Lifelong Multi-Agent Path Finding in Large-Scale Warehouses*

Jiaoyang Li,[1] Andrew Tinka,[2] Scott Kiesel,[2]
Joseph W. Durham,[2] T. K. Satish Kumar,[1] Sven Koenig[1]
[1] University of Southern California
[2] Amazon Robotics
jiaoyanl@usc.edu, {atinka, skkiesel, josepdur}@amazon.com, tkskwork@gmail.com, skoenig@usc.edu

Abstract

Multi-Agent Path Finding (MAPF) is the problem of moving a team of agents to their goal locations without collisions. In this paper, we study the lifelong variant of MAPF, where agents are constantly engaged with new goal locations, such as in large-scale automated warehouses. We propose a new framework Rolling-Horizon Collision Resolution (RHCR) for solving lifelong MAPF by decomposing the problem into a sequence of Windowed MAPF instances, where a Windowed MAPF solver resolves collisions among the paths of the agents only within a bounded time horizon and ignores collisions beyond it. RHCR is particularly well suited to generating pliable plans that adapt to continually arriving new goal locations. We empirically evaluate RHCR with a variety of MAPF solvers and show that it can produce high-quality solutions for up to 1,000 agents (= 38.9% of the empty cells on the map) for simulated warehouse instances, significantly outperforming existing work.

1 Introduction

Multi-Agent Path Finding (MAPF) is the problem of moving a team of agents from their start locations to their goal locations while avoiding collisions. The quality of a solution is measured by *flowtime* (the sum of the arrival times of all agents at their goal locations) or *makespan* (the maximum of the arrival times of all agents at their goal locations). MAPF is NP-hard to solve optimally (Yu and LaValle 2013).

MAPF has numerous real-world applications, such as autonomous aircraft-towing vehicles (Morris et al. 2016), video game characters (Li et al. 2020b), and quadrotor swarms (Hönig et al. 2018). Today, in automated warehouses, mobile robots called *drive units* already autonomously move inventory pods or flat packages from one location to another (Wurman, D'Andrea, and Mountz 2007; Kou et al. 2020). However, MAPF is only the "one-shot" variant of the actual problem in many applications. Typically, after an agent reaches its goal location, it does not stop and wait there forever. Instead, it is assigned a new goal location and required to keep moving, which is referred to as *lifelong MAPF* (Ma et al. 2017) and characterized by agents constantly being assigned new goal locations.

*This paper is an extension of (Li et al. 2020c).

Existing methods for solving lifelong MAPF include (1) solving it as a whole (Nguyen et al. 2017), (2) decomposing it into a sequence of MAPF instances where one replans paths at every timestep for all agents (Wan et al. 2018; Grenouilleau, van Hoeve, and Hooker 2019), and (3) decomposing it into a sequence of MAPF instances where one plans new paths at every timestep for only the agents with new goal locations (Cáp, Vokrínek, and Kleiner 2015; Ma et al. 2017; Liu et al. 2019).

In this paper, we propose a new framework *Rolling-Horizon Collision Resolution* (RHCR) for solving lifelong MAPF where we decompose lifelong MAPF into a sequence of Windowed MAPF instances and replan paths once every h timesteps (replanning period h is user-specified) for interleaving planning and execution. A *Windowed MAPF* instance is different from a regular MAPF instance in the following ways:

1. it allows an agent to be assigned a sequence of goal locations within the same Windowed MAPF episode, and
2. collisions need to be resolved only for the first w timesteps (time horizon $w \geq h$ is user-specified).

The benefit of this decomposition is two-fold. First, it keeps the agents continually engaged, avoiding idle time, and thus increasing throughput. Second, it generates pliable plans that adapt to continually arriving new goal locations. In fact, resolving collisions in the entire time horizon (i.e., $w = \infty$) is often unnecessary since the paths of the agents can change as new goal locations arrive.

We evaluate RHCR with various MAPF solvers, namely CA* (Silver 2005) (incomplete and suboptimal), PBS (Ma et al. 2019) (incomplete and suboptimal), ECBS (Barer et al. 2014) (complete and bounded suboptimal), and CBS (Sharon et al. 2015) (complete and optimal). We show that, for each MAPF solver, using a bounded time horizon yields similar throughput as using the entire time horizon but with a significantly smaller runtime. We also show that RHCR outperforms existing work and can scale up to 1,000 agents (= 38.9% of the empty cells on the map) for simulated warehouse instances.

2 Background

In this section, we first introduce several state-of-the-art MAPF solvers and then discuss existing research on life-

long MAPF. We finally review the elements of the bounded horizon idea that have guided previous research.

2.1 Popular MAPF Solvers

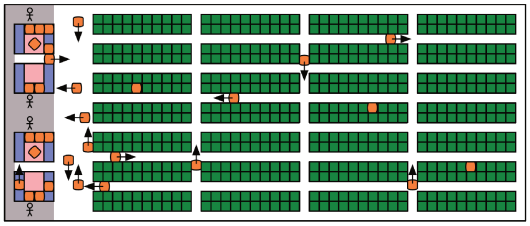
Many MAPF solvers have been proposed in recent years, including rule-based solvers (Luna and Bekris 2011; de Wilde, ter Mors, and Witteveen 2013), prioritized planning (Okumura et al. 2019), compilation-based solvers (Lam et al. 2019; Surynek 2019), A*-based solvers (Goldenberg et al. 2014; Wagner 2015), and dedicated search-based solvers (Sharon et al. 2013; Barer et al. 2014). We present four representative MAPF solvers.

CBS Conflict-Based Search (CBS) (Sharon et al. 2015) is a popular two-level MAPF solver that is complete and optimal. At the high level, CBS starts with a root node that contains a shortest path for each agent (ignoring other agents). It then chooses and resolves a collision by generating two child nodes, each with an additional constraint that prohibits one of the agents involved in the collision from being at the colliding location at the colliding timestep. It then calls its low level to replan the paths of the agents with the new constraints. CBS repeats this procedure until it finds a node with collision-free paths. CBS and its enhanced variants (Gange, Harabor, and Stuckey 2019; Li et al. 2019, 2020a) are among the state-of-the-art optimal MAPF solvers.

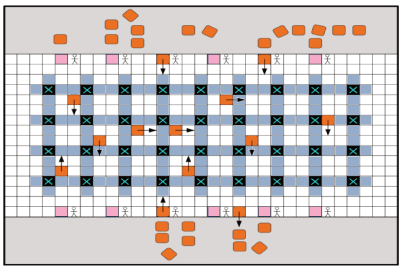
ECBS Enhanced CBS (ECBS) (Barer et al. 2014) is a complete and bounded-suboptimal variant of CBS. The bounded suboptimality (i.e., the solution cost is a user-specified factor away from the optimal cost) is achieved by using focal search (Pearl and Kim 1982), instead of best-first search, in both the high- and low-level searches of CBS. ECBS is the state-of-the-art bounded-suboptimal MAPF solver.

CA* Cooperative A* (CA*) (Silver 2005) is based on a simple prioritized-planning scheme: Each agent is given a unique priority and computes, in priority order, a shortest path that does not collide with the (already planned) paths of agents with higher priorities. CA*, or prioritized planning in general, is widely used in practice due to its small runtime. However, it is suboptimal and incomplete since its predefined priority ordering can sometimes result in solutions of bad quality or even fail to find any solutions for solvable MAPF instances.

PBS Priority-Based Search (PBS) (Ma et al. 2019) combines the ideas of CBS and CA*. The high level of PBS is similar to CBS except that, when resolving a collision, instead of adding additional constraints to the resulting child nodes, PBS assigns one of the agents involved in the collision a higher priority than the other agent in the child nodes. The low level of PBS is similar to CA* in that it plans a shortest path that is consistent with the partial priority ordering generated by the high level. PBS outperforms many variants of prioritized planning in terms of solution quality but is still incomplete and suboptimal.



(a) Fulfillment warehouse map, borrowed from (Wurman, D'Andrea, and Mountz 2007).



(b) Sorting center map, modified from (Wan et al. 2018).

Figure 1: A well-formed fulfillment warehouse map and a non-well-formed sorting center map. Orange squares represent robots. In (a), the endpoints consist of the green cells (representing locations that store inventory pods) and blue cells (representing the work stations). In (b), the endpoints consist of the blue cells (representing locations where the drive units drop off packages) and pink cells (representing the loading stations). Black cells labeled "X" represent chutes (obstacles).

2.2 Prior Work on Lifelong MAPF

We classify prior work on lifelong MAPF into three categories.

Method (1) The first method is to solve lifelong MAPF as a whole in an offline setting (i.e., knowing all goal locations a priori) by reducing lifelong MAPF to other well-studied problems. For example, Nguyen et al. (2017) formulate lifelong MAPF as an answer set programming problem. However, the method only scales up to 20 agents in their paper, each with only about 4 goal locations. This is not surprising because MAPF is a challenging problem and its lifelong variant is even harder.

Method (2) A second method is to decompose lifelong MAPF into a sequence of MAPF instances where one replans paths at every timestep for all agents. To improve the scalability, researchers have developed incremental search techniques that reuse previous search effort. For example, Wan et al. (2018) propose an incremental variant of CBS that reuses the tree of the previous high-level search. However, it has substantial overhead in constructing a new high-level tree from the previous one and thus does not improve the scalability by much. Svancara et al. (2019) use the framework of Independence Detection (Standley 2010) to reuse the paths from the previous iteration. It replans paths for

only the new agents (in our case, agents with new goal locations) and the agents whose paths are affected by the paths of the new agents. However, when the environment is dense (i.e., contains many agents and many obstacles, which is common for warehouse scenarios), almost all paths are affected, and thus it still needs to replan paths for most agents.

Method (3) A third method is similar to the second method but restricts replanning to the paths of the agents that have just reached their goal locations. The new paths need to avoid collisions not only with each other but also with the paths of the other agents. Hence, this method could degenerate to prioritized planning in case where only one agent reaches its goal location at every timestep. As a result, the general drawbacks of prioritized planning, namely its incompleteness and its potential to generate costly solutions, resurface in this method. To address the incompleteness issue, Cáp, Vokrínek, and Kleiner (2015) introduce the idea of well-formed infrastructures to enable backtrack-free search. In well-formed infrastructures, all possible goal locations are regarded as endpoints, and, for every pair of endpoints, there exists a path that connects them without traversing any other endpoints. In real-world applications, some maps, such as the one in Figure 1a, may satisfy this requirement, but other maps, such as the one in Figure 1b, may not. Moreover, additional mechanisms are required during path planning. For example, one needs to force the agents to "hold" their goal locations (Ma et al. 2017) or plan "dummy paths" for the agents (Liu et al. 2019) after they reach their goal locations. Both alternatives result in unnecessarily long paths for agents, decreasing the overall throughput, as shown in our experiments.

Summary Method (1) needs to know all goal locations a priori and has limited scalability. Method (2) can work in an online setting and scales better than Method (1). However, replanning for all agents at every timestep is time-consuming even if one uses incremental search techniques. As a result, its scalability is also limited. Method (3) scales to substantially more agents than the first two methods, but the map needs to have an additional structure to guarantee completeness. As a result, it works only for specific classes of lifelong MAPF instances. In addition, Methods (2) and (3) plan at every timestep, which may not be practical since planning is time-consuming.

2.3 Bounded-Horizon Planing

Bounded-horizon planning is not a new idea. Silver (2005) has already applied this idea to regular MAPF with CA*. He refers to it as Windowed Hierarchical Cooperative A* (WHCA*) and empirically shows that WHCA* runs faster as the length of the bounded horizon decreases but also generates longer paths. In this paper, we showcase the benefits of applying this idea to lifelong MAPF and other MAPF solvers. In particular, RHCR yields the benefits of lower computational costs for planning with bounded horizons while keeping the agents busy and yet, unlike WHCA* for regular MAPF, decreasing the solution quality only slightly.

3 Problem Definition

The input is a graph $G = (V, E)$, whose vertices V correspond to locations and whose edges E correspond to connections between two neighboring locations, and a set of m agents $\{a_1, \ldots, a_m\}$, each with an initial location. We study an online setting where we do not know all goal locations a priori. We assume that there is a *task assigner* (outside of our path-planning system) that the agents can request goal locations from during the operation of the system.[1] Time is discretized into timesteps. At each timestep, every agent can either *move* to a neighboring location or *wait* at its current location. Both move and wait actions have unit duration. A *collision* occurs iff two agents occupy the same location at the same timestep (called a vertex conflict in (Stern et al. 2019)) or traverse the same edge in opposite directions at the same timestep (called a swapping conflict in (Stern et al. 2019)). Our task is to plan collision-free paths that move all agents to their goal locations and maximize the *throughput*, i.e., the average number of goal locations visited per timestep. We refer to the set of collision-free paths for all agents as a *MAPF plan*.

We study the case where the task assigner is not within our control so that our path-planning system is applicable in different domains. But, for a particular domain, one can design a hierarchical framework that combines a domain-specific task assigner with our domain-independent path-planning system. Compared to coupled methods that solve task assignment and path finding jointly, a hierarchical framework is usually a good way to achieve efficiency. For example, the task assigners in (Ma et al. 2017; Liu et al. 2019) for fulfillment warehouse applications and in (Grenouilleau, van Hoeve, and Hooker 2019) for sorting center applications can be directly combined with our path-planning system. We also showcase two simple task assigners, one for each application, in our experiments.

We assume that the drive units can execute any MAPF plan perfectly. Although this seems to be not realistic, there exist some post-processing methods (Hönig et al. 2016) that can take the kinematic constraints of drive units into consideration and convert MAPF plans to executable commands for them that result in robust execution. For example, Hönig et al. (2019) propose a framework that interleaves planning and execution and can be directly incorporated with our framework RHCR.

4 Rolling-Horizon Collision Resolution

Rolling-Horizon Collision Resolution (RHCR) has two user-specified parameters, namely the time horizon w and the replanning period h. The time horizon w specifies that the Windowed MAPF solver has to resolve collisions within a time horizon of w timesteps. The replanning period h specifies that the Windowed MAPF solver needs to replan paths once every h timesteps. The Windowed MAPF solver has to

[1] In case there are only a finite number of tasks, after all tasks have been assigned, we assume that the task assigner will assign a dummy task to an agent whose goal location is, e.g., a charging station, an exit, or the current location of the agent.

replan paths more frequently than once every w timesteps to avoid collisions, i.e., w should be larger than or equal to h.

In every Windowed MAPF episode, say, starting at timestep t, RHCR first updates the start location s_i and the goal location sequence $\mathbf{g_i}$ for each agent a_i. RHCR sets the start location s_i of agent a_i to its location at timestep t. Then, RHCR calculates a lower bound on the number of timesteps d that agent a_i needs to visit all remaining locations in $\mathbf{g_i}$, i.e.,

$$d = \text{dist}(s_i, \mathbf{g_i}[0]) + \sum_{j=1}^{|\mathbf{g_i}|-1} \text{dist}(\mathbf{g_i}[j-1], \mathbf{g_i}[j]), \quad (1)$$

where dist(x, y) is the distance from location x to location y and $|\mathbf{x}|$ is the cardinality in sequence $\mathbf{x}$.[2] d being smaller than h indicates that agent a_i might finish visiting all its goal locations and then being idle before the next Windowed MAPF episode starts at timestep $t + h$. To avoid this situation, RHCR continually assigns new goal locations to agent a_i until $d \geq h$. Once the start locations and the goal location sequences for all agents require no more updates, RHCR calls a Windowed MAPF solver to find paths for all agents that move them from their start locations to all their goal locations in the order given by their goal location sequences and are collision-free for the first w timesteps. Finally, it moves the agents for h timesteps along the generated paths and remove the visited goal locations from their goal location sequences.

RHCR uses flowtime as the objective of the Windowed MAPF solver, which is known to be a reasonable objective for lifelong MAPF (Svancara et al. 2019). Compared to regular MAPF solvers, Windowed MAPF solvers need to be changed in two aspects:

1. each path needs to visit a sequence of goal locations, and
2. the paths need to be collision-free for only the first w timesteps.

We describe these changes in detail in the following two subsections.

4.1 A* for a Goal Location Sequence

The low-level searches of all MAPF solvers discussed in Section 2.1 need to find a path for an agent from its start location to its goal location while satisfying given spatio-temporal constraints that prohibit the agent from being at certain locations at certain timesteps. Therefore, they often use location-time A* (Silver 2005) (i.e., A* that searches in the location-time space where each state is a pair of location and timestep) or any of its variants. However, a characteristic feature of a Windowed MAPF solver is that it plans a path for each agent that visits a sequence of goal locations. Despite this difference, techniques used in the low-level search of regular MAPF solvers can be adapted to the low-level search of Windowed MAPF solvers. In fact, Grenouilleau, van Hoeve, and Hooker (2019) perform a truncated version of this adaptation for the pickup and delivery problem. They propose Multi-Label A* that can find a path for a single agent that visits two ordered goal locations, namely its assigned pickup location and its goal location. In Algorithm 1, we generalize Multi-Label A* to a sequence of goal locations.[3]

Algorithm 1: The low-level search for Windowed MAPF solvers generalizing Multi-Label A* (Grenouilleau, van Hoeve, and Hooker 2019).

Input: Start location s_i, goal location sequence $\mathbf{g_i}$.

```
1  R.location ← s_i, R.time ← 0, R.g ← 0;
2  R.label ← 0;
3  R.h ← ComputeHValue(R.location, R.label);
4  open.push(R);
5  while open is not empty do
6      P ← open.pop();          // Pop the node with the minimum f.
7      if P.location = g_i[P.label] then // Update label.
8          P.label ← P.label + 1;
9      if P.label = |g_i| then          // Goal test.
10         return the path retrieved from P;
11     foreach child node Q of P do // Generate child nodes.
12         open.push(Q);
13 return "No Solution";
14 Function ComputeHValue(Location x, Label l) :
15     return dist(x, g_i[l]) + Σ_{j=l+1}^{|g_i|-1} dist(g_i[j-1], g_i[j]);
```

Algorithm 1 uses the structure of location-time A*. For each node N, we add an additional attribute $N.label$ that indicates the number of goal locations in the goal location sequence $\mathbf{g_i}$ that the path from the root node to node N has already visited. For example, $N.label = 2$ indicates that the path has already visited goal locations $\mathbf{g_i}[0]$ and $\mathbf{g_i}[1]$ but not goal location $\mathbf{g_i}[2]$. Algorithm 1 computes the h-value of a node as the distance from the location of the node to the next goal location plus the sum of the distances between consecutive future goal locations in the goal location sequence [Lines 14-15]. In the main procedure, Algorithm 1 first creates the root node R with label 0 and pushes it into the pri-

[2] Computing d relies on the distance function dist(x, y). Here, and in any other place where dist(x, y) is required, prepossessing techniques can be used to increase efficiency. In particular, large warehouses have a candidate set of goal locations as the only possible values for y, enabling the pre-computation and caching of shortest-path trees.

[3] Planning a path for an agent to visit a sequence of goal locations is not straightforward. While one can call a sequence of location-time A* to plan a shortest path between every two consecutive goal locations and concatenate the resulting paths, the overall path is not necessarily the shortest because the presence of spatio-temporal constraints introduces spatio-temporal dependencies among the path segments between different goal locations, e.g., arriving at the first goal location at the earliest timestep may result in a longer overall path than arriving there later. We therefore need Algorithm 1.

oritized queue $open$ [Lines 1-4]. While $open$ is not empty [Line 5], the node P with the smallest f-value is selected for expansion [Line 6]. If P has reached its current goal location [Line 7], $P.label$ is incremented [Line 8]. If $P.label$ equals the cardinality of the goal location sequence [Line 9], Algorithm 1 terminates and returns the path [Line 10]. Otherwise, it generates child nodes that respect the given spatio-temporal constraints [Lines 11-12]. The labels of the child nodes equal $P.label$. Checking the priority queue for duplicates requires a comparison of labels in addition to other attributes.

4.2 Bounded-Horizon MAPF Solvers

Another characteristic feature of Windowed MAPF solvers is the use of a bounded horizon. Regular MAPF solvers can be easily adapted to resolving collisions for only the first w timesteps. Beyond the first w timesteps, the solvers ignore collisions among agents and assume that each agent follows its shortest path to visit all its goal locations, which ensures that the agents head in the correct directions in most cases. We now provide details on how to modify the various MAPF solvers discussed in Section 2.1.

Bounded-Horizon (E)CBS Both CBS and ECBS search by detecting and resolving collisions. In their bounded-horizon variants, we only need to modify the collision detection function. While (E)CBS finds collisions among all paths and can then resolve any one of them, bounded-horizon (E)CBS only finds collisions among all paths that occur in the first w timesteps and can then resolve any one of them. The remaining parts of (E)CBS stay the same. Since bounded-horizon (E)CBS needs to resolve fewer collisions, it generates a smaller high-level tree and thus runs faster than standard (E)CBS.

Bounded-Horizon CA* CA* searches based on priorities, where an agent avoids collisions with all higher-priority agents. In its bounded-horizon variant, an agent is required to avoid collisions with all higher-priority agents but only during the first w timesteps. Therefore, when running location-time A* for each agent, we only consider the spatio-temporal constraints during the first w timesteps induced by the paths of higher-priority agents. The remaining parts of CA* stay the same. Since bounded-horizon CA* has fewer spatio-temporal constraints, it runs faster and is less likely to fail to find solutions than CA*. Bounded-horizon CA* is identical to WHCA* in (Silver 2005).

Bounded-Horizon PBS The high-level search of PBS is similar to that of CBS and is based on resolving collisions, while the low-level search of PBS is similar to that of CA* and plans paths that are consistent with the partial priority ordering generated by the high-level search. Hence, we need to modify the collision detection function of the high level of PBS (just like how we modify CBS) and incorporate the limited consideration of spatio-temporal constraints into its low level (just like how we modify CA*). As a result, bounded-horizon PBS generates smaller high-level trees and runs faster in its low level than standard PBS.

4.3 Behavior of RHCR

We first show that resolving collisions for a longer time horizon in lifelong MAPF does not necessarily result in better solutions. Below is such an example.

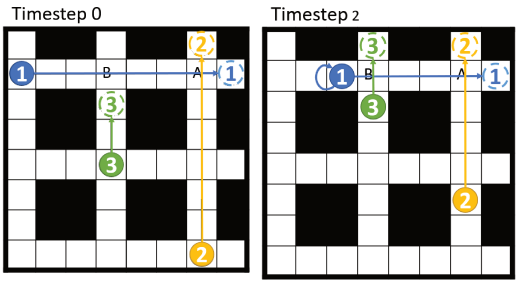
Example 1. *Consider the lifelong MAPF instance shown in Figure 2a with time horizon $w = 4$ and replanning period $h = 2$, and assume that we use an optimal Windowed MAPF solver. At timestep 0 (left figure), all agents follow their shortest paths as no collisions will occur during the first 4 timesteps. Then, agent a_3 reaches its goal location at timestep 2 and is assigned a new goal location (right figure). If agents a_1 and a_3 both follow their shortest paths, the Windowed MAPF solver finds a collision between them at cell B at timestep 3 and forces agent a_1 to wait for one timestep. The resulting number of wait actions is 1. However, if we solve this example with time horizon $w = 8$, as shown in Figure 2b, we could generate paths with more wait actions. At timestep 0 (left figure), the Windowed MAPF solver finds a collision between agents a_1 and a_2 at cell A at timestep 6 and thus forces agent a_2 to wait for one timestep. Then, at timestep 2 (right figure), the Windowed MAPF solver finds a collision between agents a_1 and a_3 at cell B at timestep 3 and forces agent a_3 to wait for one timestep. The resulting number of wait actions is 2.*

Similar cases are also found in our experiments: sometimes RHCR with smaller time horizons achieves higher throughput than with larger time horizons. All of these cases support our claim that, in lifelong MAPF, resolving all collisions in the entire time horizon is unnecessary, which is different from regular MAPF. Nevertheless, the bounded-horizon method also has a drawback since using too small a value for the time horizon may generate *deadlocks* that prevent agents from reaching their goal locations, as shown in Example 2.

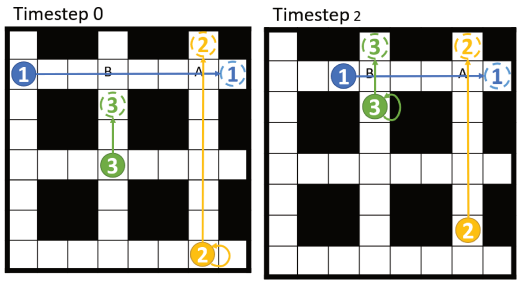
Example 2. *Consider the lifelong MAPF instance shown in Figure 2c with time horizon $w = 2$ and replanning period $h = 2$, and assume that we use an optimal Windowed MAPF solver. At timestep 0, the Windowed MAPF solver returns path [B, B, B, C, D, E] (of length 5) for agent a_1 and path [C, C, C, B, A, L] (of length 5) for agent a_2, which are collision-free for the first 2 timesteps. It does not return the collision-free paths where one of the agents uses the upper corridor, nor the collision-free paths where one of the agents leaves the lower corridor first (to let the other agent reach its goal location) and then re-enters it, because the resulting flowtime is larger than $5 + 5 = 10$. Therefore, at timestep 2, both agents are still waiting at cells B and C. The Windowed MAPF solver then finds the same paths for both agents again and forces them to wait for two more timesteps. Overall, the agents wait at cells B and C forever and never reach their goal locations.*

4.4 Avoiding Deadlocks

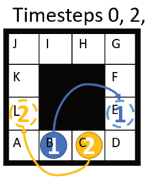
To address the deadlock issue shown in Example 2, we can design a potential function to evaluate the progress of the



(a) A lifelong MAPF instance with time horizon $w = 4$. Agent a_3 reaches its goal location at timestep 2 and is then assigned a new goal location.



(b) The same lifelong MAPF instance as shown in (a) with time horizon $w = 8$.



(c) A lifelong MAPF instance with time horizon $w = 2$.

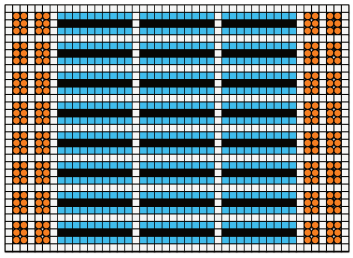
Figure 2: Lifelong MAPF instances with replanning period $h = 2$. Solid (dashed) circles represent the current (goal) locations of the agents.

agents and increase the time horizon if the agents do not make sufficient progress. For example, after the Windowed MAPF solver returns a set of paths, we compute the potential function $P(w) = |\{a_i | \text{COMPUTEHVALUE}(x_i, l_i) < \text{COMPUTEHVALUE}(s_i, 0), 1 \leq i \leq m\}|$, where function COMPUTEHVALUE$(\cdot, \cdot)$ is defined on Lines 14-15 in Algorithm 1, x_i is the location of agent a_i at timestep w, l_i is the number of goal locations that it has visited during the first w timesteps, and s_i is its location at timestep 0. $P(w)$ estimates the number of agents that need fewer timesteps to visit all their goal locations from timestep w on than from timestep 0 on. We increase w and continue running the Windowed MAPF solver until $P(w) \geq p$, where $p \in [0, m]$ is a user-specified parameter. This ensures that at least p agents have visited (some of) their goal locations or got closer to their next goal locations during the first w timesteps.

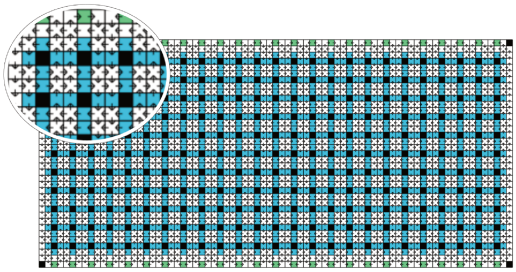
Example 3. *Consider again the lifelong MAPF instance in Figure 2c. Assume that $p = 1$. When the time horizon $w = 2$, as discussed in Example 2, both agents keep staying at their start locations, and thus $P(2) = 0$. When we increase w to 3, the Windowed MAPF solver finds the paths [B, C, D, E] (of length 3) for agent a_1 and [C, D, E, ..., K, L] (of length 9) for agent a_2. Now, $P(3) = 1$ because agent a_1 is at cell E at timestep 3 and needs 0 more timesteps to visit its goal locations. Since $P(3) = p$, the Windowed MAPF solver returns this set of paths and avoids the deadlock.*

There are several methods for designing such potential functions, e.g., the number of goal locations that have been reached before timestep w or the sum of timesteps that all agents need to visit their goal locations from timestep w on minus that the sum of timesteps that all agents need to visit their goal locations from timestep 0 on. In our experiments, we use only the one described above. We intend to design more potential functions and compare their effectiveness in the future.

Unfortunately, RHCR with the deadlock avoidance mechanism is still incomplete. Imagine an intersection where many agents are moving horizontally but only one agent wants to move vertically. If we always let the horizontal agents move and the vertical agent wait, we maximize the throughput but lose completeness (as the vertical agent can never reach its goal location). But if we let the vertical agent move and the horizontal agents wait, we might guarantee completeness but will achieve a lower throughput. This issue can occur even if we use time horizon $w = \infty$. Since throughput and completeness can compete with each other, we choose to focus on throughput instead of completeness in this paper.



(a) Fulfillment warehouse map.



(b) Sorting center map.

Figure 3: Two typical warehouse maps. Black cells represent obstacles, which the agents cannot occupy. Cells of other colors represent empty locations, which the agents can occupy and traverse.

5 Empirical Results

We implement RHCR in C++ with four Windowed MAPF solvers based on CBS, ECBS, CA* and PBS.[4] We use

[4]The code is available at https://github.com/Jiaoyang-Li/RHCR.

Framework	$m = 60$	$m = 100$	$m = 140$
RHCR	2.33	3.56	4.55
HE	2.17 (-6.80%)	3.33 (-6.33%)	4.35 (-4.25%)
RDP	2.19 (-6.00%)	3.41 (-4.16%)	4.50 (-1.06%)
RHCR	0.33 ± 0.01	2.04 ± 0.04	7.78 ± 0.14
HE	0.01 ± 0.00	0.02 ± 0.00	0.04 ± 0.01
RDP	0.02 ± 0.00	0.05 ± 0.01	0.17 ± 0.05

Table 1: Average throughput (Rows 2-4) and average runtime (in seconds) per run (Rows 5-7) of RHCR, holding endpoints (denoted by HE) and reserving dummy paths (denoted by RDP). Numbers in parenthesis characterize throughput differences (in percentage) compared to RHCR. Numbers after "$\pm$" indicate standard deviations.

SIPP (Phillips and Likhachev 2011), an advanced variant of location-time A*, as the low-level solver for CA* and PBS. We use Soft Conflict SIPP (SCIPP) (Cohen et al. 2019), a recent variant of SIPP that generally breaks ties in favor of paths with lower numbers of collisions, for CBS and ECBS. We use CA* with random restarts where we repeatedly restart CA* with a new random priority ordering until it finds a solution. We also implement two existing realizations of Method (3) for comparison, namely holding endpoints (Ma et al. 2017) and reserving dummy paths (Liu et al. 2019). We do not compare against Method (1) since it does not work in our online setting. We do not compare against Method (2) since we choose dense environments to stress test various methods and its performance in dense environments is similar to that of RHCR with an infinite time horizon. We simulate 5,000 timesteps for each experiment with potential function threshold $p = 1$. We conduct all experiments on Amazon EC2 instances of type "m4.xlarge" with 16 GB memory.

5.1 Fulfillment Warehouse Application

In this subsection, we introduce fulfillment warehouse problems, that are commonplace in automated warehouses and are characterized by blocks of inventory pods in the center of the map and work stations on its perimeter. Method (3) is applicable in such well-formed infrastructures, and we thus compare RHCR with both realizations of Method (3). We use the map in Figure 3a from (Liu et al. 2019). It is a 33×46 4-neighbor grid with 16% obstacles. The initial locations of agents are uniformly chosen at random from the orange cells, and the task assigner chooses the goal locations for agents uniformly at random from the blue cells. For RHCR, we use time horizon $w = 20$ and replanning period $h = 5$. For the other two methods, we replan at every timestep, as required by Method (3). All methods use PBS as their (Windowed) MAPF solvers.

Table 1 reports the throughput and runtime of these methods with different numbers of agents m. In terms of throughput, RHCR outperforms the reserving dummy path method, which in turn outperforms the holding endpoints method. This is because, as discussed in Section 2.2, Method (3) usually generates unnecessary longer paths in its solutions. In terms of runtime, however, our method is slower per run (i.e., per call to the (Windowed) MAPF solver) because the competing methods usually replan for fewer than 5 agents. The disadvantages of these methods are that they need to replan at every timestep, achieve a lower throughput, and are not applicable to all maps.

5.2 Sorting Center Application

In this subsection, we introduce sorting center problems, that are also commonplace in warehouses and are characterized by uniformly placed chutes in the center of the map and work stations on its perimeter. Method (3) is not applicable since they are typically not well-formed infrastructures. We use the map in Figure 3b. It is a 37×77 4-neighbor grid with 10% obstacles. The 50 green cells on the top and bottom boundaries represent work stations where humans put packages on the drive units. The 275 black cells (except for the four corner cells) represent the chutes where drive units occupy one of the adjacent blue cells and drop their packages down the chutes. The drive units are assigned to green cells and blue cells alternately. In our simulation, the task assigner chooses blue cells uniformly at random and chooses green cells that are closest to the current locations of the drive units. The initial locations of the drive units are uniformly chosen at random from the empty cells (i.e., cells that are not black). We use a directed version of this map to make MAPF solvers more efficient since they do not have to resolve swapping conflicts, which allows us to focus on the efficiency of the overall framework. Our handcrafted horizontal directions include two rows with movement from left to right alternating with two rows with movement from right to left, and our handcrafted vertical directions include two columns with movement from top to bottom alternating with two columns with movement from bottom to top. We use replanning period $h = 5$.

Tables 2 and 3 report the throughput and runtime of RHCR using PBS, ECBS with suboptimality factor 1.1, CA*, and CBS for different values of time horizon w. As expected, w does not substantially affect the throughput. In most cases, small values of w change the throughput by less than 1% compared to $w = \infty$. However, w substantially affects the runtime. In all cases, small values of w speed up RHCR by up to a factor of 6 without compromising the throughput. Small values of w also yield scalability with respect to the number of agents, as indicated in both tables by missing "-". For example, PBS with $w = \infty$ can only solve instances up to 700 agents, while PBS with $w = 5$ can solve instances up to at least 1,000 agents.

5.3 Dynamic Bounded Horizons

We evaluate whether we can use the deadlock avoidance mechanism to decide the value of w for each Windowed MAPF episode automatically by using a larger value of p and starting with a smaller value of w. We use RHCR with $w = 5$ and $p = 60$ on the instances in Section 5.1 with 60 agents. We use ECBS with suboptimality factor 1.5 as the Windowed MAPF solver. The average time horizon that is actually used in each Windowed MAPF episode is 9.97 timesteps. The throughput and runtime are 2.10 and 0.35s,

	w	$m = 400$	$m = 500$	$m = 600$	$m = 700$	$m = 800$	$m = 900$	$m = 1000$
Throughput	5	12.27 (-1.56%)	15.17 (-1.84%)	17.97 (-2.35%)	20.69 (-2.85%)	23.36	25.79	27.95
	10	12.41 (-0.41%)	15.43 (-0.19%)	18.38 (-0.11%)	21.19 (-0.52%)	23.94	26.44	28.77
	20	12.45 (-0.07%)	15.48 (+0.12%)	18.38 (-0.11%)	21.24 (-0.26%)	23.91	-	-
	∞	12.46	15.46	18.40	21.30	-	-	-
Runtime	5	0.61 ± 0.00	1.12 ± 0.01	1.87 ± 0.01	3.01 ± 0.01	4.73 ± 0.02	7.30 ± 0.04	10.97 ± 0.06
	10	0.89 ± 0.00	1.66 ± 0.01	2.91 ± 0.01	4.81 ± 0.02	7.79 ± 0.04	12.66 ± 0.07	21.31 ± 0.14
	20	1.36 ± 0.01	2.71 ± 0.01	5.11 ± 0.03	9.28 ± 0.06	17.46 ± 0.14	-	-
	∞	1.83 ± 0.01	3.84 ± 0.03	7.63 ± 0.06	16.16 ± 0.17	-	-	-

Table 2: Average throughput and average runtime (in seconds) per run of RHCR using PBS. "-" indicates that it takes more than 1 minute for the Windowed MAPF solver to find a solution in any run. Numbers in parenthesis characterize throughput differences (in percentage) compared to time horizon $w = \infty$. Numbers after "$\pm$" indicate standard deviations.

	w	$m = 100$	$m = 200$	$m = 300$	$m = 400$	$m = 500$	$m = 600$
Throughput	5	3.19 (+1.02%)	6.23 (-1.21%)	9.17 (-1.47%)	12.03 (-2.03%)	14.79 (-2.68%)	17.28
	∞	3.16	6.31	9.31	12.28	15.20	-
Runtime	5	0.07 ± 0.00	0.26 ± 0.00	0.64 ± 0.00	1.27 ± 0.01	2.37 ± 0.02	4.22 ± 0.10
	∞	0.38 ± 0.00	1.81 ± 0.01	5.09 ± 0.03	11.48 ± 0.09	23.47 ± 0.22	-

(a) RHCR using ECBS.

	w	$m = 100$	$m = 200$	$m = 300$	$m = 400$
Throughput	5	3.19 (+0.53%)	6.17 (-0.48%)	9.12 (-0.35%)	-
	∞	3.17	6.20	9.16	-
Runtime	5	0.05 ± 0.00	0.21 ± 0.01	1.07 ± 0.10	-
	∞	0.19 ± 0.00	0.84 ± 0.02	2.58 ± 0.12	-

(b) RHCR using CA*.

	w	$m = 100$	$m = 200$
Throughput	5	3.17	-
	∞	-	-
Runtime	5	0.14 ± 0.03	-
	∞	-	-

(c) RHCR using CBS.

Table 3: Results of RHCR using ECBS, CA*, and CBS. Numbers are reported in the same format as in Table 2.

respectively. However, if we use a fixed w (i.e., $p = 0$), we achieve a throughput of 1.72 and a runtime of 0.07s for time horizon $w = 5$ and a throughput of 2.02 and a runtime of 0.17s for time horizon $w = 10$. Therefore, this dynamic bounded-horizon method is able to find a good horizon length that produces high throughput but induces runtime overhead as it needs to increase the time horizon repeatedly.

6 Conclusions

In this paper, we proposed Rolling-Horizon Collision Resolution (RHCR) for solving lifelong MAPF by decomposing it into a sequence of Windowed MAPF instances. We showed how to transform several regular MAPF solvers to Windowed MAPF solvers. Although RHCR does not guarantee completeness or optimality, we empirically demonstrated its success on fulfillment warehouse maps and sorting center maps. We demonstrated its scalability up to 1,000 agents while also producing solutions of high throughput. Compared to Method (3), RHCR not only applies to general graphs but also yields better throughput. Overall, RHCR applies to general graphs, invokes replanning at a user-specified frequency, and is able to generate pliable plans that cannot only adapt to continually arriving new goal locations but also avoids wasting computational effort in anticipating a distant future.

RHCR is simple, flexible, and powerful. It introduces a new direction for solving lifelong MAPF problems. There are many avenues of future work:

1. adjusting the time horizon w automatically based on the congestion and the planning time budget,
2. grouping the agents and planning in parallel, and
3. deploying incremental search techniques to reuse search effort from previous searches.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712 as well as a gift from Amazon. Part of the research was completed during Jiaoyang Li's internship at Amazon Robotics. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

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 *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 19–27.

Cáp, M.; Vokrínek, J.; and Kleiner, A. 2015. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-Formed Infrastructures. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 324–332.

Cohen, L.; Uras, T.; Kumar, T. K. S.; and Koenig, S. 2019. Optimal and Bounded-Suboptimal Multi-Agent Motion Planning. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 44–51.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2013. Push and Rotate: Cooperative Multi-Agent Path Planning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 87–94.

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 155–162.

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 *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 181–185.

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 *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 477–485.

Hönig, W.; Preiss, J. A.; Kumar, T. K. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory Planning for Quadrotor Swarms. *IEEE Transactions on Robotics* 34(4): 856–869.

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 *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 9925–9932.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1289–1296.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 442–449.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020a. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201.

Li, J.; Sun, K.; Ma, H.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2020b. Moving Agents in Formation in Congested Environments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 726–734.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020c. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1898–1900.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1152–1160.

Luna, R.; and Bekris, K. E. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 294–300.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7643–7650.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 837–845.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI Workshop on Planning for Hybrid Systems*.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized Target Assignment and Path Finding Using Answer Set Programming. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1216–1223.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2019. Priority Inheritance with Backtracking for Iterative Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 535–542.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4(4): 392–399.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *Proceedings*

of the IEEE International Conference on Robotics and Automation (ICRA), 5628–5635.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 219: 40–66.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 195: 470–495.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 117–122.

Standley, T. S. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the AAAI Conference on Artificial Intelligence (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.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 151–159.

Surynek, P. 2019. Unifying Search-Based and Compilation-Based Approaches to Multi-Agent Path Finding through Satisfiability Modulo Theories. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1177–1183.

Svancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online Multi-Agent Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7732–7739.

Wagner, G. 2015. *Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning*. Ph.D. thesis, Carnegie Mellon University.

Wan, Q.; Gu, C.; Sun, S.; Chen, M.; Huang, H.; and Jia, X. 2018. Lifelong Multi-Agent Path Finding in a Dynamic Environment. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 875–882.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1752–1760.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1444–1449.