The Reachability Objective in Multi-Agent Path Finding

Noy Gabay*
Ben-Gurion University of the Negev
Be'er Sheva, Israel
noygab@post.bgu.ac.il

Jonathan Morag*
Ben-Gurion University of the Negev
Be'er Sheva, Israel
moragj@post.bgu.ac.il

Ariel Felner
Computer and Information Sciences
Ben-Gurion University of the Negev
Be'er Sheva, Israel
felner@bgu.ac.il

Roni Stern
Computer and Information Sciences
Ben-Gurion University of the Negev
Be'er Sheva, Israel
roni.stern@gmail.com

ABSTRACT

Multi-Agent Path Finding (MAPF) is the problem of path planning for multiple agents while avoiding collisions. In MAPF, each agent must reach a designated target location and stay there. We consider a different objective: each agent must reach its target, but it may move away afterwards. We call this MAPF with Reachability Objective (MAPF-RO). Despite compelling real-world use cases, few prior works have explicitly studied MAPF-RO. We propose several efficient and complete algorithms for MAPF-RO, based on state-of-the-art MAPF algorithms. Experimental results show that using MAPF-RO algorithms instead of classic MAPF algorithms yields huge benefits in terms of runtime and solvability, solving in some cases problems with 1,000 more agents than baseline approaches. Then, we show a reduction that allows using our MAPF-RO to solve MAPF with Unassigned Agents, a recent practical variant of MAPF in which only a subset of the agents are assigned targets.

KEYWORDS

Multi-agent path finding; Reachability objective; Unassigned agents

ACM Reference Format:
Noy Gabay, Jonathan Morag, Ariel Felner, and Roni Stern. 2026. The Reachability Objective in 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, 9 pages. https://doi.org/10.65109/OOLP5568

1 INTRODUCTION

Multi-Agent Path Finding (MAPF) is the problem of finding paths for multiple agents from their start locations to specified target locations without *conflicts*. A conflict occurs when two agents occupy the same location or traverse the same edge in opposite directions at the same time. MAPF is computationally challenging: solving MAPF is NP-hard on general graphs [15] and finding optimal solutions for common cost functions is NP-hard even for undirected graphs [25, 30]. MAPF arises in a wide range of real-world applications, including automated warehouses, autonomous vehicle routing, and multi-robot systems. Consequently, MAPF has been extensively studied [1, 5, 6, 8, 18, 29].

In classic MAPF [24], the agents must end up at their targets. In this work, we focus on a different objective, where each agent needs to *reach* its target at some point, but it may then move and end up at a different location. We call this problem *MAPF with the Reachability Objective* (MAPF-RO).

MAPF-RO captures practical scenarios such as a *search and rescue* mission, where a team of emergency vehicles — agents — must navigate dangerous terrain to reach designated survivors and attend to them. Mission control assigns agents to survivors, and then the objective is to attend to the survivors as fast as possible, while avoiding collisions between the emergency vehicles. Enforcing that the agents reach the survivors at the same time is redundant, may increase planning complexity, and yield highly inefficient solutions. Similarly, in *pickup and delivery* [12], the agents have no reason to remain at a location after picking up or delivering a package at a target location. The distinction between MAPF and MAPF-RO is critical in environments that are densely populated with agents, where most locations are occupied. In such dense environments, forcing the agents to reach a strict goal configuration resembles solving a sliding-tile puzzle and significantly complicates planning. Indeed, MAPF research rarely considers such dense problems.

The reachability objective is also natural in Lifelong MAPF (LMAPF) [11], where an agent is assigned a new target once the previous one has been reached. Recently, Morag et al. [14] explored the MAPF-RO problem within the context of Lifelong MAPF (they referred to this problem as MAPF4L). They proved it is NP-hard to solve in directed graphs and NP-hard to optimize in undirected graphs, under common solution cost functions. They also showed how to adapt basic MAPF algorithms to solve MAPF-RO. However, the resulting MAPF-RO algorithms could not scale to solve hard MAPF-RO problems.

Our first contribution is complete, highly scalable, MAPF-RO algorithms based on LaCAM [17], a state-of-the-art MAPF algorithm. A key challenge when designing a MAPF-RO algorithm is where to guide an agent *after* it has reached its target. We explore several heuristics for addressing this challenge, and show a simple and effective heuristic for setting *dummy targets* for such agents. Then, we propose an anytime algorithm for solving MAPF-RO, that

*Both authors contributed equally to this research.

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/OOLP5568

is inspired by the PIE framework for interleaving planning and execution in MAPF [31].

Our second contribution is a comprehensive experimental study showing that the proposed algorithms can solve extremely difficult problems where the agents occupy almost every available space. These MAPF-RO algorithms outperform all previously proposed MAPF-RO algorithms as well as their classic MAPF counterparts in terms of coverage (# problems solved with a given time limit) and solution cost, as agent density increases.[1]

An important feature of all MAPF-RO algorithms is that they are allowed to move an agent *after* it has reached its assigned target. Such *unassigned agents* also exists when solving the *MAPF with Unassigned Agents* (MAPF-UA) problem — a generalization of MAPF-RO where all agents can move, but only a subset of agents are *assigned* targets and the rest are *unassigned* [2]. Practical use cases of MAPF-UA include reconfigurable warehouses [28] and autonomous parking lots [**?**]. In autonomous parking lots, typical operations involve only a handful of vehicles that are assigned targets (e.g., entering, exiting, or repositioning). The locations and movements of the other vehicles (the vast majority) are of no concern, except in the service of facilitating the movements of the active vehicles. **Our third contribution** is to show how MAPF-RO algorithms can be used to solve MAPF-UA problems under the reachability objective. Our experiments reveal that using MAPF-RO-based algorithms to solve MAPF-UA problems significantly improves runtime and coverage when compared to baseline approaches.

2 DEFINITIONS AND BACKGROUND

A *Multi-Agent Path Finding (MAPF)* problem is defined by a graph $G = (V, E)$, a set of agents A, and for each agent $a_i \in A$, a designated start vertex $s_i \in V$ and target vertex $t_i \in V$. A *solution* π assigns every agent $a_i \in A$ a path $\pi_i = \langle v_i^0, v_i^1, \dots \rangle$ that starts at s_i and ends at t_i. Each consecutive pair (v_i^x, v_i^{x+1}) represents a valid move at time step x, which can be either a move to an adjacent vertex ($(v_i^x, v_i^{x+1}) \in E$), or a wait action at the current vertex ($v_i^x = v_i^{x+1}$). The solution must be conflict-free, considering two standard conflict types in MAPF [24]. A *vertex conflict* occurs if there exists a time step x such that $\pi_i[x] = \pi_j[x]$ for some pair of agents $a_i \neq a_j$, meaning both occupy the same vertex simultaneously. A *swapping conflict* occurs if there exists a time step x such that $\pi_i[x] = \pi_j[x+1]$ and $\pi_i[x+1] = \pi_j[x]$, meaning the agents attempt to traverse the same edge in opposite directions at the same time. The cost of a path π_i is the number of time-steps required to execute it, i.e., $|\pi_i| - 1$. Two standard cost functions for MAPF solutions are *Sum Of Costs* (SOC): the sum of individual path costs over all agents, and *Makespan*: the maximum cost among all agents' paths.

MAPF algorithms. Certain MAPF algorithms, such as Conflict-Based Search (CBS) [22], are guaranteed to return optimal solutions with respect to SOC or makespan. Other algorithms, such as LaCAM [17], sacrifice optimality for runtime but are still *complete*, i.e., are guaranteed to return a valid solution if one exists and identify if one does not exist in finite time. Other algorithms, such as Prioritized Planning (PrP) [3], do not guarantee optimality or completeness. *Anytime* MAPF algorithms, such as MAPF-LNS [9], LaCAM* [17], and PIE [31] return an initial solution quickly and continuously improve its quality as more computation time is given.

[1]Code, Supplementary Material: github.com/J-morag/MAPF/releases/26.AAMAS.RO

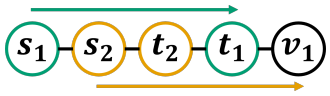

Figure 1: A MAPF instance that is not solvable for classic MAPF but solved with MAPF-RO [14].

3 MAPF WITH REACHABILITY

In this work, we focus on a relaxed variant of the classic MAPF problem, which we refer to as MAPF with the Reachability Objective (MAPF-RO). The input to MAPF-RO is identical to classic MAPF. A solution in MAPF-RO maps each a_i to a path π_i that starts at s_i and *contains* t_i. That is, in MAPF-RO, each agent may reach its target asynchronously from other agents, and agents do not need to reach a configuration where all agents are at their targets simultaneously. MAPF-RO aligns with applications such as automated warehouses, where robots must complete tasks at certain locations, but do not necessarily have to complete their tasks simultaneously. Other use cases, such as search and rescue, were given in the introduction.

Figure 1 shows a scenario highlighting the difference between classic MAPF and MAPF-RO. The green and orange agents aim to move from s_1 to t_1 and s_2 to t_2, respectively. As a classic MAPF problem, this problem is unsolvable: both agents must reach and remain at their targets, which requires them to swap locations, causing a conflict. However, the same scenario can be easily solved as a MAPF-RO problem: agent a_2 (orange) visits t_2 and then further moves to v_1, allowing agent a_1 (green) to reach t_1.

The SOC and makespan cost function can also be used to measure the quality of a MAPF-RO solution. However, SOC and makespan do not optimize how quickly the agents reach their target. An alternative cost function is the *Sum of Service Times* (SST) [14, 19]. SST directly considers the earliest time step when each agent visited its target, ignoring subsequent steps. Formally:

$$SST(\pi) = \sum_{i=1}^{|A|} \min\{x \mid \pi_i[x] = t_i\}$$

For example, in Figure 1, the orange agent visits its target t_2 at time step 1, but continues to v_1 (arriving at time step 3), while the green agent reaches t_1 at time step 3. The SOC of this solution is $3 + 3 = 6$, but its SST is only $1 + 3 = 4$, since it ignores the steps the orange agent takes after reaching t_2.

3.1 Baseline and Existing Approaches

Several baseline techniques that were proposed for LMAPF can be used to adapt existing MAPF algorithms to solve MAPF-RO. One technique is to remove agents from the environment once they reach their targets [13, 26]. This simplifies planning, but may require online replanning to avoid collisions in applications where the agents do not disappear from the environment. Another technique employs dummy targets to guide agents away from their targets after they reach them [11]. Selecting effective dummy targets, however, is non-trivial, and poorly selected dummy targets

Table 1: Approaches for MAPF-RO. (*) PIBT is complete for bi-connected graphs. (**) CBS_{RO} is solution-complete.

Approach	Sound	Complete	Optimal
Disappear at target	No	Yes	n/a
Dummy targets [31]	Yes	No	No
RHCR [11]	Yes	No	No
PIBT [19]	Yes	Partially (*)	No
PrP_{RO} [14]	Yes	No	No
LNS_{RO} [14]	Yes	No	No
CBS_{RO} [14]	Yes	Yes (**)	Yes
$LaCAM_{RO}$	Yes	Yes	No
PIE_{RO}	Yes	Yes	No

can introduce inefficiency and even lead to incompleteness. Rolling Horizon Collision Resolution (RHCR) [11] is a well-known technique for LMAPF in which the agents plan every fixed number of steps, ignoring conflicts that are planned to occur beyond a fixed time window. RHCR may be used to solve MAPF-RO, but it is incomplete because it is not known how to choose the window size, and whether an appropriate window size exists.

Reachability (as we consider) was first introduced by Okumura et al. [19] in their seminal work on the *Priority Inheritance with Backtracking* (PIBT) algorithm. PIBT is a rule-based MAPF algorithm that plans one step at a time. If the environment is bi-connected, i.e., every pair of adjacent vertices belongs to a simple cycle, then PIBT guarantees that every agent will reach its target in finite time. Thus, under these conditions PIBT is *complete* for MAPF-RO. However, PIBT is incomplete for MAPF-RO in environments that are not bi-connected, nor does it provide any solution quality guarantees. In general, all these baseline approaches risk overlooking longer-term interactions, losing either soundness or completeness, and potentially leading to deadlocks and inefficiencies.

Morag et al. [14] directly studied the MAPF-RO problem within the context of LMAPF. They implemented several MAPF-RO algorithms based on existing MAPF algorithms, namely PrP [4], LNS [9], and CBS [22]. These MAPF algorithms rely on a low-level search algorithm, e.g., A*, to find paths for individual agents subject to specific constraints — actions and locations that must be avoided at specific time steps. To adapt these algorithms to MAPF-RO, Morag et al. modified their low-level search such that each state includes a Boolean flag indicating whether the agent has already reached its target. The low-level search halts when (1) the Boolean flag is `True`, meaning the agent has reached its target at some point, and (2) the agent can remain at its current location (which may be different from its target) indefinitely without violating any constraints. They also modified the cost function used by the low-level search (and for CBS passed to the high-level) to optimize SST. To achieve this, the cost of making additional moves after an agent reaches its target is set to 0. That is, in a given branch of the low-level search tree, all nodes with the flag set to `True` have the same cost. As a result, once the low-level search finds the agent's target, if some future constraint prevents the search from terminating at the target, it will then behave similarly to breadth-first search around the agent's target, looking for a location with no future constraints. We refer to Morag et al.'s version of PrP and LNS, and CBS as PrP_{RO}, LNS_{RO}, and CBS_{RO}, respectively.

4 SCALABLE SOLUTIONS FOR MAPF-RO

The top seven rows in Table 1 list the aforementioned existing approaches to solving MAPF-RO and their different properties. As can be seen, only CBS_{RO} is sound, complete, and optimal. A known limitation of optimal algorithms solving NP-hard problems, such as MAPF-RO, is their scalability. While Morag et al. did not evaluate CBS_{RO} empirically, our experimental results confirmed that indeed CBS_{RO} fails to scale to solve large MAPF-RO problems. In this work, we focus on MAPF-RO algorithms that are complete and scale well, at the cost of finding possibly suboptimal solutions.

4.1 LaCAM for MAPF-RO

The first MAPF-RO algorithm we present is an adaptation of the *Lazy Constraints Addition Search for MAPF* (LaCAM) [17] algorithm. LaCAM is a complete and highly scalable MAPF algorithm. Like CBS, it also employs a two-level search. The high-level search is a depth-first search over the configuration space, where a configuration is a vector containing a location for each agent. The high-level search continues until it reaches a node with the goal configuration; i.e., each agent is at its target location. When expanding a high-level node, LaCAM invokes a low-level search in the space of possible next-step configurations (the Cartesian product of the sets of legal moves for each agent), eventually choosing a single configuration to generate next. The LaCAM low-level search starts by prioritizing the agents, either arbitrarily or using some heuristic. A commonly used heuristic for this is the distance from each agent's current location to its target location, with larger distances receiving higher (better) priority. Then, a process akin to a one-step PIBT is employed to generate the next configuration: each agent tries to apply the action that moves it closest to its target while avoiding conflicts with the actions applied by higher priority agents. Priority inheritance and backtracking mechanisms from PIBT are used to handle conflicting actions.

LaCAM may call the low-level search to generate a configuration for the same high-level node multiple times. This occurs either when the high-level depth-first search backtracks (when a dead-end is reached by the search) or when the same configuration is reached from different paths. LaCAM ensures that whenever this occurs, a different configuration is generated by maintaining an internal constraint tree in every high-level node. Consequently, LaCAM is guaranteed to eventually visit all configurations in the worst case, and is therefore a complete MAPF algorithm. LaCAM is not complete for MAPF-RO since some MAPF-RO problems are not solvable as a MAPF problem (e.g., Figure 1).

4.2 $LaCAM_{RO}$

$LaCAM_{RO}$ modifies the LaCAM high-level search by adding a Boolean array to each high-level node, denoted as `Reached`, with one entry per agent. `Reached`[i] is `True` if agent a_i has, at any point along the current high-level search branch, reached its target. In the root node of the search, all entries in `Reached` are set to `False`. Whenever a high-level node is generated, it inherits the `Reached`

values from its parent node. If an agent visits its target location in the new node's configuration, the corresponding entry (Reached[i]) is set to True. LaCAM$_{RO}$ terminates successfully when it generates a high-level node where all values in Reached are True. Importantly, two high-level nodes in LaCAM$_{RO}$ are considered duplicates only if they have the same configuration *and* Reached values.

Theorem 1. *LaCAM$_{RO}$ is complete for MAPF-RO.*

Proof outline: In a MAPF-RO solution, there is no need to revisit the same configuration with the same subset of agents that have reached their targets. Thus, a MAPF-RO solution can be represented as a sequence of unique high-level MAPF-RO nodes. The LaCAM$_{RO}$ search space is finite, including at most every possible configuration of agents and boolean assignments to the Reached vector. Thus, the DFS performed in the high-level search will eventually visit every high-level node, since LaCAM$_{RO}$ detects duplicates. □

4.2.1 Enhancements to the LaCAM$_{RO}$ low-level search. We also implemented several changes in the LaCAM$_{RO}$ low-level search to better tune it for solving MAPF-RO efficiently. Recall that in the LaCAM low-level search, we first prioritize by agents, and then each agent prioritizes the actions it may take. LaCAM$_{RO}$ sets the priority of agents that have already visited their target to the lowest priority, causing them to make way for agents that have yet to reach their targets. We also considered several ways in which each agent could prioritize its actions after it has reached its target. A basic approach would be for the agent to randomly choose its next action (wait or move to some adjacent location) without any preference. We call this variant LaCAM$_{RO(b)}$. Further, inspired by Zhang et al. [31], we propose to prioritize actions by assigning a "dummy" target an agent aims to reach after it has reached its original target. Specifically, when solving LMAPF with PIE, Zhang et al. [31] proposed to select a unique random dummy target for each agent from the highest-degree vertices in the graph. An agent's actions are then prioritized based on how close they move the agent towards its dummy target. We modified the dummy target selection slightly by assigning each agent the *closest* highest-degree vertex as its dummy target. We call this approach LaCAM$_{RO(cd)}$. We also suggest another baseline, LaCAM$_{RO(rd)}$, where dummy targets are chosen completely at random.

Algorithm 1 describes how LaCAM$_{RO(cd)}$ selects dummy targets. First, we find all dummy location candidates, which are vertices that have the highest degree in the graph (line 1). Then, we iterate over the agents (lines 4), and greedily assign each agent with a dummy that is closest to its original target (line 8-10). We make sure that no two agents are assigned the same dummy target. If all vertices with the highest degree are used, but agents without an assigned dummy target remain, we start assigning vertices with the next-highest degree in the graph (lines 5-7).

Importantly, the high-level search of the LaCAM$_{RO}$ variants halts when all agents have reached their original targets — agents need not reach their dummy targets. Additionally, since no configurations are eliminated (only prioritized), completeness is maintained.

Algorithm 1 Closest-Highest-Degree Dummy Targets

Require: Graph $G = (V, E)$; set of agents A ; distance heuristic $h : V \to \mathbb{N}$
Ensure: Dummy target function $Dummy : A \to V$
1: $C \leftarrow$ all vertices with maximal degree in V ▷ candidates
2: $Used \leftarrow \emptyset$
3: $Dummy \leftarrow \emptyset$
4: **for** $a_i \in A$ **do**
5: **if** $C \setminus Used = \emptyset$ **then**
6: $C \leftarrow$ all vertices with maximal degree in $V \setminus Used$
7: **end if**
8: $u \leftarrow \arg\min_{v \in C \setminus Used} h(t_i, v)$
9: $Dummy \cup \{\langle a, u \rangle\}$
10: $Used \leftarrow Used \cup \{u\}$
11: **end for**
12: **return** $Dummy$

4.3 An Anytime Algorithm for MAPF-RO

A known limitation of LaCAM is the quality of the solution it generates. One approach to mitigate this is LaCAM*, an anytime version of LaCAM in which the high-level search continues, even after reaching a goal configuration, to search for lower cost solutions [18]. A more effective design for an anytime algorithm, inspired by the recently introduced Planning and Improving while Executing (PIE) [31] algorithmic framework, is to combine LaCAM and LNS, using the former to quickly find an initial plan and then the latter to iteratively refine it. PIE was shown to be more effective than LaCAM* [27], and thus we designed our anytime MAPF-RO algorithm accordingly, and refer to it as PIE$_{RO}$. One disadvantage of this approach is that LaCAM* guarantees to eventually return an optimal solution, whereas PIE does not. PIE$_{RO}$ follows the same structure as PIE but modifies both the initial and iterative planning components: it uses LaCAM$_{RO}$ instead of LaCAM to find an initial solution, and uses LNS$_{RO}$ instead of LNS to iteratively refine it. Additionally, all components of PIE$_{RO}$ are modified to minimize SST instead of SOC. While we use PIE$_{RO}$ as an anytime MAPF-RO algorithm, it can naturally also be used when planning and execution are interleaved, as the intended use of the original PIE.

5 EXPERIMENTAL RESULTS

In this section, we present an experimental evaluation of MAPF-RO algorithms in various environments. As baselines, we considered the classic MAPF versions of these algorithms, namely LaCAM and PIE, as well as the previously proposed MAPF-RO algorithms, namely PrP$_{RO}$, LNS$_{RO}$, and CBS$_{RO}$. We selected a representative set of grids from the standard grid-based MAPF benchmark [24], featuring varied topological patterns, including empty grids with and without random obstacles, game maps, mazes, and warehouses. Each grid is associated with a set of *scenario files*. A scenario file defines a list of potential agents, each with a specified start and target location. For each instance, a subset of agents is selected from this file based on the desired number of agents. The relaxation provided by MAPF-RO compared to classic MAPF allows solving significantly harder and denser problems than those provided by the scenario files usually used for this benchmark. To challenge the evaluated algorithms, we therefore generated 50 new scenario files per map, where the maximum number of agents allowed equals the number of free cells, meaning each map can be filled with agents

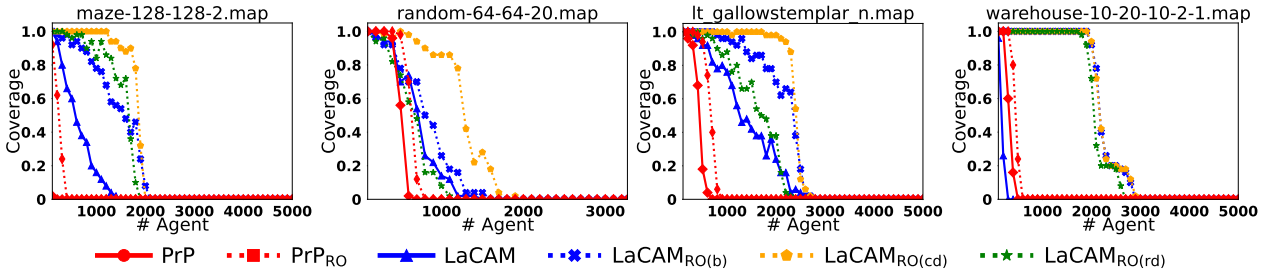

Figure 2: Coverage achieved by different algorithms in different dense maps as a function of the number of agents.

to full capacity. In all scenario files, agents are assigned unique start locations and unique target locations. We implemented all algorithms using Java 18. We ran each instance with a time limit of 60 seconds per solver on Linux virtual machines, in a cluster of AMD EPYC 7763 CPUs, with 20GB of RAM and a single core each.

5.1 Coverage results

In the first set of experiments, we measured the ratio of MAPF-RO problems solved by each algorithm within the 60-second time limit. This is also known as *coverage*. Here, we show results only for PrP, PrP$_{RO}$, LaCAM, and all versions of LaCAM$_{RO}$. PIE and PIE$_{RO}$ are omitted since they only improve the initial solution generated by LaCAM and LaCAM$_{RO}$, respectively, so they provide the same coverage. Similarly, LNS and LNS$_{RO}$ only improve the initial solution generated by PrP and PrP$_{RO}$, respectively, and thus their results are also omitted. We also implemented a version of MAPF-LNS2 [10] for MAPF-RO, but it always yielded significantly weaker performance than LaCAM$_{RO}$. These results are presented in the supplementary material.

In this experiment, we varied the number of agents starting from 100 and increasing in increments of 100 up to 5000 or until there were no more empty cells for agents. Figure 2 presents the coverage results for `maze-128-128-2`, `random-64-64-20`, `lt_gallowstemplar_n`, and `warehouse-10-20-10-2-1`. Results for the other grids are available in the supplementary material. The x-axis displays the number of agents, and the y-axis displays the coverage achieved by each algorithm. The general trend across all maps shows that all versions of LaCAM$_{RO}$ consistently outperform the original LaCAM, PrP$_{RO}$, and PrP in terms of coverage, particularly when the number of agents increases. The only exception is LaCAM$_{RO(rd)}$, which sometimes slightly underperforms LaCAM, likely because in very dense scenarios, moving towards a random location is not always better than staying in place. For example, in `warehouse-10-20-10-2-1`, neither PrP nor LaCAM were able to solve more than 400 agents, while LaCAM$_{RO(rd)}$ and LaCAM$_{RO(cd)}$ maintained 100% coverage even up to 2,600 agents, and partial coverage up to 2,900 agents. These results demonstrate the advantage of using a solver specifically designed for MAPF-RO. In addition, we show that our LaCAM-based MAPF-RO solvers greatly outperform prior MAPF-RO algorithms.

Next, we consider the results of the different LaCAM$_{RO}$ variants (Section 4.2.1). In most maps, LaCAM$_{RO(cd)}$ matched or outperformed LaCAM$_{RO(b)}$ and LaCAM$_{RO(rd)}$. For example, in `lt_gallowstemplar_n`, LaCAM$_{RO(cd)}$ maintains above 90% coverage

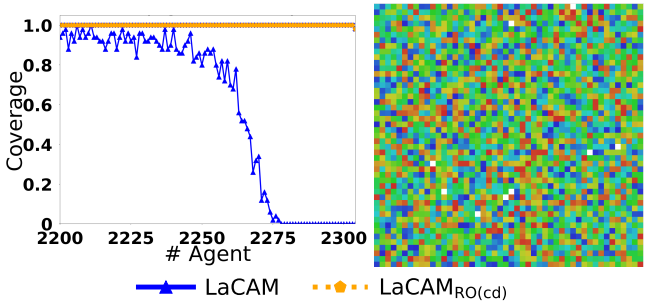

Figure 3: (left) Coverage results on the 48x48 dense instances, and (right) an illustration of such an instance.

up to 2,300 agents, while LaCAM$_{RO(b)}$ dropped below 90% coverage beyond 900 agents. Overall, LaCAM$_{RO(cd)}$ tends to be better and more robust, with rare map-dependent deviations. Thus, we used this variant hereinafter, and will simply refer to it as LaCAM$_{RO}$.

5.2 Coverage in extremely dense environments

The results above show that using MAPF-RO algorithms allows solving problems that are much denser with agents compared to classic MAPF algorithms. To demonstrate this further, we performed an additional set of experiments in which the *agent density*, i.e., the ratio of grid cells occupied by agents, is extremely high. For these experiments, we chose the grids `empty-16-16`, `empty-32-32`, and `empty-48-48` from the grid-based MAPF benchmark. For each grid, we began with the maximum possible number of agents, where every empty cell is occupied by an agent, and gradually reduced the number of agents by one agent at a time, down to a pre-defined constant. These constants, which were 220, 950, and 2,200, for `empty-16-16`, `empty-32-32`, and `empty-48-48`, respectively, correspond roughly to the number of agents, where all evaluated algorithms achieved full coverage. Figure 3 (right) illustrates such dense instances in the `empty-48-48` map with 2,295 agents, i.e., where only 9 cells remain unoccupied (marked in white). The results for `empty-48-48` are presented in Figure 3 (left). Similar trends were observed in the other grids (available in the supplementary material). The x-axis displays the number of agents, and the y-axis displays coverage achieved by each algorithm. As can be seen, LaCAM$_{RO}$ (implemented as LaCAM$_{RO(cd)}$) shows a clear, growing advantage over standard LaCAM as the number of agents approaches the maximum capacity, while the coverage achieved by LaCAM$_{RO}$ remains 100% even in the maximum number of agents. This highlights how

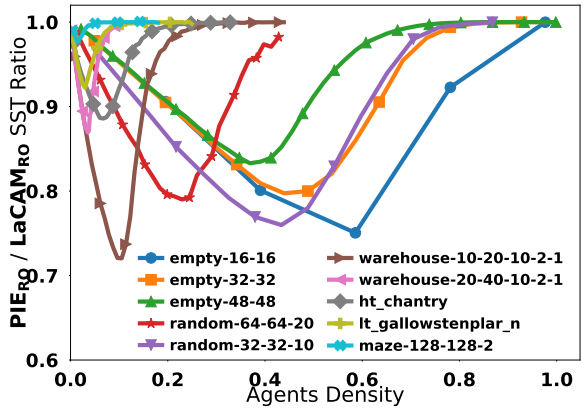

Figure 4: SST ratio (PIE_{RO}/$LaCAM_{RO}$) as a function of the agent density in each map.

LaCAM struggles in high-density configurations, and the benefit of using a dedicated MAPF-RO solver, namely $LaCAM_{RO}$.

5.3 Solution quality

Next, we investigated the benefit of using PIE_{RO} in the MAPF-RO setting to improve the cost, measured in SST, of the solution found by $LaCAM_{RO}$. Figure 4 presents the ratio between the SST achieved by PIE_{RO} and by $LaCAM_{RO}$, averaged over 50 instances for each agent count. We denote the *agent density* by *alr*, i.e., the ratio between the number of agents and the number of empty locations. The x-axis represents *alr* and the y-axis shows the SST ratio, with lower values representing more improvements made by PIE_{RO}. The results are averaged over different grids and numbers of agents, ranging from 50 agents to either the maximum possible on each map or 2,500 agents, after which all algorithms failed to solve.

As can be seen, PIE_{RO} can significantly improve the initial $LaCAM_{RO}$ solution; most notably in maps such as `warehouse-10-20-10-2-1`, `random-32-32-10`, and `empty-16-16`. A clear trend emerges: for low values of *alr*, PIE_{RO} and $LaCAM_{RO}$ perform similarly (ratio ≈ 1); as density increases, PIE_{RO} increasingly outperforms $LaCAM_{RO}$, reaching a point of maximum advantage. In `warehouse-10-20-10-2-1`, for example, PIE_{RO} reaches nearly 30% improvement at 0.105 agent density (600 agents). Beyond that point, the ratio gradually returns to 1. This trend can be explained as follows: at low densities (low *alr*), $LaCAM_{RO}$ can produce near-optimal solutions, leaving limited room for improvement by PIE_{RO}. As the density grows, $LaCAM_{RO}$'s solution quality degrades due to increased congestion, allowing PIE_{RO} to make more significant improvements. At very high densities ($alr \approx 1$, the approach used by PIE_{RO} to improve its incumbent solution — applying LNS— is ineffective. LNS works by replanning for a small subset of agents at a time (the neighborhood), but as the environment becomes more dense, such local improvements are either non-existent or are harder to identify, leading to little gains. These results show that MAPF-RO algorithms are effective even in anytime settings, enabling PIE_{RO} to not only find solutions, but also quickly improve their quality even in high agent densities.

Neither $LaCAM_{RO}$ nor PIE_{RO} yields optimal solutions. To evaluate how suboptimal the solutions they return are, we compared the cost of the solutions they return with that of CBS_{RO}, which is guaranteed to return an optimal solution. For reference, we also used the MAPF counterparts of the above algorithms as baselines, namely LaCAM and PIE. Due to space constraints, we only report here results for `warehouse-10-20-10-2-1` for problems with 10 to 100 agents that were solved by CBS_{RO}. Problems with more agents were not solvable by CBS_{RO} within our time limit. Table 2 shows the average *suboptimality ratio* of each algorithm, which is the ratio between its SST and the optimal SST values, which were found by CBS_{RO} (described in Section 3.1). That is, a suboptimality ratio larger than one indicates a suboptimal solution. The values in brackets are the ratio of problems where a solution with suboptimal SST was returned. As can be seen, all algorithms return SST values close to optimal, but the suboptimality grows as the problems get harder, as expected. The advantage of PIE_{RO} over $LaCAM_{RO}$ is also very clear. For example, for problems with 70 agents PIE_{RO} obtained near-optimal SST, while $LaCAM_{RO}$ obtained a suboptimality ratio of 1.068. Interestingly, the SST of PIE was also close to optimal, yet PIE_{RO} was still able to find optimal solutions in more cases. For example, with 70 agents, PIE found suboptimal solutions in all cases while PIE_{RO} found optimal solutions in 76% of the cases.

Table 2: Suboptimality ratio of our MAPF-RO algorithms.

Agents	LaCAM	$LaCAM_{RO}$	PIE	PIE_{RO}
10	1.008 (0.32)	1.005 (0.36)	1.000 (0.04)	1.000 (0.00)
20	1.014 (0.88)	1.016 (0.88)	1.001 (0.20)	1.000 (0.00)
30	1.025 (0.98)	1.026 (1.00)	1.001 (0.32)	1.000 (0.00)
40	1.036 (1.00)	1.034 (1.00)	1.001 (0.52)	1.000 (0.00)
50	1.046 (1.00)	1.043 (1.00)	1.001 (0.67)	1.000 (0.03)
60	1.055 (1.00)	1.054 (1.00)	1.002 (0.88)	1.000 (0.09)
70	1.061 (1.00)	1.068 (1.00)	1.002 (1.00)	1.000 (0.24)
80-100	1.081 (1.00)	1.077 (1.00)	1.003 (1.00)	1.000 (0.41)

6 MAPF WITH UNASSIGNED AGENTS

We next consider a new variant of MAPF, which can also be seen as a special case of MAPF-RO, where only a subset of the agents are *assigned* targets they must reach, while other agents are *unassigned* and have no targets. This problem is called *Multi-Agent Path Finding with Unassigned Agents (MAPF-UA)* [2]. Solving MAPF-UA introduces new planning challenges: while *unassigned agents* are not fixed obstacles, they still occupy space and may obstruct assigned agents from moving. So both types of agents must coordinate their paths to allow the assigned agents to quickly reach their targets.

A MAPF-UA problem instance is defined by a graph $G = (V, E)$, a set of agents $A = \{a_1, \ldots, a_n\}$, a start vertex $s_i \in V$ for every agent $a_i \in A$, a subset of agents $A' \subseteq A$, and a target vertex $t_i \in V$ for every agent $a_i \in A'$. The agents in A' are referred to as the *assigned agents*. Agents not in A', referred to as the *unassigned agents*, have no targets to go to. A solution π to MAPF-UA maps each agent (assigned or unassigned) to a path in G, where for each assigned agent $a_i \in A'$, the path π_i starts at s_i and contains t_i. For each unassigned agent $a_j \in A$, the path π_j only needs to start at s_j, but since they have no target, they can end up anywhere. The paths of all agents, assigned or unassigned, must be conflict-free.

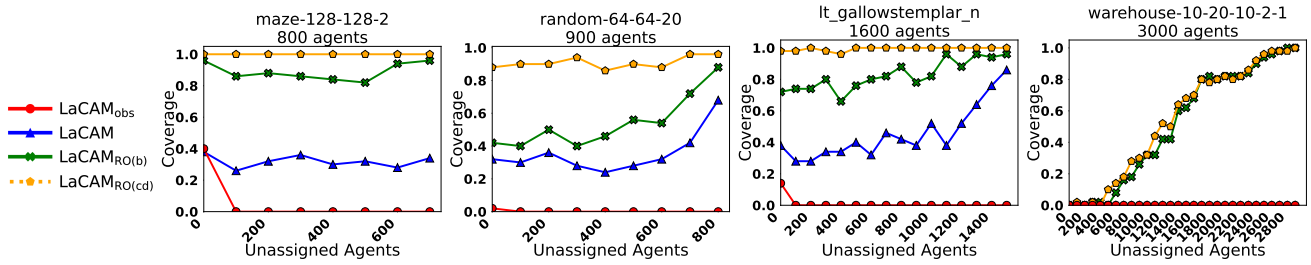

Figure 5: Coverage results for solving MAPF-UA problems, varying the number of unassigned agents.

MAPF-UA is applicable for warehouses where some of the robots do not have tasks, or for modern warehouses where packages themselves have moving abilities but only some of the packages have targets assigned to them. Another example is a dense parking lot where a few cars need to exit, but other cars are blocking them. MAPF-UA is also highly applicable for a heterogeneous team of robots, only some of which are needed at a time, depending on their functions (e.g., construction site, airport, or mine). In this work, we use the SST of the paths mapped to the assigned agents as the cost of MAPF-UA. This quantifies how quickly the targets are visited. Since unassigned agents do not have targets, their paths do not contribute to the SST. Naturally, other cost functions may also be considered for MAPF-UA. Examples include: **(1)** Adding the costs of the unassigned agents. **(2)** Minimizing the number of unassigned agents that are moved (e.g., unassigned cars in the parking lot). **(3)** Minimizing only the move actions (Fuel) of all the agents or only of the assigned agents (as was done by Pertzovsky et al. [21]).

Related Work. Recent work proposed optimal solutions for different MAPF-UA variants using heuristic search [2] or Integer-Linear Programming [16]. These techniques fail to scale to large MAPF-UA problems. Sherma et al. [23] introduced the problem of controlling a set of robots that need to move obstacles to achieve a target configuration. By contrast, in MAPF-UA, there are no carrying robots and *all* agents can move independently. Fu et al. [7] deal with the Block Rearrangement Problem (BRaP). This problem differs from MAPF-UA in that in BRaP (1) each agent is associated with multiple possible target locations it can choose from, and (2) the assigned agents must reach and stay at their target locations, (unlike the reachability objective). In service of their objective, they focused on adapting PIBT within LaCAM to pursue multiple targets.

6.1 Solving MAPF-UA

One approach to solving MAPF-UA is by treating the unassigned agents as static obstacles. This forces the planner to avoid moving any other agent via their start locations, which may lead to *incompleteness*, as some MAPF-UA problems cannot be solved without moving unassigned agents. Another approach to solve MAPF-UA problems is to ignore unassigned agents entirely, assuming during planning that they do not exist in the environment. This can lead to *unsound solutions*, since the planner may generate paths in which assigned agents traverse unassigned agents' locations.

We propose to solve MAPF-UA using MAPF-RO algorithms via the following reduction. For every unassigned agent, assign its start location as its target. This means they will immediately reach their target (before the first move) and effectively become unassigned. This reduction allows us to solve any MAPF-UA problem instance with any MAPF-RO algorithm without needing any specialized logic. It is easy to see that this reduction preserves soundness, optimality, and completeness. That is, any solution to the resulting MAPF-RO problem is a solution to the corresponding MAPF-UA problem (soundness), any optimal solution to the MAPF-UA is an optimal solution to MAPF-RO and vice versa, and a complete MAPF-RO algorithm yields a complete MAPF-UA algorithm.

6.2 MAPF-UA Experimental Results

We evaluated various LaCAM algorithms running on MAPF-UA instances on different maps. For each map, we chose a different total number of agents, ensuring the problem is challenging, yet reasonable for most algorithms. The exact number of agents used is shown in Figure 5. Using this total number of agents, we then varied the number of unassigned agents among them. For instance, in `lt_gallowstemplar_n`, we ran experiments with a total of 1,600 agents and varied the number of unassigned agents from 100 to 1,500. Results for each map were averaged over 50 randomized instances. As baselines, we considered two versions of LaCAM: one that requires all unassigned agents to return to their initial locations, and one that considers unassigned agents as immovable obstacles. We denote the former as LaCAM and the latter as LaCAM_{obs}. These were compared to $\text{LaCAM}_{RO(b)}$ and $\text{LaCAM}_{RO(cd)}$.

Figure 5 plots our results for `maze-128-128-2`, `random-64-64-20`, `lt_gallowstemplar_n`, and `warehouse-10-20-10-2-1`. Results for the other grids are available in the supplementary material. The x-axis shows the number of unassigned agents, and the y-axis shows coverage. Across all grids, LaCAM_{obs} quickly fails as the number of unassigned agents increases, highlighting its incompleteness (due to the existence of static obstacles). LaCAM performs better than LaCAM_{obs}, but achieves lower coverage than the MAPF-RO versions of LaCAM ($\text{LaCAM}_{RO(b)}$ and $\text{LaCAM}_{RO(cd)}$) in many grids. For example, observe the results for `maze-128-128-2`, `random-64-64-20`, and `lt_gallowstemplar_n`. Comparing $\text{LaCAM}_{RO(b)}$ and $\text{LaCAM}_{RO(cd)}$, we see the same trend observed in the MAPF-RO experiments, where $\text{LaCAM}_{RO(cd)}$ usually achieves better coverage, especially in complex or congested maps. Next, consider the impact of increasing the number of unassigned agents. As expected, freeing more agents to be unassigned makes the problem easier for all algorithms except LaCAM_{obs}, which results in increased coverage. LaCAM_{obs} fails when the number of unassigned agents increases, since the numerous static obstacles

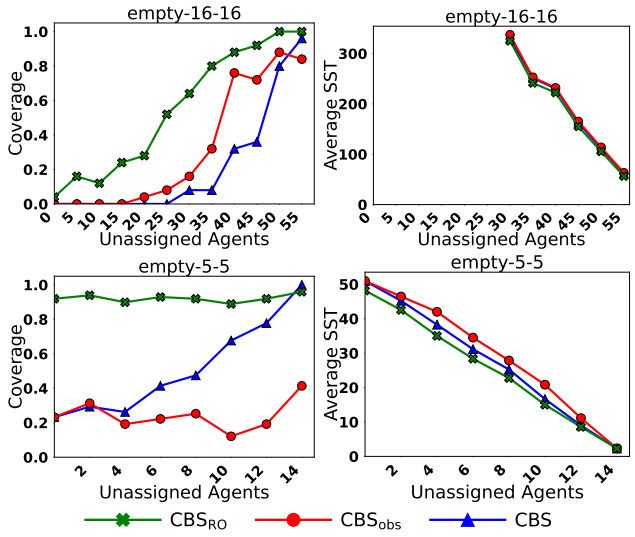

Figure 6: Varying the number of unassigned agents.

create bottlenecks, or even make problems unsolvable by separating the graph into multiple connected components. These trends were most evident in environments like `warehouse-10-20-10-2-1`, `lt_gallowstemplar_n`, and `random-64-64-20`.

We also examined the SST achieved by these algorithms in the same experiment. As expected, the SST decreases as the number of unassigned agents increases, since it does not consider the cost of their movement. We did not observe any significant advantage in terms of SST between the algorithms. Relevant results are available in the supplementary materials. Overall, our results confirm that using our reduction allows MAPF-RO algorithms to solve large MAPF-UA problems efficiently.

6.3 Coverage and SST of CBS for MAPF-UA

We next provide experiments on optimal solutions for MAPF-UA by comparing relevant variants of CBS [22]. We evaluated three policies for handling unassigned agents within CBS. **(1)** CBS_{obs} treats unassigned agents as static obstacles that cannot be moved. **(2)** CBS solves a classic MAPF problem, where the unassigned agents are assigned their start locations as their targets. That is, they are allowed to move away, but must return to their start locations. **(3)** CBS_{RO} solves a MAPF-RO problem as follows. The assigned agents need to travel to their targets. Based on the reduction given in section 6.1, the unassigned agents are assigned their start locations as their targets. This immediately satisfies the requirement for them to visit their targets at some point, and thus (unlike CBS in item 2), they are free to move away without returning to their start/target.

Figure 6 (top) shows results where a total of 60 agents were present in a 16×16 grid, and the number of unassigned agents was varied. The x-axis shows the number of unassigned agents (out of the 60 agents), and the y-axis shows coverage when given 1 minute per instance (left) or average SST (right). As can be seen, the coverage of all CBS variants increases when more agents become unassigned. This is reasonable because *all* three policies for unassigned agents have less planning to do than for regular assigned agents. In CBS_{RO} (green), the unassigned agents have the highest degree of freedom, and it has the best coverage among all variants. Classic CBS (blue) allows unassigned agents to move but forces them to return to their start locations later, while CBS_{obs} (red) treats them as obstacles. Thus, in a sense, classic CBS gives more freedom to the unassigned agents than CBS_{obs}. But, the results show CBS_{obs} has better coverage. This is probably because CBS must create a non-conflicting path for each of these agents, which demands computation time, whereas CBS_{obs} does not have to plan anything for these agents, as they must stay in place. This is especially helpful for CBS_{obs} on relatively sparse environments such as our 16×16 grid with $alr = 60/256 = 0.23$. On this grid, it was easy for CBS_{obs} to find bypasses around the unassigned agents (obstacles). The average SST per agent (over only instances that were solved by all policies) is given in Figure 6 (top-right). All variants have similar SST since the environment was relatively sparse. Thus, all assigned agents found relatively similar cost paths, even though some of them had more time-consuming computations.

Figure 6 (bottom) shows similar results, but for a denser grid. 15 agents were placed into a 5×5 grid where $alr = 15/25 = 0.6$. Coverage is shown in Figure 6 (bottom left). Here, since the graph was dense, the presence of static obstacles made problems more difficult (or impossible) for CBS_{obs}, and thus CBS had better coverage than CBS_{obs}. Again, CBS_{RO} had the best coverage. SST is shown in Figure 6 (bottom right). CBS_{RO} had the lowest SST. CBS had a better SST than CBS_{obs}. The existence of static obstacles caused longer paths for assigned agents. Moving away from these obstacles and returning the agents to their start states resulted in a better SST than keeping them static.

7 CONCLUSION AND FUTURE WORK

We proposed several complete and scalable solvers for MAPF-RO, where agents are only required to reach their targets but are not required to stay there. We adapted LaCAM to solve MAPF-RO, explored several design choices in its implementation, and proposed a scalable anytime MAPF-RO algorithm based on PIE. Our results demonstrated that our MAPF-RO algorithms, LaCAM_{RO}, and PIE_{RO} solve significantly harder problems than existing MAPF-RO algorithms or MAPF baselines, including densely populated problems where agents occupy almost all available locations. Then, we consider the MAPF with Unassigned Agents (MAPF-UA) problem, which is a recently proposed MAPF problem where only a subset of the agents is actively tasked with missions to complete [2]. We showed a reduction from MAPF-UA to MAPF-RO, which allowed our MAPF-RO algorithms to solve hard MAPF-UA problems with thousands of agents. Our exploration of MAPF-RO and MAPF-UA opens many directions for future work. First, our LaCAM_{RO} could be augmented by including recent algorithmic improvements to LaCAM [18, 20]. Another direction is to explore MAPF-UA under different objectives and cost functions. This includes cost functions that also consider the costs of the unassigned agents, and a goal condition in which the assigned agents must stay at their targets, like classic MAPF. In addition, other MAPF algorithms should be modified to solve MAPF-UA, and original MAPF-UA algorithms can also be developed.

ACKNOWLEDGMENTS

This research was funded by ISF grants No. 1238/23 to Roni Stern, No. 909/23 to Ariel Felner and by the Ministry of Science grant No. 0008612/1001948158 to Ariel Felner and Roni Stern.

REFERENCES

[1] Jean-Marc Alkazzi and Keisuke Okumura. 2024. A comprehensive review on leveraging machine learning for multi-agent path finding. *IEEE Access* 12 (2024), 57390–57409.

[2] Roni Stern Ariel Felner. 2026. Blue-Sky: Multi-Agent Path Finding with Unassigned Agents (MAPFUA). In *AAAI Conference on Artificial Intelligence*.

[3] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 1. 271–276.

[4] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Selecký. 2015. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering* 12, 3 (2015), 835–849.

[5] Shao-Hung Chan, Roni Stern, Ariel Felner, and Sven Koenig. 2023. Greedy Priority-Based Search for Suboptimal Multi-Agent Path Finding. In *International Symposium on Combinatorial Search (SOCS)*. 11–19.

[6] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. 2021. PRIMAL _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters* 6, 2 (2021), 2666–2673.

[7] Bo Fu, Zhe Chen, Rahul Chandan, Alex Barbosa, Michael Caldara, Joey Durham, and Federico Pecora. 2025. Symbolic Planning and Multi-Agent Path Finding in Extremely Dense Environments with Movable Obstacles. *arXiv preprint arXiv:2509.01022* (2025).

[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] Jiaoyang Li, Zhe Chen, Daniel Harabor, P Stuckey, and Sven Koenig. 2021. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[10] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. 2022. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *AAAI Conference on Artificial Intelligence*, Vol. 36. 10256–10265.

[11] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *AAAI Conference on Artificial Intelligence*, Vol. 35. 11272–11281.

[12] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[13] Jonathan Morag, Ariel Felner, Roni Stern, Dor Atzmon, and Eli Boyarski. 2022. Online multi-agent path finding: New results. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 15. 229–233.

[14] Jonathan Morag, Noy Gabay, Daniel Koyfman, and Roni Stern. 2025. Should Multi-Agent Path Finding Algorithms Coordinate Target Arrival Times?. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 18. 231–235.

[15] Bernhard Nebel. 2020. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30. 212–216.

[16] Ayano Okoso, Keisuke Otaki, Satoshi Koide, and Tomoki Nishi. 2022. High Density Automated Valet Parking via Multi-agent Path Finding. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2146–2153.

[17] Keisuke Okumura. 2023. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 11655–11662.

[18] Keisuke Okumura. 2024. Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum (Eds.). 1501–1509.

[19] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence* 310 (2022), 103752.

[20] Keisuke Okumura and Hiroki Nagai. 2025. Lightweight and Effective Preference Construction in PIBT for Large-Scale Multi-Agent Pathfinding. *arXiv preprint arXiv:2505.12623* (2025).

[21] Arseniy Pertzovsky, Roni Stern, and Roie Zivan. 2024. CGA: Corridor Generating Algorithm for Multi-Agent Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3455–3462.

[22] 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.

[23] Yaakov Sherma, Eyal Weiss, and Oren Salzman. 2025. From Agent Centric to Obstacle Centric Planning: A Makespan-Optimal Algorithm for the Multi-Agent Warehouse Rearrangement Problem. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 18. 136–144.

[24] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Symposium on Combinatorial Search (SoCS)*.

[25] Pavel Surynek. 2010. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 24. 1261–1263.

[26] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. 2019. Online multi-agent pathfinding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 7732–7739.

[27] Jiaqi Tan, Yudong Luo, Jiaoyang Li, and Hang Ma. 2025. Reevaluation of Large Neighborhood Search for MAPF: Findings and Opportunities. *Proceedings of the International Symposium on Combinatorial Search* 18, 1 (Jul. 2025), 212–220. https://doi.org/10.1609/socs.v18i1.35996

[28] David Vainshtain and Oren Salzman. 2021. Multi-agent terraforming: Efficient multi-agent path finding via environment manipulation. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 12. 239–241.

[29] Yutong Wang, Tanishq Duhan, Jiaoyang Li, and Guillaume Adrien Sartoretti. 2025. LNS2+RL: Combining Multi-agent Reinforcement Learning with Large Neighborhood Search in Multi-agent Path Finding. In *AAAI Conference on Artificial Intelligence*.

[30] Jingjin Yu and Steven LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 27. 1443–1449.

[31] Yue Zhang, Zhe Chen, Daniel Harabor, Pierre Le Bodic, and Peter J Stuckey. 2024. Planning and execution in multi-agent path finding: models and algorithms. In *International Conference on Automated Planning and Scheduling*. 707–715.