from running FPP on an EKPPMAPF problem P, denoted $IS_i^t(\Pi)$, is the set $\{v|F_i(v) \subseteq FoV_i^t(\Pi)\}$.*

Definition 2 (Agent Group's FoV). *An agent group a_i's FoV in timestep t on a given plan Π generated by FPP on an EKPPMAPF problem P, denoted $FoV_i^t(\Pi)$, is the set $\{v|\exists a_{i_j} v \in F_i(\Pi_j(t))\}$.*

Definition 3 (Runtime k-Privacy). *A MAPF solution Π preserves* Runtime k-Privacy *if the belief state generated from finding Π preserves k-Privacy and there are no FoV conflicts in Π.*

Definition 4 (Separated Safe Zones). *A set of safe-zones $S(\Pi) = \{S_i(\Pi)|\forall i \in A\}$, is called separated, if and only if:*
$\forall t \forall i \in A \forall v \in S_i^t(\Pi) \forall j \neq i \nexists u \in S_j^t(\Pi) : u \in F_i(v) || v \in F_j(u)$.

Definition 5 (Rules for Extending Safe Zone). *The function $a_i.ExtendSafeZone(ES^t)$ is a function that a_i uses for extending the safe zone ES_i^t. It must pick one vertex v to add to ES_i^t. It returns TRUE if found v that was added to ES_i^t, FALSE otherwise, and it must follow the following rules:*

1. *v is a neighbor of some $u \in ES_i^t$.*
2. *$\forall a_j, v \notin ES_j^t$.*
3. *$\forall a_j \neq a_i, \nexists u \in ES_j^t : u \in F_i(v) || v \in F_j(u)$.*
4. *Using it must preserve the runtime k-Privacy of μ, Π after running it.*

Theorem 1 (Initial safe-Zone Safeness). *An initial safe-zone $IS(\Pi)$ calculated from definition 1, when $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric, is safe.*

Proof. Given an EKPPMAPF problem P, a valid plan Π that solves P, and was returned by using FPP on P, an initial safe-zone $IS(\Pi)$ calculated from definition 1, and $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric. Let $a_i, a_j \in A | a_i \neq a_j, t \in T$. Also, let $v \in IS_i^t(\Pi)$. Assume negatively that $\exists u \in IS_j^t : \big(u \in F_i(v) || v \in F_j(u)\big)$, without loss of generality, assume $u \in F_i(v)$. From definition 1, $u \in FoV_i^t(\Pi)$, hence from definition 2, $\exists a_{i_p} : u \in F_i(\Pi_p(t))$, because that F_i is symmetric, $\Rightarrow \Pi_p(t) \in F_i(u)$ and because $F_i = F_j \Rightarrow \Pi_p(t) \in F_j(u)$. Because of $u \in IS_j^t(\Pi)$, then $F_j(u) \subseteq FoV_j^t(\Pi) \Rightarrow \Pi_p(t) \in FoV_j^t(\Pi)$, and so from definition 2, $\exists a_{j_q} : \Pi_p(t) \in F_j(\Pi_q(t))$. But a_{i_p} and a_{j_q} are of different agent groups, and so this is a FoV conflict $\Rightarrow$ from definition 3, Π does *not* preserve Runtime k-Privacy, and so Π is not a valid EKPPMAPF plan, in contradiction to the assumption that Π is a valid plan that solves P. $\Rightarrow \nexists u \in IS_j^t : \big(u \in F_i(v) || v \in F_j(u)\big) \Rightarrow$ from definition 4 $IS(\Pi)$ is considered safe. □

Theorem 2 (Extended Safe-Zone Safeness). *An extended safe-zone $ES(\Pi)$ calculated from running ExtendSafeZone in line 8 in algorithm 1 on IS calculated from definition 1, when the function $a_i.ExtendSafeZone(ES^t)$ from line 14 follows the rules from definition 5, when $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric, is safe.*

Proof. Given an EKPPMAPF problem P, a valid plan Π that solves P, and was returned by using FPP on P, an extended safe-zone $ES(\Pi)$ calculated from running *ExtendSafeZone*, where the function $a_i.ExtendSafeZone(ES^t)$ from line 14 follows the rules from definition 5, and $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric. Let $a_i, a_j \in A | a_i \neq a_j, t \in T$. Also, let $v \in ES_i^t(\Pi), u \in ES_j^t(\Pi)$. There are four possible cases:

Algorithm 1: Post-Processing (FPP) Solutions (PPFPP)

```
1  Input: Π: An EKPPMAPF solution
2  Output: Π′ refined for all agent groups
3  Function PPfPP:
4      IS(Π) ← initial safe zone as defined in definition 1 ;
5      ES(Π) ← ExtendSafeZone(IS(Π)) ;
6      foreach a_i ∈ A do
7          a_i.FindPathForRealAgent(ES_i(Π)) ;
8  Function ExtendSafeZone(IS):
9      ES ← IS ;
10     foreach t ∈ T do
11         do
12             done ← TRUE;
13             foreach a_i ∈ A do
14                 picked ← a_i.ExtendSafeZone(ES^t) ;
15                 done ← done ∧ ¬picked ;
16         while ¬done;
17     return ES ;
```

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/

- $v \in IS_i^t(\Pi)$ & $u \in IS_j^t(\Pi)$: From theorem 1 and definition 4, $u \notin F_i(v)$ & $v \notin F_j(u)$.
- $v \in IS_i^t(\Pi)$ & $u \notin IS_j^t(\Pi)$: So in line 14 in algorithm 1, there was some iteration where u was added to ES_j^t. In this iteration we called $a_i.ExtendSafeZone(ES^t)$ where $IS_i^t \subseteq ES_i^t$, since it was set at line 9 to be IS_i^t, and $u \notin ES_j^t$ since it was added in this iteration to ES_j^t. From definition 5, rule number 3 we can conclude that $u \notin F_i(v)$ & $v \notin F_j(u)$.
- $v \notin IS_i^t(\Pi)$ & $u \in IS_j^t(\Pi)$: Same as the previous case.
- $v \notin IS_i^t(\Pi)$ & $u \notin IS_j^t(\Pi)$: So, without loss of generality, there was an iteration in line 14 in algorithm 1, where v was added to ES_i^t, and u was not yet added to ES_j^t. This iteration is the same as the previous case, and so $u \notin F_i(v)$ & $v \notin F_j(u)$.

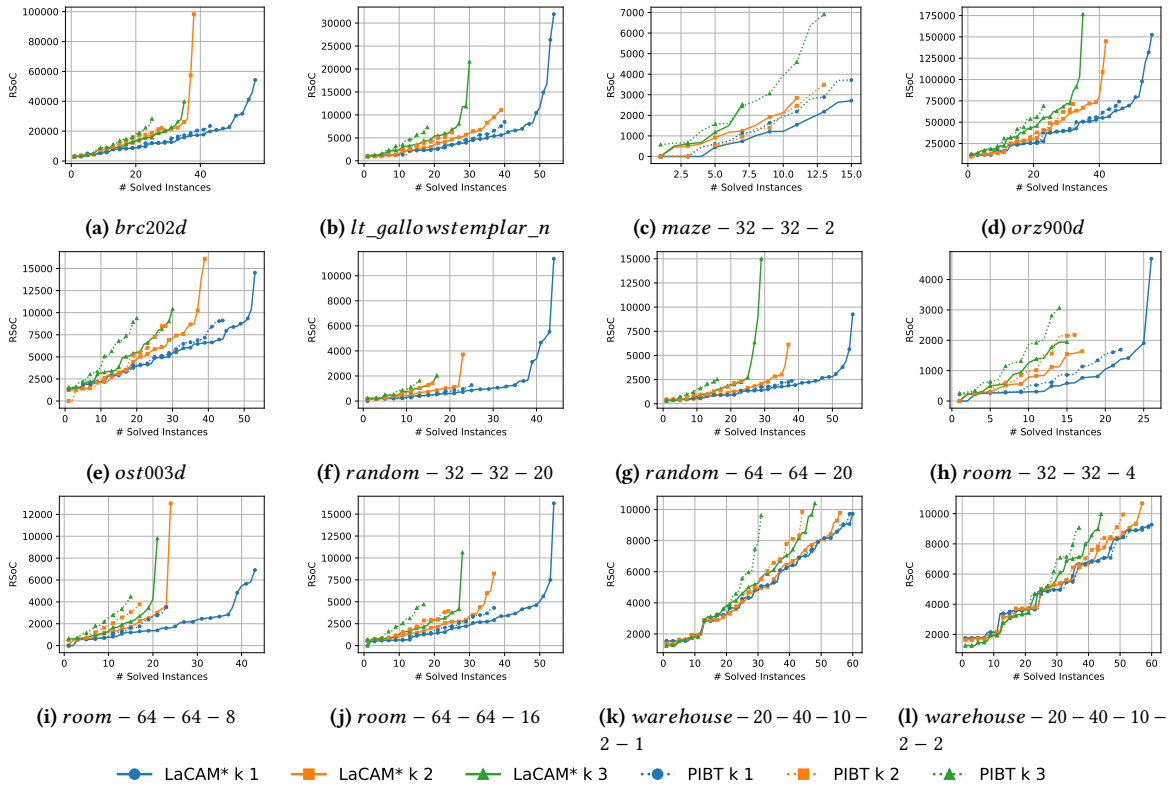
As you can see, in all cases $u \notin F_i(v)$ & $v \notin F_j(u) \Rightarrow$ from definition 4 $ES(\Pi)$ is considered safe. □

Theorem 3 (PPFPP Solves ekPPMAPF Problems). *If ES was calculated using a $a_i.ExtendSafeZone(ES^t)$ function that follows the rules from definition 5, and $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric & $\forall u \in v.neighbors : u \in F_i(v)$, then PPFPP returns valid ekPPMAPF plans.*

Proof. Given that ES in algorithm 1 is calculated using a $a_i.ExtendSafeZone(ES^t)$ function that follows the rules from definition 5, and $\forall a_i, a_j \in A \forall v \in V : F_i(v) = F_j(v)$ & F_i is symmetric & $\forall u \in v.neighbors : u \in F_i(v)$. Also given an ekPPMAPF problem P, a plan Π originated from running the FPP algorithm, and a plan Π' that returned from running PPFPP on Π. From theorem 2 we can conclude that $ES(\Pi)$ is safe. So $\forall a_i \in A$, if we find a plan for $a_i.real\_agent$ that stays only in the safe-zone ES_i, it will not conflict with any other $a_j \neq a_i$ in the FoV. Also, there will be no *vertex* conflicts, because of rule number 2 in definition 5. Let $a_i, a_j \in A : a_i \neq a_j$ assume negatively that $\exists t \in T$: there is a swapping conflict between a_i and a_j in Π' on time steps $t-1, t$. Let v be the vertex a_i is at time $t-1$ on Π' and u be the vertex a_j is at time $t-1$ on Pi'. From the assumptions $(\forall a_i \in A \forall v \in V \forall u \in v.neighbors : u \in F_i(v))$, $\Rightarrow u \in F_i(v)$ & $v \in F_j(u) \Rightarrow$ in time $t-1$ there is a FoV conflict - but we proved (in this proof) that there are no FoV conflicts, so there is a contradiction, $\Rightarrow \nexists t \in T$: there is a swapping conflict between a_i and a_j in Π' on time steps $t-1, t \Rightarrow$ there are no *swapping* conflicts in the resulting plan Π'. So Π' is a valid plan for P. □

2 EXPERIMENTAL RESULTS

In this section we show all experimental results not shown in the main paper. Trends are the same as in the main paper and so are not discussed here.

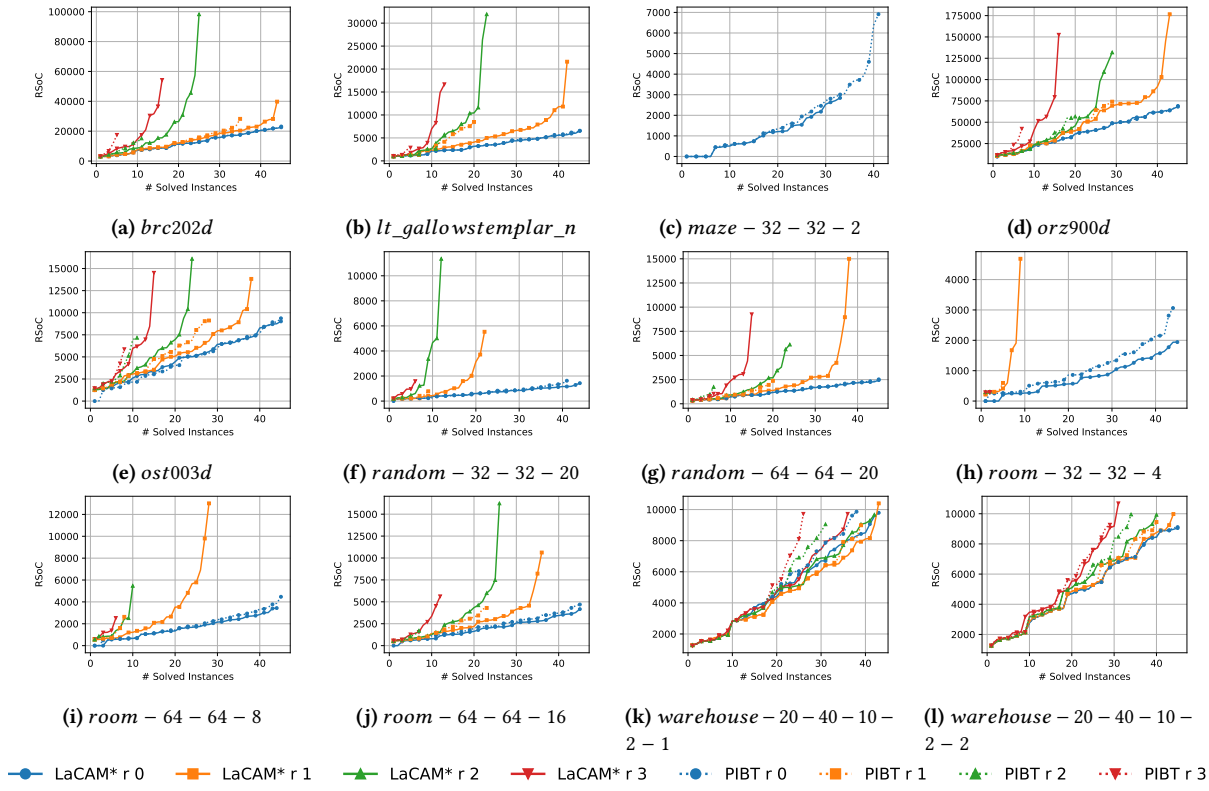


(a) $brc202d$ **(b)** $lt\_gallowstemplar\_n$ **(c)** $maze-32-32-2$ **(d)** $orz900d$

(e) $ost003d$ **(f)** $random-32-32-20$ **(g)** $random-64-64-20$ **(h)** $room-32-32-4$

(i) $room-64-64-8$ **(j)** $room-64-64-16$ **(k)** $warehouse-20-40-10-2-1$ **(l)** $warehouse-20-40-10-2-2$

Figure 1: Cactus chart of $RSoC$ for each configuration of k in each sub-solver ($LaCAM/PIBT$) over # of solved instances.



(a) $brc202d$ **(b)** $lt\_gallowstemplar\_n$ **(c)** $maze-32-32-2$ **(d)** $orz900d$

(e) $ost003d$ **(f)** $random-32-32-20$ **(g)** $random-64-64-20$ **(h)** $room-32-32-4$

(i) $room-64-64-8$ **(j)** $room-64-64-16$ **(k)** $warehouse-20-40-10-2-1$ **(l)** $warehouse-20-40-10-2-2$

Figure 2: Cactus chart of $RSoC$ for each configuration of FoV in each sub-solver $(LaCAM/PIBT)$ over # of solved instances.

Table 1: Average *RSoC improvement%* $\left(\frac{RSoC(fPP)-RSoC(PPfPP)}{RSoC(fPP)}\right) * 100\%$ (mean ± std, max and median (med)) of PPFPP over FPP, across domains with different k values. #S is the amount of solved instances by FPP (all instances also were also solved by PPFPP). * and ** marks significant improvement of PPFPP over FPP (Wilcoxon, $p < 0.05$ and $p < 0.01$ accordingly). FPP MS is the makespan of the original plan created by FPP. Each map shows the amount of vertices in it ($|V|$). This table contains part of the maps we ran on, and the other part is in the second table 2 which also contains the mean over all maps.

Map	k	*RSoC Improvement%*			#S	time	FPP
		mean ± std	max	med		[s]	MS
brc202d $\|V\| = 43{,}151$	2	$0.77 \pm 2.47^{**}$	11.50	0.02	56	113.6	943.9
	3	$0.64 \pm 1.65^{**}$	7.25	0.02	55	116.0	975.4
	4	$1.04 \pm 2.69^{**}$	11.30	0.02	47	121.2	1022.4
	5	$1.06 \pm 2.66^{**}$	11.27	0.00	40	125.3	1041.3
	6	$1.20 \pm 2.68^{**}$	10.72	0.02	39	124.0	1040.9
	7	$1.80 \pm 3.90^{**}$	12.90	0.03	42	131.0	1094.5
	8	$2.04 \pm 4.45^{**}$	16.91	0.07	30	137.6	1170.1
	9	$3.21 \pm 6.40^{**}$	22.83	0.11	25	143.4	1203.6
	10	$1.01 \pm 2.69^{**}$	13.21	0.07	25	143.8	1312.6
lt_gallowstemplar_n $\|V\| = 10{,}021$	2	$0.64 \pm 1.66^{**}$	8.23	0.00	59	3.2	245.2
	3	$0.88 \pm 1.77^{**}$	8.02	0.14	50	3.2	270.3
	4	$0.24 \pm 0.84^{**}$	5.60	0.07	45	3.3	284.4
	5	$0.87 \pm 1.65^{**}$	5.14	0.14	34	3.7	316.6
	6	$0.74 \pm 1.75^{**}$	7.20	0.15	24	3.9	348.0
	7	$0.70 \pm 2.51^{**}$	12.87	0.09	26	3.8	342.4
	8	$0.89 \pm 2.06^{**}$	9.54	0.26	21	4.8	421.0
	9	$2.06 \pm 3.43^{**}$	10.30	0.36	12	5.0	418.8
	10	$0.45 \pm 1.13^{**}$	4.03	0.15	12	5.9	508.2
ost003d $\|V\| = 13{,}214$	2	$0.59 \pm 1.84^{**}$	8.83	0.00	52	7.2	346.6
	3	$1.19 \pm 3.14^{**}$	13.15	0.05	49	7.0	363.9
	4	$0.68 \pm 1.88^{**}$	9.27	0.04	40	6.9	375.8
	5	$1.22 \pm 2.86^{**}$	10.38	0.06	30	7.9	410.8
	6	$2.13 \pm 4.02^{**}$	17.06	0.15	30	7.5	401.2
	7	$1.29 \pm 2.45^{**}$	6.89	0.13	18	7.8	427.9
	8	$2.20 \pm 4.54^{**}$	16.25	0.15	14	9.7	518.4
	9	$2.49 \pm 3.75^{**}$	10.33	0.20	14	9.0	461.8
	10	$0.10 \pm 0.06^{**}$	0.17	0.11	8	12.9	665.5
random-32-32-20 $\|V\| = 819$	2	$0.64 \pm 1.38^{**}$	4.83	0.00	43	0.0	48.3
	3	$2.14 \pm 3.51^{**}$	10.66	0.41	30	0.1	61.5
	4	$0.78 \pm 0.84^{**}$	3.33	0.50	19	0.1	108.5
	5	$1.73 \pm 2.66^{**}$	8.38	0.41	11	0.1	192.6
	6	0.14 ± 0.13	0.25	0.16	4	0.2	286.5
random-64-64-20 $\|V\| = 3{,}270$	2	$0.20 \pm 1.05^{**}$	7.75	0.00	56	0.3	90.5
	3	$0.62 \pm 1.89^{**}$	10.34	0.00	48	0.3	96.7
	4	$0.73 \pm 1.84^{**}$	7.65	0.00	38	0.3	100.3
	5	$1.33 \pm 2.60^{**}$	10.00	0.23	29	0.4	106.8
	6	$1.04 \pm 2.30^{**}$	8.33	0.20	21	0.4	120.4
	7	$0.70 \pm 1.24^{**}$	4.85	0.19	19	0.4	146.4
	8	$0.93 \pm 1.88^{**}$	5.72	0.23	15	0.5	151.8
	9	$2.97 \pm 5.36^{**}$	18.32	0.16	12	0.7	244.6
	10	$0.86 \pm 1.55^{*}$	3.64	0.16	5	0.9	294.6
room-32-32-4 $\|V\| = 682$	2	$1.17 \pm 1.61^{**}$	4.32	0.29	28	0.0	65.2
	3	$0.93 \pm 2.08^{**}$	6.84	0.32	10	0.1	123.9

Table 2: Average *RSoC improvement%* $\left(\frac{RSoC(fPP)-RSoC(PPfPP)}{RSoC(fPP)}\right) * 100\%$ (mean ± std, max and median (med)) of PPFPP over FPP, across domains with different k values. #S is the amount of solved instances by FPP (all instances also were also solved by PPFPP). * and ** marks significant improvement of PPFPP over FPP (Wilcoxon, $p < 0.05$ and $p < 0.01$ accordingly). FPP MS is the makespan of the original plan created by FPP. Each map shows the amount of vertices in it ($|V|$). This is a complementary table for the rest of the maps not shown in table 1 including the mean over all maps.

Map	k	*RSoC Improvement%*			#S	time	FPP
		mean ± std	max	med		[s]	MS
room-64-64-16 $\|V\| = 3{,}646$	2	$0.95 \pm 2.47^{**}$	11.21	0.00	56	0.6	164.0
	3	$1.06 \pm 3.73^{**}$	25.74	0.14	50	0.5	172.4
	4	$0.50 \pm 0.98^{**}$	3.51	0.11	36	0.6	180.5
	5	$1.80 \pm 3.98^{**}$	19.11	0.19	27	0.7	196.7
	6	$1.13 \pm 2.45^{**}$	11.09	0.11	22	0.8	242.2
	7	$2.76 \pm 5.64^{**}$	18.33	0.24	12	0.9	258.7
	8	$2.02 \pm 3.53^{**}$	9.80	0.33	11	0.8	239.5
	9	0.08 ± 0.11	0.15	0.08	2	1.1	300.5
	10	0.11 ± 0.08	0.21	0.10	4	2.2	635.5
room-64-64-8 $\|V\| = 3{,}232$	2	$0.32 \pm 1.11^{**}$	6.93	0.00	51	0.3	125.6
	3	$0.13 \pm 0.14^{**}$	0.38	0.11	34	0.4	144.9
	4	$0.56 \pm 1.68^{**}$	8.02	0.17	22	0.5	182.4
	5	$0.66 \pm 1.29^{**}$	3.94	0.13	16	0.6	241.9
	6	$0.19 \pm 0.19^{*}$	0.45	0.06	5	0.8	350.2
	7	$0.11 \pm nan$	0.11	0.11	1	0.5	197.0
warehouse-20-40-10-2-1 $\|V\| = 22{,}599$	2	$0.00 \pm 0.01^{*}$	0.08	0.00	59	11.5	341.2
	3	0.00 ± 0.01	0.05	0.00	59	11.1	348.7
	4	$0.00 \pm 0.02^{*}$	0.08	0.00	56	11.2	361.3
	5	$0.26 \pm 1.25^{**}$	9.03	0.00	57	11.4	367.8
	6	$0.34 \pm 1.17^{**}$	4.91	0.00	55	11.5	374.7
	7	$0.67 \pm 2.16^{**}$	10.98	0.00	50	11.9	386.5
	8	$0.27 \pm 1.06^{**}$	4.99	0.00	50	11.9	384.4
	9	$0.99 \pm 2.39^{**}$	9.29	0.00	49	12.7	408.0
	10	$0.95 \pm 2.14^{**}$	9.29	0.00	50	12.4	404.5
warehouse-20-40-10-2-2 $\|V\| = 38{,}756$	2	$0.05 \pm 0.22^{*}$	1.07	0.00	58	41.7	377.2
	3	$0.04 \pm 0.17^{**}$	1.11	0.00	55	40.3	389.1
	4	$0.32 \pm 1.27^{**}$	6.97	0.00	57	36.2	392.9
	5	$0.63 \pm 1.75^{**}$	8.65	0.00	45	34.6	397.6
	6	$0.19 \pm 0.60^{**}$	2.65	0.00	44	35.1	406.7
	7	$0.35 \pm 1.46^{**}$	8.31	0.00	34	37.2	428.3
	8	$0.75 \pm 1.73^{**}$	7.41	0.00	35	39.7	455.7
	9	$0.16 \pm 0.65^{**}$	3.11	0.00	23	38.3	437.3
	10	$0.02 \pm 0.03^{*}$	0.12	0.00	15	35.9	434.6
Mean	2	$0.50 \pm 1.61^{**}$	11.50	0.00	518	19.5	293.3
	3	$0.68 \pm 2.22^{**}$	25.74	0.00	440	22.3	336.9
	4	$0.49 \pm 1.55^{**}$	11.30	0.00	360	24.6	374.7
	5	$0.93 \pm 2.35^{**}$	19.11	0.02	289	26.4	408.3
	6	$0.83 \pm 2.25^{**}$	17.06	0.00	244	30.2	451.8
	7	$1.04 \pm 2.86^{**}$	18.33	0.03	202	37.7	507.6
	8	$1.06 \pm 2.78^{**}$	16.91	0.04	176	36.2	518.6
	9	$1.66 \pm 3.90^{**}$	22.83	0.05	137	38.6	548.7
	10	$0.71 \pm 1.93^{**}$	13.21	0.05	119	41.5	630.2