Which MAPF Model Works Best for Automated Warehousing?

Sumanth Varambally,[1] Jiaoyang Li,[2] Sven Koenig[2]

[1] Indian Institute of Technology, Delhi
[2] University of Southern California
Sumanth.Varambally.mt617@maths.iitd.ac.in, {jiaoyanl, skoenig}@usc.edu

Abstract

Multi-Agent Path Finding (MAPF) algorithms and their variants can find high-quality collision-free plans for automated warehousing under simplified assumptions about the robot dynamics. However, these simplifying assumptions pose challenging implementational issues as the robots cannot follow the plans precisely. Various robust execution frameworks, such as the Action Dependency Graph (ADG) framework, have been proposed to enable the real-world execution of MAPF plans. Under such a framework, it is unclear how the simplifying assumptions affect the performance of the robots. In this work, we first argue that the ADG framework provides the same robustness guarantees as the single-agent framework (where plans are generated independently for each robot and collisions are avoided through a reservation table), which is widely used in industry. We then improve the efficiency of the ADG framework by integrating it with the Rolling-Horizon Collision-Resolution framework to solve MAPF problems with a persistent stream of online tasks. Using the integrated framework, we compare the standard MAPF model with many of its more complex variants, such as MAPF with rotation, k-robust MAPF, and continuous-time MAPF (taking robot dynamics into account). We examine their effectiveness in improving throughput through realistic simulations of warehouse settings with the Gazebo simulator.

1 Introduction

Multi-Agent Path Finding (MAPF) (Stern et al. 2019) is the problem of planning collision-free paths that move a team of agents in discretized timesteps to their predefined goal vertices on a given graph while minimizing their travel times. Despite its intractability (Yu and LaValle 2013; Ma et al. 2016), a significant amount of effort has been spent in recent years on solving this problem more efficiently. For instance, researchers have improved the scalability of optimal MAPF algorithms in the past decade by one order of magnitude in terms of agents (Lam and Bodic 2020; Li et al. 2021b) from initially roughly less than a dozen of agents (Standley 2010; Wagner and Choset 2011; Sharon et al. 2012). Sub-optimal MAPF algorithms can find near-optimal plans for thousands of agents (Wang and Rubenstein 2020; Li et al. 2022). Due to the success in improving the scalability of MAPF, researchers have applied MAPF technologies to a

(a) Kiva. (b) Sortation. (c) AutoStore.

Figure 1: Photos of different types of warehouses.[1]

wide range of multi-robot systems, including UAV traffic management (Ho et al. 2019), railway planning (Li et al. 2021a), and airport surface operation (Li et al. 2019).

Among the various applications, coordinating a large team of mobile robots in automated warehouses (see examples in Figure 1) receives the most attention as the setup of a warehouse is usually close to many assumptions used in the traditional MAPF literature (e.g., a large team of agents is controlled centrally in a known environment represented by a grid map). Research includes how to adapt MAPF to a lifelong setting, where agents are assigned new goal cells after they reach their current ones (Li et al. 2021c; Damani et al. 2021), and how to combine the goal-vertex assignment with MAPF (Ma et al. 2017; Nguyen et al. 2017; Liu et al. 2019; Kou et al. 2020; Chen et al. 2021b; Contini and Farinelli 2021). These works look promising as they usually scale well (e.g., can handle hundreds or even thousands of agents in highly congested environments) and produce a high throughput in simulated warehouse environments.

Unfortunately, they cannot be directly applied in the real world as they make many simplified and thus unrealistic assumptions on the robot dynamics and actuator uncertainty. Therefore, recent research has focused on more complicated MAPF models to close the gap, such as MAPF with non-unit travel times (Walker, Sturtevant, and Felner 2020; Walker et al. 2021; Andreychuk et al. 2021), kinematic constraints (Le and Plaku 2018; Cohen et al. 2019; Andreychuk 2020; Solis et al. 2021; Chen et al. 2021a), and time uncertainty (Ma, Kumar, and Koenig 2017; Atzmon et al. 2020b,a; Shahar et al. 2021). These MAPF models are more applica-

[1]Figures (a) and (b) are from (Wurman, D'Andrea, and Mountz 2007; Kou et al. 2020), respectively. Figure (c) is by Euro-Friwa GmbH from https://commons.wikimedia.org/w/index.php?curid=62341670 licensed under CC BY-SA 4.0.

ble in the real world, but they are also harder to solve. Moreover, these models cannot be directly applied since there are still gaps between them and the real world.

Rather than building perfect MAPF models to close the gap, Hönig et al. (2019) proposed an execution framework that modifies the MAPF plans in real-time by delaying the robots intelligently when necessary to overcome the imperfection of the MAPF model. It achieves this by converting a MAPF plan to an *action dependency graph* (ADG) and making the robots execute the actions following the precedence constraints indicated by the ADG. This ADG framework guarantees collision-freeness, deadlock-freeness, and persistence over a long time horizon. It also allows for replanning during execution, which avoids robot idle time during replanning.

In this work, we first argue that the ADG framework provides the same robustness guarantees as the single-agent framework (Wang and Botea 2008), which is widely used in industry. We then combine the ADG framework with Rolling-Horizon Collision-Resolution (RHCR) (Li et al. 2021c), an efficient lifelong MAPF framework that repeatedly uses windowing (i.e., a limited planning horizon) to speed up the search. We call the resulting framework Rolling-Horizon Collision-Resolution and Execution (RHCRE). The efficiency of RHCRE allows us to try various MAPF models and study their runtime and quality tradeoffs. Intuitively, a more accurate MAPF model requires the ADG to delay fewer actions during execution, resulting in higher throughput. In particular, we consider four different MAPF models, namely standard MAPF (Stern et al. 2019); MAPF with rotations (Hönig et al. 2019), that considers the rotation time of the robots; k-robust MAPF (Atzmon et al. 2020b), that guarantees the solution to be robust to bounded time uncertainties; and continuous-time MAPF (Andreychuk et al. 2021), that models actions with non-unit travel times.

We use two MAPF algorithms, ECBS (Barer et al. 2014) and prioritized planning (Silver 2005) to experiment with these MAPF models using the Gazebo simulator (Koenig and Howard 2004). We find that windowing and considering rotation time results in substantial improvements in throughput. k-robust MAPF plans improve throughput in some instances. Planning in continuous time with prioritized planning that takes the exact execution time of different actions (based on the robot dynamics) into account achieves the best throughput among all considered algorithms.

2 Preliminaries

Today, hundreds of robots already navigate autonomously inside warehouses to move inventory shelves or flat packages. Figure 1a shows a Kiva warehouse (Wurman, D'Andrea, and Mountz 2007) whose example layout is shown in Figure 2, where robots move the inventory shelves from the storage locations to the work stations and human workers pick ordered items from the shelves. By moving the inventory to the worker, rather than the other way around, the worker productivity at least doubles (Wurman, D'Andrea, and Mountz 2007). In addition to Kiva-like warehouses, other popular warehouse structures include sortation centers (Kou et al. 2020), as shown in Figure 1b, where human workers put packages on the robots and the robots move the packages from the work stations to the corresponding chutes (which correspond to the loading docks that the packages have to be delivered to), and AutoStore,[2] as shown in Figure 1c, where robots navigate on grid-like tracks on top of stacks of bins and dig out needed bins from the top.

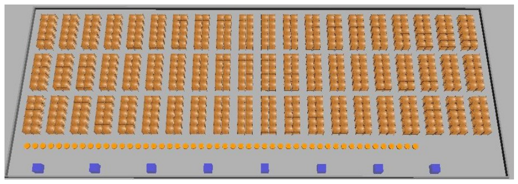


Figure 2: Snapshot of the Gazebo simulator with 600 shelves (yellow squares), 50 robots (yellow circles), and 8 stations (blue squares).

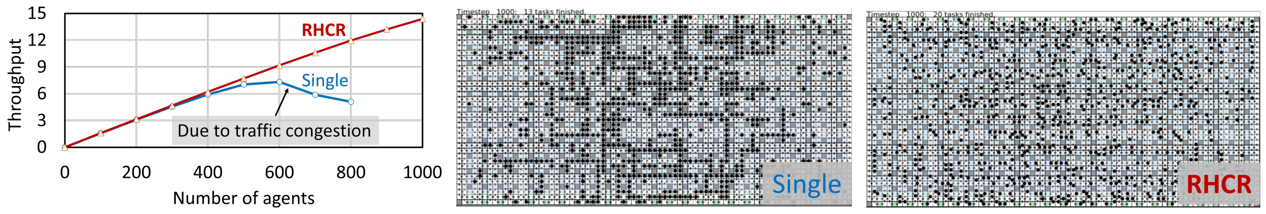
Although these warehouses have different designs with different target application scenarios, they share many similarities: (1) the robots all navigate on a 4-neighbor grid; (2) they use similar motion models; (3) each robot needs to visit a stream of assigned goal cells without colliding with other robots; and (4) each robot receives commands to execute actions from a central controller. We therefore provide a formal definition of the graph that the robots navigate on, the agent and robot models that we use in this paper, and MAPF.

Definition 2.1 (Graph)**.** We use a *graph* $G = (V, E)$ to represent the work space of the warehouse, which is a 4-neighbor grid with V representing empty cells on the grid and E representing the connections between empty cells that the agents/robots can use to move from cell to cell.

Definition 2.2 (Agent)**.** We refer to an *agent* as an abstract robot model used by the MAPF algorithms during planning. Unless specified otherwise, we follow the agent model widely used in the MAPF literature, where every agent always occupies one cell at each timestep and moves in synchronized discretized timesteps with the other agents. It can perform two types of actions, both of which consume one timestep: A move action moves the agent from its current cell to a neighboring cell, and a wait action keeps the agent at its current cell. Two agents collide when they try to occupy the same cell (called a vertex collision) or swap their positions at neighboring cells (called an edge collision) at the same timestep.

Definition 2.3 (Robot)**.** We refer to a *robot* as a realistic robot model that reflects the physical reality of mobile robots in warehouses and is used by our robot simulator during execution. Each cell $v \in V$ is large enough to contain at least one robot. Robots move in continuous time asynchronously. Each robot can receive four action commands from the central controller, namely move forward to the next cell, wait at its current cell, turn-in-place by 90 degrees left, and turn-in-place by 90 degrees right, and executes its actions independently and autonomously using an onboard PID controller. It uses a command queue to maintain the received action commands and can combine sequential actions in its queue

[2]https://www.autostoresystem.com/



(a) Throughput comparison. (b) Single-agent framework. (c) Multi-agent framework.

Figure 3: Comparison of the MAPF framework RHCR (Li et al. 2021c) with the single-agent framework Single (Wang and Botea 2008) for a sortation center (as illustrated in Figure 1b). (a) shows the throughput (= average number of reached goal cells per timestep), and (b) and (c) show a snapshot of the simulator using Single and RHCR, respectively, with 800 agents (represented by the black dots). This simulator follows the setup in (Li et al. 2021c), where a stream of goal cells (as opposed to a single goal cell) is assigned to each agent. The movement of the robots is simulated with the agent model in Definition 2.2 instead of a realistic robot model.

(e.g., several actions of moving forward by one cell into one action of moving forward by several cells) in order to perform smooth, continuous, and effective motions. It stands still when its command queue is empty. As a result, the execution time of the actions is stochastic. The robot signals the central controller when it has finished executing an action. A robot can localize itself in the warehouse, for example, via the fiducials on the floor, and ensure that it never diverges significantly from its assigned path spatially.

Definition 2.4 (MAPF)**.** The Multi-Agent Path Finding (MAPF) problem takes as input a graph (defined in Definition 2.1) and a set of agents (defined in Definition 2.2), each with a start cell and a goal cell. A *path* for an agent is a sequence of actions that moves it from its start cell to its goal cell, with the path cost being the arrival time of the agent at its goal cell. The agent remains at its goal cell forever after it completes its path. A *solution*, which we also call a *MAPF plan*, is a set of collision-free paths, one for each agent. The cost of a solution is the sum of the path costs of the agents.

3 Execution Frameworks

We introduce two distinct execution frameworks (used by the central controller) to coordinate the warehouse robots. The first framework is based on single-agent path finding and widely used in industry. The second framework is based on MAPF and shows promise in producing high throughput.

3.1 Single-Agent Execution Framework

Traditional single-agent frameworks approach the warehouse robot coordination problem from a control engineering perspective by employing a reactive paradigm to collision avoidance. Each robot follows a shortest (or minimum-estimated-travel-time) path to its goal cell, planned by a single-agent path finding algorithm, such as A*. Robot-robot collisions are resolved locally during execution via manually-designed traffic rules, the most popular one of which is a reservation system (Wang and Botea 2008).

A reservation system allows the robots to reserve the cells ahead of time that they are going to use. The reservation system keeps a reservation map that maps every cell to either zero or one, indicating that the cell is empty or reserved, respectively. Each robot needs to reserve a cell (or a fixed number of cells) on its path, usually in a first-come-first-served manner, before it can move there. A cell can be reserved only if its current value in the reservation map is zero. The reservation is released once the robot leaves the cell.

A reservation system is robust to problematic scenarios, such as agents moving faster or slower than expected or even breaking down and stopping forever, because it ensures that two robots will never be at the same cell and thus never collide. Due to these strong robustness guarantees and its simple implementation, the single-agent framework is widely used in industry. However, it is prone to congestion issues as the robot density increases. This limits the number of robots one can put in a warehouse and the resulting throughput.

3.2 MAPF Execution Framework

Recent advances in MAPF show the promise of using MAPF algorithms to coordinate warehouse robots: They can coordinate robots with less congestion and higher throughput than the single-agent framework, as shown in Figure 3.

A MAPF-based robot-coordination system first calls a MAPF algorithm to search for a set of globally coordinated collision-free paths, one for each robot, and then asks the robots to start following the paths. The main drawback of the deliberative paradigm is that the robots are guaranteed to reach their goal cells only if they follow the planned paths precisely both in space (which is typically easy to achieve) and time (which is typically more difficult to achieve).

Several MAPF execution frameworks (Cáp, Gregoire, and Frazzoli 2016; Hönig et al. 2016; Ma, Kumar, and Koenig 2017; Coskun and O'Kane 2019) have been proposed to mitigate this issue and allow the robots to follow their paths without collision. These frameworks do not change the agent models used by the MAPF algorithms. For example, the ADG framework (Hönig et al. 2019) is such a framework specifically designed for warehouse applications. It ensures the collision-free and deadlock-free execution of MAPF plans by imposing an order on several robots visiting the same cell. More details will be provided in Section 5.2.

We argue that the ADG framework provides the same robustness against robot collisions and deadlocks as the single-agent framework, regardless of the accuracy of the agent model used by the MAPF algorithms. The precedence rules enforced by the ADG framework guarantee that a robot can never move to a cell before the robot occupying it currently vacates it. This guarantees the collision-free execution of MAPF plans despite robots moving faster or slower than intended or breaking down and stopping forever.

4 Modeling and Solving MAPF

The ADG framework allows us to use an agent model of our choice. However, in practice, there is a trade-off. Intuitively, a more accurate agent model requires the ADG to delay fewer actions during execution, resulting in a better solution quality. On the other hand, finding collision-free paths for a more accurate agent model typically takes more time. In addition, the solution quality and runtime also depend on the MAPF algorithm used. Therefore, in this section, we first provide an overview of existing search-based MAPF algorithms and then an overview of four different agent models.

4.1 Overview of MAPF Algorithms

We summarize search-based MAPF algorithms in different categories with a focus on how they have been generalized to handle more realistic agent models.

A*-Based Algorithms. A naïve way to solve MAPF optimally is to apply A* to the joint state space of the agents, where a joint state is a list of different cells, one for each agent, that represents the positions of all agents at some timestep. Examples include EPEA* (Goldenberg et al. 2014) and M* (Wagner and Choset 2011). However, due to the limited scalability of the A*-based algorithms and their requirement of synchronized actions, only limited work has been carried out on generalizing such algorithms to more realistic agent models. One example is UM* (Wagner and Choset 2017), a variant of M* that deals with uncertainty caused by unmodeled dynamics and localization and sensing errors.

CBS-Based Algorithms. CBS (Sharon et al. 2012) solves MAPF optimally in the collision-resolution space. It plans paths for agents independently first (by ignoring their collisions with other agents) and resolves the resulting collisions afterward via a best-first search on a binary tree, called a constraint tree. Significant progress has been made on speeding up CBS: its optimal and bounded-suboptimal variants can plan paths for a few hundred and a thousand agents within a minute, respectively (Li et al. 2021b; Li, Ruml, and Koenig 2021). Since it finds paths in the single-agent state space, it can be easily generalized to different agent models. For instance, researchers have developed CBS variants for all agent models discussed in Section 4.2. Therefore, we select a bounded-suboptimal CBS variant, called ECBS (Barer et al. 2014), as one of the MAPF algorithms for evaluating different agent models in our experiments.

Prioritized Algorithms. Prioritized planning (Erdmann and Lozano-Perez 1987) is a simple and fast but incomplete and suboptimal MAPF algorithm. It uses a predefined priority ordering on the agents and plans paths for the agents one by one from the highest-priority agent to the lowest-priority agent. Each time, it plans a path for an agent that does not collide with the already planned paths of all higher-priority agents. Specifically, it records the cells and edges occupied by all higher-priority agents at each timestep in a reservation table and calls A* to find the shortest path for the agent that does not use any reserved resources. Like CBS-based algorithms, it finds paths in the single-agent state space and thus can be easily generalized to different agent models. Therefore, we select prioritized planning with random restarts (in the following called PP), which repeatedly calls prioritized planning with a randomly generated priority ordering until a solution is found, as one of the MAPF algorithms for evaluating different agent models in our experiments.

4.2 Overview of Agent Models

Standard MAPF. Definition 2.4 defines standard MAPF. We use ECBS and PP in our experiments to solve it, both of which use space-time A* (Silver 2005) to plan paths, i.e., A* searches in the single-agent state space where each state (v, t) is a pair of a cell $v \in V$ and a timestep $t \in \mathbb{N}$.

MAPF with Rotation. Standard MAPF assumes four possible move actions, each corresponding to a movement in one of the four directions: move north, south, east, or west. Thus, standard MAPF does not take the rotation time of the agent, if any, into account. In MAPF with rotation (Hönig et al. 2019), we use the following move actions instead: move forward, rotate 90° left, rotate 90° right, and rotate 180°. Each move action still takes one timestep to execute. This approach explicitly models the rotation time, but it increases the time needed to solve MAPF because the single-agent state space is larger – each state is now a triple (v, t, θ) that contains an orientation $\theta \in \{north, south, east, west\}$ in addition to a cell and a timestep.

MAPF with Time Uncertainty. Standard MAPF unrealistically assumes that the robots move deterministically. Considerable effort has gone into ensuring that MAPF plans are robust to stochastic delays during execution (Cáp, Gregoire, and Frazzoli 2016; Ma, Kumar, and Koenig 2017; Wagner and Choset 2017; Li et al. 2019; Coskun and O'Kane 2019; Atzmon et al. 2020a; Street et al. 2020). Atzmon et al. (2020b) propose k-robust MAPF, where a MAPF plan is collision-free even if each agent can be delayed by up to k timesteps. This is done by ensuring that $|t_i - t_j| > k$ for any agents i and j that visit the same cell at timesteps t_i and t_j respectively. If this condition is not satisfied, then the two agents are said to be in a vertex collision, which is resolved in the same manner as done traditionally in MAPF algorithms. (The definition of edge collisions remains the same.) For example, CBS-based algorithms resolve the collision via a best-first search on the constraint tree, while prioritized algorithms record an occupied cell not only at the timestep it is occupied but also for $k - 1$ additional timesteps after its occupation. This approach increases the time needed to solve MAPF: CBS-based algorithms need to resolve more collisions than their non-robust variants, and prioritized algorithms are more prone to failure than their non-robust counterparts since the cells are occupied by the higher-priority agents for a longer time.

Continuous-Time MAPF. Standard MAPF assumes that time is discretized and each action is always completed in one timestep. In continuous-time MAPF (Andreychuk et al. 2021), we drop this assumption and allow actions to take an arbitrary amount of time to complete. Such models can be used to express low-level details about the agent dynamics, such as the exact execution time of move actions. Since space-time A* requires discrete timesteps, we replace it with SIPP (Phillips and Likhachev 2011), an advanced variant of space-time A* that allows for real-valued action execution times. Since continuous-time CBS-based algorithms are significantly less efficient than their discretized-time variants (Andreychuk et al. 2021), we use only PP for this model.

5 From MAPF to Automated Warehousing

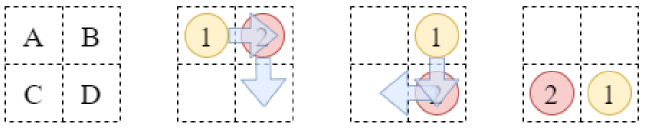
In addition to the imperfection of the agent models used in MAPF, we need to address the online nature of the coordination of warehouse robots. MAPF is an offline problem where each agent is pre-assigned exactly one goal cell and does not start to move until a set of collision-free paths for all agents is found. This raises two issues, namely, how to generalize MAPF to lifelong MAPF with streams of newly arriving goal cells and how to overcome the computational overhead of MAPF algorithms for real-time robot operations. To this end, we introduce our new algorithm RHCRE (see Section 5.3) that combines two existing algorithms, namely RHCR introduced in Section 5.1 and the ADG framework introduced in Section 5.2. The MAPF algorithms used in the RHCR and ADG frameworks use the standard MAPF model and the MAPF with rotation model, respectively. That is, they both assume discretized timesteps and no time uncertainty. We follow their convention and use "timesteps" in the following subsections. However, all three algorithms can be used for continuous-time MAPF with minimal changes.

5.1 Solving Lifelong MAPF

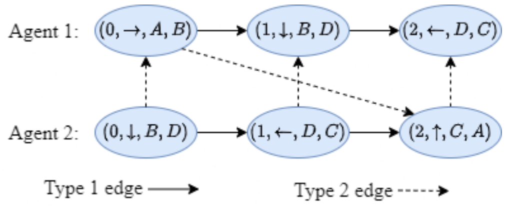
Standard MAPF is a "one-shot" version of the warehouse robot coordination problem where each agent is assigned only one single-start-single-goal navigation task and stays at its goal cell forever after it completes its task. The warehouse robot coordination problem is a lifelong MAPF problem with a continuous stream of navigation tasks that arrive on-the-fly and that agents must address as soon as they complete their currently assigned tasks. The difficulty in adapting MAPF algorithms to the lifelong setting is due to MAPF planning being carried out jointly for all agents; yet, not all agents finish their currently assigned tasks simultaneously.

One can address this issue in several ways. For instance, one can decompose a lifelong MAPF instance into a sequence of MAPF instances where one replans paths at every timestep for all agents (Grenouilleau, van Hoeve, and Hooker 2019); however, this approach does not scale to large numbers of agents. One can also decompose a lifelong MAPF instance into a sequence of MAPF instances where one plans new paths at every timestep for only the agents with newly assigned goal cells; however, this approach can only be applied to particular types of graphs without losing its completeness guarantee. Another drawback of the above approaches is that they need to call MAPF algorithms at every timestep, which may not meet the real-time requirement.

Rolling Horizon Collision Resolution (RHCR) (Li et al. 2021c) effectively combats the above issues by using windowing. It decomposes a lifelong MAPF instance into a sequence of *windowed* MAPF instances where one replans paths with a predefined frequency for all agents but resolves collisions only for the first several timesteps within a predefined time window. Specifically, RHCR uses two hyperparameters, namely a window size w and a replanning period h. It calls a MAPF algorithm every h timesteps to replan the paths of all agents. In each call, it resolves collisions only for the first w timesteps, with subsequent collisions being ignored, which can substantially reduce the computational time. Naturally, we need $w \geq h$ to ensure collision-freeness.



(a) MAPF plan for two agents on a 2×2 grid.



(b) Corresponding ADG Graph.

Figure 4: ADG example. The arrows in (b) indicate the precedence of actions, i.e., the execution of an action that an arrow points to can begin only after the action that the arrow points away from is completed.

5.2 Overlapping Planning and Execution

The Action Dependency Graph (ADG) framework (Hönig et al. 2019) is a robust MAPF execution framework that uses an ADG to ensure the collision-free and deadlock-free execution of MAPF plans and to prevent robots from waiting for replanning to finish and thus from being idle.

Definition 5.1 (ADG)**.** An Action Dependency Graph (ADG) is a directed acyclic graph $\mathcal{G}_{\text{ADG}} = (\mathcal{V}_{\text{ADG}}, \mathcal{E}_{\text{ADG}})$ that represents the action-precedence relationships of a given MAPF plan. A node $p_n^i \in \mathcal{V}_{\text{ADG}}$ corresponds to the n^{th} action in the path of robot i. More specifically, p_n^i is a 4-tuple $(t_n^i, a_n^i, s_n^i, g_n^i)$ that indicates the n^{th} action a_n^i executed starting at timestep t_n^i, taking robot i from cell $s_n^i \in V$ to cell $g_n^i \in V$. An edge $(p_n^i, p_{n'}^{i'}) \in \mathcal{E}_{\text{ADG}}$ indicates that robot i' can only begin executing action $a_{n'}^{i'}$ after robot i completes the execution of action a_n^i. Type 1 edges are edges that connect actions of the same robot. Type 2 edges are edges that connect actions of different robots, ensuring that the order in which robots visit the same cell does not change.

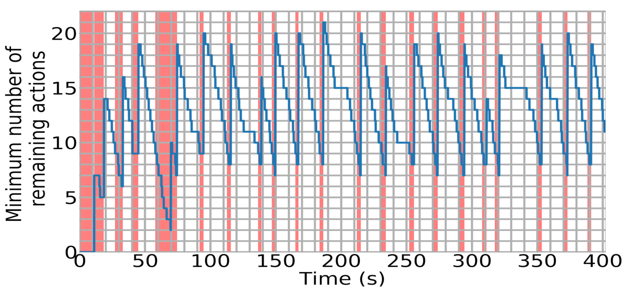


Figure 5: Minimum number of remaining actions over all robots in their command queues over time. The red bars indicate the time during which planning takes place. The minimum number of remaining actions never reaches 0, indicating that the robots are never idle, even during replanning.

We therefore add a Type 2 edge from node p_n^i to node $p_{n'}^{i'}$ iff robot i visits a certain cell before robot i' (or, more specifically, $s_n^i = g_{n'}^{i'}$ and $t_n^i \leq t_{n'}^{i'}$). Figure 4 shows an example.

Execution. Each robot maintains a command queue during execution (recall Definition 2.3). Our task is to enqueue actions into the command queue in a timely manner so that the robots always have actions to execute while ensuring that no deadlocks or collisions can happen during execution. We enqueue an action a_n^i (with $p_n^i \in \mathcal{V}_{\text{ADG}}$) into the command queue of robot i when action a_{n-1}^i is already enqueued and the execution of all actions whose corresponding nodes are connected to node p_n^i via Type-2 edges is finished. An action is marked as finished once the robot notifies the central controller of the successful execution of the action.

Replanning. A significant benefit that stems from encoding MAPF plans in this manner is the ability to overlap planning and execution. When a robot has fewer actions left to execute than a given threshold, replanning is triggered. Every robot i picks one of its future nodes $p_{n_i}^i \in \mathcal{V}_{\text{ADG}}$ to be the last node[3] in the ADG that it will execute and discards all nodes afterward (i.e., all nodes p_n^i with $n > n_i$). The next round of planning commences with the start cell of each agent i being $g_{n_i}^i$. A MAPF algorithm is then called to replan paths for all agents from these start cells to their goal cells, assuming that the agents will be at these start cells at timestep 0. While the MAPF algorithm is running, the robots keep moving. When the MAPF algorithm terminates, the found MAPF plan is converted to a new ADG and appended to the current one. In this manner, we can overlap planning and execution, as illustrated in Figure 5.

5.3 RHCRE

The ADG framework calls a MAPF algorithm to plan collision-free paths for all agents. We improve it by incorporating the windowing idea from RHCR into it. During replanning, we call a windowed MAPF algorithm to plan paths that are collision-free for only the first w timesteps, where w is a predefined parameter. Then, instead of converting the entire MAPF plan to an ADG, we only convert the part of the MAPF plan that contains the actions from the first w timesteps. We call the resulting algorithm Rolling-Horizon Collision Resolution and Execution (RHCRE).

In addition, the procedure for determining the last nodes in (Hönig et al. 2019) has a corner-case problem since the last nodes of two different agents can result in the same cell and thus a collision at timestep 0 when we replan paths for them. This can happen for both the ADG framework and RHCRE. We address this issue by explicitly checking for such a situation and, if necessary, advancing the last node for one of the agents to its next node in the ADG.

During replanning, RHCRE calls a MAPF algorithm that always considers exactly the next goal cell for each agent. This is different from the original RHCR (Li et al. 2021c), which can ask the MAPF algorithm to consider the next plus one or more of the subsequent goal cells for each agent in order to prevent any agent from reaching its goal cell within h timesteps and then being idle. RHCRE does not need to worry about this issue because it replans when a robot might run out of actions to execute. That is, its replanning period h is adaptive instead of being a fixed value.

6 Empirical Evaluation

We follow the experimental setup in (Hönig et al. 2019) and use the Gazebo simulator (Koenig and Howard 2004) to simulate a Kiva warehouse on map `small1` from Amazon's HARMONIES simulator, containing 600 shelves, 50 robots (based on the iRobot Create 2 robots), and 8 stations (shown in Figure 2). A *job* describes the task of a robot picking up a shelf from the warehouse floor, delivering it to a station where it waits for a few seconds (during which the worker presumably picks items off the shelf), and returning it to a potentially different location in the warehouse. Therefore, in addition to the navigation-related actions introduced in Definition 2.3, the robots can perform three additional warehouse-specific actions when they wait, namely attaching themselves to a shelf, detaching themselves from a shelf, and waiting at a station. More details can be found in (Hönig et al. 2019).

We use three MAPF algorithms, namely ECBS, PP, and Continuous PP (i.e., PP with SIPP for its low-level search). In order to allow RHCRE to work properly, we also need a task assigner that assigns the jobs to the robots. For ECBS, we follow (Hönig et al. 2019) and use ECBS-TA (Hönig et al. 2018) instead of ECBS, a variant of ECBS that simultaneously assigns tasks and plans collision-free paths in a way so that the costs of its solutions are at most a factor of ϵ larger than optimal. We use $\epsilon = 2$ in our experiments. For PP and Continuous PP, we assign tasks with the task-assignment algorithm in ECBS-TA, which produces optimal assignments (if one ignores robot-robot collisions).

Our implementation is based on (Hönig et al. 2019) and written in C++. We use AWS EC2 instances to run our experiments. The server program (running the Gazebo simulator) is run on `m4.xlarge` instances, while the client pro-

[3]The last nodes are chosen so that their end cells $g_{n_i}^i$ can be reached simultaneously without violating the precedences indicated by the ADG, and the expected execution time of the remaining actions is greater than the expected runtime of the MAPF algorithm (to avoid idle time). See (Hönig et al. 2019) for details.

	ECBS-TA, $k = 0$		**ECBS-TA,** $k = 1$		**ECBS-TA with rot.,** $k = 0$		**ECBS-TA with rot.,** $k = 1$	
w	**#Replans**	**Throughput**	**#Replans**	**Throughput**	**#Replans**	**Throughput**	**#Replans**	**Throughput**
12	42.78	$0.235 \pm 3.5\%$	49.67	$\mathbf{0.246 \pm 0.5}\%$	50.33	$0.257 \pm 1.0\%$	51.67	$\mathbf{0.286 \pm 3.8}\%$
15	44.75	$0.230 \pm 2.4\%$	45.33	$0.240 \pm 1.2\%$	46.00	$0.251 \pm 1.7\%$	46.33	$0.271 \pm 6.0\%$
17	43.57	$0.232 \pm 1.6\%$	44.00	$0.233 \pm 1.9\%$	44.00	$0.245 \pm 2.1\%$	44.00	$0.263 \pm 0.2\%$
20	43.04	$0.238 \pm 1.9\%$	43.00	$0.227 \pm 2.2\%$	42.00	$0.238 \pm 3.3\%$	42.33	$0.256 \pm 2.4\%$
22	41.26	$0.233 \pm 1.8\%$	41.75	$0.227 \pm 1.3\%$	39.67	$0.233 \pm 2.0\%$	42.67	$0.258 \pm 2.7\%$
25	39.91	$0.221 \pm 3.5\%$	40.33	$0.217 \pm 2.9\%$	40.67	$0.233 \pm 1.7\%$	42.67	$0.269 \pm 2.6\%$
∞	40.25	$0.225 \pm 3.7\%$	41.00	$0.226 \pm 4.7\%$	38.00	$0.229 \pm 2.5\%$	42.56	$0.262 \pm 2.6\%$

Table 1: Number of replans and throughput for ECBS-TA.

	PP, $k = 0$		**PP,** $k = 1$		**PP with rot.,** $k = 0$		**PP with rot.,** $k = 1$	
w	**#Replans**	**Throughput**	**#Replans**	**Throughput**	**#Replans**	**Throughput**	**#Replans**	**Throughput**
12	47.00	$0.243 \pm 1.1\%$	50.33	$0.241 \pm 3.5\%$	56.00	$\mathbf{0.283 \pm 2.7}\%$	57.33	$0.276 \pm 3.5\%$
15	44.33	$0.242 \pm 1.9\%$	45.33	$0.236 \pm 2.1\%$	50.00	$0.272 \pm 3.1\%$	52.33	$0.277 \pm 3.1\%$
17	44.33	$0.242 \pm 2.8\%$	44.33	$0.235 \pm 2.2\%$	46.33	$0.258 \pm 5.4\%$	50.00	$0.272 \pm 1.6\%$
20	44.00	$0.238 \pm 0.2\%$	43.33	$0.229 \pm 2.8\%$	45.33	$0.254 \pm 1.4\%$	47.00	$0.260 \pm 1.7\%$
22	43.33	$0.240 \pm 3.2\%$	43.00	$0.228 \pm 4.0\%$	46.00	$0.261 \pm 4.5\%$	46.33	$0.260 \pm 4.5\%$
25	43.00	$\mathbf{0.245 \pm 5.8}\%$	45.00	$0.237 \pm 1.5\%$	43.33	$0.258 \pm 3.4\%$	42.67	$0.252 \pm 5.5\%$
∞	41.33	$0.232 \pm 2.7\%$	43.67	$0.233 \pm 2.5\%$	43.00	$0.255 \pm 4.3\%$	43.33	$0.257 \pm 3.4\%$

Table 2: Number of replans and throughput for PP.

	Continuous PP with rot., $k = 0.0s$		**Continuous PP with rot.,** $k = 0.5s$		**Continuous PP with rot.,** $k = 1.0s$	
w	**#Replans**	**Throughput**	**#Replans**	**Throughput**	**#Replans**	**Throughput**
12s	68.67	$0.299 \pm 1.6\%$	72.00	$\mathbf{0.305 \pm 2.0}\%$	73.33	$0.302 \pm 1.6\%$
15s	60.00	$0.295 \pm 1.5\%$	62.50	$0.298 \pm 2.0\%$	63.67	$0.289 \pm 2.8\%$
17s	55.67	$0.280 \pm 2.7\%$	57.33	$0.288 \pm 2.4\%$	59.33	$0.285 \pm 2.5\%$
20s	51.67	$0.274 \pm 1.0\%$	52.50	$0.272 \pm 2.2\%$	53.75	$0.269 \pm 2.7\%$
22s	49.00	$0.268 \pm 2.6\%$	49.67	$0.266 \pm 3.5\%$	51.67	$0.262 \pm 1.7\%$
25s	47.33	$0.258 \pm 0.7\%$	47.00	$0.253 \pm 0.9\%$	47.33	$0.248 \pm 2.9\%$
30s	43.67	$0.242 \pm 5.4\%$	44.00	$0.241 \pm 2.2\%$	44.00	$0.243 \pm 2.6\%$
35s	43.67	$0.250 \pm 1.3\%$	40.75	$0.232 \pm 4.7\%$	41.33	$0.225 \pm 5.0\%$
∞	35.33	$0.211 \pm 2.8\%$	36.33	$0.217 \pm 4.5\%$	33.00	$0.197 \pm 3.7\%$

Table 3: Number of replans and throughput for Continuous PP.

gram is run on `m4.large` instances. Due to computational constraints, the Gazebo simulator does not run in real-time. To compensate for this disparity, we delay the communication of the commands from the client to the server by the ratio of the simulation time of the Gazebo simulator and the wall-clock time. We run each setting for 1,000 simulation-time seconds and record the throughput, measured in jobs finished per second. In calculating this quantity, we omit the planning time for generating the first MAPF plan as we are more interested in the long-term performance.

We present the throughputs and numbers of replans for all algorithms in Tables 1 to 3. We also present their average planning time in Table 4. All results are averaged over three runs, with the standard deviation expressed as a percentage of the mean. "Rot." in the tables is short for rotation. We do not present results for Continuous PP without rotations in Table 3 because we designed Continuous PP to take the actual execution times of different actions into account and ignoring the rotation time thus does not make sense.

Effect of Windowing. A smaller window size w tends to result in higher throughput in most cases. For high enough values of w, the throughput plateaus and reaches roughly the same level as without windowing (i.e., $w = \infty$) in most cases. We attribute this to smaller window sizes resulting in shorter MAPF plans to be sent to the ADG each time and thus leading to higher numbers of replans. Frequent replanning corrects the errors caused by the simplifying assumptions made by the MAPF algorithms.

Effect of Rotations. Adding rotations as separate actions and considering their costs during planning improves the throughput of both ECBS-TA and PP, presumably due to the more accurate MAPF model being used.

Effect of k-Robustness. The effect of k-robustness appears to depend on the algorithm. k-robustness improves the throughput of ECBS-TA with rotations by roughly 11% on average, while it leads to mixed results for ECBS-TA without rotations, especially with large window sizes. k-

Algorithm	Rot.	k	w	Planning time (s)
ECBS-TA	✓	0	∞	$12.7 \pm 3.7\%$
			12	$11.9 \pm 0.4\%$
		1	∞	$16.2 \pm 1.5\%$
			12	$14.6 \pm 4.8\%$
	✗	0	∞	$7.2 \pm 2.1\%$
			12	$5.3 \pm 1.2\%$
		1	∞	$7.5 \pm 3.7\%$
			12	$6.4 \pm 1.8\%$
PP	✓	0	∞	$3.5 \pm 4.1\%$
			12	$3.0 \pm 6.7\%$
		1	∞	$4.9 \pm 6.2\%$
			12	$3.9 \pm 3.2\%$
	✗	0	∞	$0.8 \pm 3.3\%$
			12	$0.6 \pm 2.8\%$
		1	∞	$1.3 \pm 14.8\%$
			12	$1.1 \pm 2.2\%$
Continuous PP	✓	0.0s	∞	$4.6 \pm 6.8\%$
			12s	$3.2 \pm 7.0\%$
		0.5s	∞	$4.2 \pm 4.1\%$
			12s	$3.2 \pm 9.0\%$
		1.0s	∞	$5.8 \pm 15.2\%$
			12s	$4.2 \pm 7.3\%$

Table 4: Average planning time.

robustness appears to have a slightly negative effect on the throughput of PP. The throughput of Continuous PP with small window sizes slightly increases from $k = 0s$ to $k = 0.5s$ but decreases from $k = 0.5s$ to $k = 1s$. We attribute this behavior to a trade-off between increasing the throughput due to shorter wait times caused by the Type 2 edges of the ADG and decreasing the throughput due to longer planning times and MAPF plans.

Effect of Using Exact Execution Times. In order to estimate the execution times of different actions, we run predetermined action sequences on small maps and record the average execution time of each action. Based on these measurements, we use $1.250s$ as the time of a forward action, $0.540s$ as the time of a 90° (left or right) turn, $0.956s$ as the time of a 180° turn, $2.000s$ as the time of both attaching to and detaching from a shelf, and $5.000s$ as the time of waiting at a station. Using exact execution times with Continuous PP substantially improves the throughput of the comparable algorithm PP, especially for small window sizes.

Planning Times. Table 4 shows the average planning time of each algorithm. We only include the planning times for the lowest and highest window sizes used, namely $w = 12$ and $w = \infty$. We observe that: (1) Windowing reduces the average planning time. This is expected, since solving windowed problems is easier than solving the original problems, (2) Using rotation actions and k-robustness increases the average planning time. This can be attributed to the resulting increased problem difficulty. (3) Both PP algorithms are significantly faster than ECBS-TA. PP is slightly faster than Continuous PP. We also observe a higher standard deviation for the planning time of the PP algorithms compared to ECBS-TA. A few anomalous runs of the PP algorithms had high planning times and thus idle times between plans for the agents due to the frequent restarts of the planner.

7 Conclusion

We studied the real-world performance of different MAPF models in warehouse settings. We first argued that the MAPF execution frameworks, such as the ADG framework, offer the same robustness guarantees as the single-agent execution frameworks currently used in industry. We incorporated the windowing framework RHCR for lifelong MAPF into the ADG framework and ran extensive simulations on the robot simulator Gazebo to compare different MAPF models in the resulting framework. We observed that: (1) Using windowing and considering rotation time during planning significantly improves throughput in most cases. (2) Using k-robustness boosts the throughput in some cases. (3) Considering the robot dynamics to determine and use the exact execution times of actions during planning achieves the highest throughput when used in conjunction with windowing, rotation, and k-robustness. It is future work to incorporate higher-order dynamics models (Kou et al. 2019; Andreychuk 2020) into our framework.

Acknowledgments

Sumanth performed his research as an IUSSTF-Viterbi summer student at the University of Southern California. The research was supported by the National Science Foundation under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533 as well as a gift from Amazon.

References

Andreychuk, A. 2020. Multi-Agent Path Finding with Kinematic Constraints via Conflict Based Search. In *RCAI*, 29–45.

Andreychuk, A.; Yakovlev, K.; Boyarski, E.; and Stern, R. 2021. Improving Continuous-Time Conflict Based Search. In *AAAI*, 11220–11227.

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020a. Probabilistic Robust Multi-Agent Path Finding. In *ICAPS*, 29–37.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N. 2020b. Robust Multi-Agent Path Finding and Executing. *Journal of Artificial Intelligence Research*, 67: 549–579.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *SoCS*, 19–27.

Cáp, M.; Gregoire, J.; and Frazzoli, E. 2016. Provably Safe and Deadlock-Free Execution of Multi-Robot Plans under Delaying Disturbances. In *IROS*, 5113–5118.

Chen, J.; Li, J.; Fan, C.; and Williams, B. 2021a. Scalable and Safe Multi-Agent Motion Planning with Nonlinear Dynamics and Bounded Disturbances. In *AAAI*, 11237–11245.

Chen, Z.; Alonso-Mora, J.; Bai, X.; Harabor, D. D.; and Stuckey, P. J. 2021b. Integrated Task Assignment and Path Planning for Capacitated Multi-Agent Pickup and Delivery. *IEEE Robotics and Automation Letters*, 6(3): 5816–5823.

Cohen, L.; Uras, T.; Kumar, T. K. S.; and Koenig, S. 2019. Optimal and Bounded-Suboptimal Multi-Agent Motion Planning. In *SoCS*, 44–51.

Contini, A.; and Farinelli, A. 2021. Coordination Approaches for Multi-Item Pickup and Delivery in Logistic Scenarios. *Robotics and Autonomous Systems*, 146: 103871.

Coskun, A.; and O'Kane, J. M. 2019. Online Plan Repair in Multi-robot Coordination with Disturbances. In *ICRA*, 3333–3339.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. PRIMAL$_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

Erdmann, M.; and Lozano-Perez, T. 1987. On Multiple Moving Objects. *Algorithmica*, 2(1-4): 477.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research*, 50: 141–187.

Grenouilleau, F.; van Hoeve, W.; and Hooker, J. N. 2019. A Multi-Label A* Algorithm for Multi-Agent Pathfinding. In *ICAPS*, 181–185.

Ho, F.; Salta, A.; Geraldes, R.; Goncalves, A.; Cavazza, M.; and Prendinger, H. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *AAMAS*, 131–139.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2018. Conflict-Based Search with Optimal Task Assignment. In *AAMAS*, 757–765.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics and Automation Letters*, 4(2): 1125–1131.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *ICAPS*, 477–485.

Koenig, N. P.; and Howard, A. 2004. Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In *IROS*, 2149–2154.

Kou, N. M.; Peng, C.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *AAAI*, 9925–9932.

Kou, N. M.; Peng, C.; Yan, X.; Yang, Z.; Liu, H.; Zhou, K.; Zhao, H.; Zhu, L.; and Xu, Y. 2019. Multi-agent Path Planning with Non-constant Velocity Motion. In *AAMAS*, 2069–2071.

Lam, E.; and Bodic, P. L. 2020. New Valid Inequalities in Branch-and-Cut-and-Price for Multi-Agent Path Finding. In *ICAPS*, 184–192.

Le, D.; and Plaku, E. 2018. Cooperative, Dynamics-based, and Abstraction-Guided Multi-Robot Motion Planning. *Journal of Artificial Intelligence Research*, 63: 361–390.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Repairing Multi-Agent Path Finding via Large Neighborhood Search. In *AAAI*.

Li, J.; Chen, Z.; Zheng, Y.; Chen, S.-H.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021a. Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. In *ICAPS*, 477–485.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021b. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. *Artificial Intelligence*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: Bounded-Suboptimal Search for Multi-Agent Path Finding. In *AAAI*, 12353–12362.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021c. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAAI*, 11272–11281.

Li, J.; Zhang, H.; Gong, M.; Liang, Z.; Liu, W.; Tong, Z.; Yi, L.; Morris, R.; Pasareanu, C.; and Koenig, S. 2019. Scheduling and Airport Taxiway Path Planning under Uncertainty. In *AIAA Aviation Forum*.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *AAMAS*, 1152–1160.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-Agent Path Finding with Delay Probabilities. In *AAAI*, 3605–3612.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *AAMAS*, 837–845.

Ma, H.; Tovey, C. A.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI*, 3166–3173.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized Target Assignment and Path Finding Using Answer Set Programming. In *IJCAI*, 1216–1223.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe Interval Path Planning for Dynamic Environments. In *ICRA*, 5628–5635.

Shahar, T.; Shekhar, S.; Atzmon, D.; Saffidine, A.; Juba, B.; and Stern, R. 2021. Safe Multi-Agent Pathfinding with Time Uncertainty. *Journal of Artificial Intelligence Research*, 70: 923–954.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-Based Search For Optimal Multi-Agent Path Finding. In *AAAI*, 563–569.

Silver, D. 2005. Cooperative Pathfinding. In *AIIDE*, 117–122.

Solis, I.; Motes, J.; Sandström, R.; and Amato, N. M. 2021. Representation-Optimal Multi-Robot Motion Planning Using Conflict-Based Search. *IEEE Robotics and Automation Letters*, 6(3): 4608–4615.

Standley, T. S. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*, 173–178.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–159.

Street, C.; Lacerda, B.; Mühlig, M.; and Hawes, N. 2020. Multi-Robot Planning Under Uncertainty with Congestion-Aware Models. In *AAMAS*, 1314–1322.

Wagner, G.; and Choset, H. 2011. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In *IROS*, 3260–3267.

Wagner, G.; and Choset, H. 2017. Path Planning for Multiple Agents under Uncertainty. In *ICAPS*, 577–585.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2020. Generalized and Sub-Optimal Bipartite Constraints for Conflict-Based Search. In *AAAI*, 7277–7284.

Walker, T. T.; Sturtevant, N. R.; Zhang, H.; Li, J.; Felner, A.; and Kumar, T. K. S. 2021. Conflict-Based Increasing Cost Search. In *ICAPS*, 385–395.

Wang, H.; and Rubenstein, M. 2020. Walk, Stop, Count, and Swap: Decentralized Multi-Agent Path Finding With Theoretical Guarantees. *IEEE Robotics and Automation Letters*, 5(2): 1119–1126.

Wang, K. C.; and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *AAAI*, 1752–1760.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*, 1444–1449.