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] and **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 warehouses. We propose a new framework 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 finite time horizon and ignores collisions beyond it. Our framework is particularly well suited to generating pliable plans that adapt to continually arriving new goal locations. Theoretically, we analyze the advantages and disadvantages of our framework. Empirically, we evaluate our framework with a variety of MAPF solvers and show that it can produce high-quality solutions for up to 1,000 agents, significantly outperforming existing methods.

1 Introduction

Multi-Agent Path Finding (MAPF) is the problem of moving a team of agents on a graph 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], office robots [Veloso *et al.*, 2015], video game characters [Ma *et al.*, 2017b], and quadrotor swarms [Hönig *et al.*, 2018]. Today, in autonomous warehouses, mobile robots called *drive units* already autonomously move inventory pods or flat packages from one location to another [Wurman *et al.*, 2007; Kou *et al.*, 2020]. However, MAPF is only the "one-shot" variant of the actual problem in many real-world 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.*, 2017a] and characterized by agents constantly being assigned new goal locations.

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 *et al.*, 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 *et al.*, 2015; Ma *et al.*, 2017a; Liu *et al.*, 2019].

In this paper, we propose a new framework for solving lifelong MAPF where we decompose lifelong MAPF into a sequence of Windowed MAPF instances and replan paths once every h timesteps (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 planning horizon, and (2) collisions need to be resolved only for the first w timesteps ($w \geq h$ is user-specified). The benefit of this decomposition is two-fold. First, it keeps the agents continually engaged, avoiding idle time, and increasing throughput. Second, it generates pliable plans that adapt to continually arriving new goal locations. In fact, resolving all collisions within the entire time horizon is often unnecessary since the paths of the agents can change as new goal locations arrive.

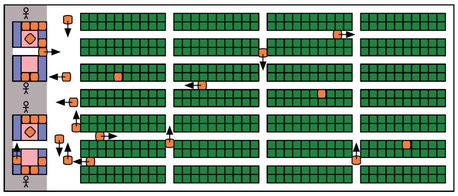
We evaluate our framework with various MAPF solvers, namely, CA* [Silver, 2005] (incomplete and suboptimal), PBS [Ma *et al.*, 2019] (incomplete and suboptimal), ECBS [Barer *et al.*, 2014] (complete and bounded suboptimal) and CBS [Sharon *et al.*, 2015] (complete and optimal). We show that, for each MAPF solver, using a bounded horizon yields similar throughput as using the full horizon but with significantly smaller runtime. We also show that our framework outperforms existing work and can scale up to 1,000 agents.

2 Background

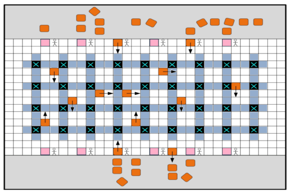
In this section, we first introduce several state-of-the-art MAPF solvers, and we then discuss several existing research on lifelong MAPF. We finally review the elements of the bounded horizon idea that have guided previous research.

2.1 Popular MAPF Solvers

CBS Conflict-Based Search (CBS) [Sharon *et al.*, 2015] is a popular two-level MAPF solver that is complete and optimal.



(a) Fulfillment warehouse map, borrowed from [Wurman *et al.*, 2007].



(b) Sorting center map, reproduced from [Wan *et al.*, 2018].

Figure 1: A well-formed fulfillment warehouse map and a non-well-formed sorting center map. In (a), the endpoints consist of green cells (representing locations that store inventory pods) and blue cells (representing the packing stations). In (b), the endpoints consist of 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).

At the high level, CBS starts with a root node which contains a shortest path for each agent (ignoring other agents). It then checks for collisions. It 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 new constraints. CBS repeats this procedure until finding a node with collision-free paths.

ECBS Enhanced CBS (ECBS) [Barer *et al.*, 2014] is a complete and bounded-suboptimal variant of CBS. The bounded suboptimality (that is, the solution cost is within a user-specified factor times the optimal solution 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.

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 in general, prioritized planning, is well known for its small runtime. However, it is incomplete and suboptimal.

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 the constraint it adds to each child node is that one of the agents involved in the collision has higher priority than the other agent. The low level of PBS is similar to CA* where it plans a shortest path that is consistent with the partial priority ordering generated by the high level. It outperforms many variants of prioritized planning solvers but is still incomplete and suboptimal.

2.2 Prior Work on Lifelong MAPF

Method (1) Nguyen *et al.* [2017] solves lifelong MAPF as a whole in an offline setting and formulates it as an answer set programming problem. However, in their paper, the method only scales up to 20 agents, each with only about 4 goal locations. This is not surprising because MAPF itself 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 to reuse search efforts. For example, Wan *et al.* [2018] proposes an incremental CBS which reuses the high-level tree of the previous 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] uses 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 (that is, many agents with many obstacles), almost all paths are affected, and thus it still needs to replan paths for all agents.

Method (3) A third method is similar to the second method, but restricts the replanning to planning paths for only 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 other agents. Hence, this method could degenerate to prioritized planning in the 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 *et al.* [2015] introduces the idea of the well-formed infrastructure to enable backtrack-free search. In a *well-formed* infrastructure, 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 Figure 1(a), may satisfy this well-formed requirement, but some other maps, such as Figure 1(b), may not. Moreover, additional mechanisms during path planning are required. For example, one needs to force the agents to "hold" their goal locations [Ma *et al.*, 2017a] or plan "dummy paths" for the agents [Liu *et al.*, 2019] after they reach their goal locations. Both alternatives cause unnecessarily longer 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 for 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 both the map and the MAPF model need to have additional structure to guarantee the 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 solve regular MAPF with CA*. He refers to it as WHCA* and empirically shows that, as the length of the horizon decreases, WHCA* runs faster but also generates longer paths. In this paper, we showcase the benefits of applying this idea to lifelong MAPF and to other types of MAPF solvers. In particular, our framework yields the benefits of lower computational costs for planning with bounded horizons, while continually keeping the agents busy, and yet making only a negligible compromise on the solution qualities. When executing lifelong MAPF plans on drive units, Hönig *et al.* [2019] uses a similar framework interleaving planning and execution. In such domains, our new framework can be incorporated to interleave planning and execution more effectively. Such interleaving is even used in video games [Sigurdson *et al.*, 2018].

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 are interested in 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. 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 plan to occupy the same location at the same timestep (called a vertex conflict in [Stern *et al.*, 2019]) or to 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*, that is, the average number of goal locations visited per timestep.

The task assignment is usually domain-dependent and could have constraints and preferences of its own in different domains. Therefore, we study the general case in which the task assigner is not necessarily within our control so that our path-planning system is applicable in many different domains. But, of course, for a particular domain, we can design a hierarchical framework that combines a domain-specific task assigner with our path-planning system. For example, the task assigners in [Ma *et al.*, 2017a; Liu *et al.*, 2019] for fulfillment warehouse applications and in [Grenouilleau *et al.*, 2019] for sorting center applications can be directly combined with our path-planning system. We also showcase two simple task assigners in our experiments. Moreover, the hierarchical framework is also usually a good way to achieve efficiency and improve scalability.

4 Framework

Our framework has two user-specified parameters, namely, the time horizon w and the replanning frequency h. The time horizon w specifies that the Windowed MAPF solver has to resolve collisions within a time horizon of w timesteps. The replanning frequency h specifies that the Windowed MAPF solver has to replan paths once every h timesteps. To execute successfully, the Windowed MAPF solver has to replan paths more frequently than once every w timesteps, i.e., w should be larger than or equal to h.

In every Windowed MAPF episode, say, starting at timestep t, we first update the start location s_i and the goal location sequence $\mathbf{g_i}$ for each agent a_i. We set the start location s_i of agent a_i to its location at timestep t. Then, we calculate the minimum number of steps d that agent a_i needs to visit all remaining locations in $\mathbf{g_i}$, i.e.,

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

where dist(x, y) is the distance between locations x and y and $|\mathbf{x}|$ is the cardinality of set $\mathbf{x}$. d being smaller than h indicates that agent a_i might finish visiting all its goal locations and being idle before the next planning episode starts at timestep $t + h$. To avoid this situation, we continually assign 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, we call a Windowed MAPF solver to find paths for all agents that are collision-free for the first w timesteps and that move them from their start locations through all their goal locations in the order given by their goal location sequences. Finally, we move the agents for h timesteps along the generated paths and remove the visited goal locations from the goal location sequences.

We use 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, the Windowed MAPF solvers need to be changed in two aspects: (1) each path needs to go through a sequence of goal locations, and (2) any two 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 Sequence of Goal Locations

All the MAPF solvers discussed in Section 2.1 use state-time A* [Silver, 2005] or any of its variants in their low-level searches to find a path for each agent from its start location to its unique goal location while satisfying some given spatio-temporal constraints that prohibit the agent from being at certain locations at certain timesteps. However, a characteristic feature of a Windowed MAPF solver is that, for each agent, it plans a path that goes through a sequence of goal locations. In fact, Grenouilleau *et al.* [2019] performs a truncated version of this adaptation in the study of the pickup and delivery problem. They propose Multi-Label A* that can find a path for a single agent that goes through two ordered goal locations, namely its assigned pick up location and goal location. In Algorithm 1, we generalize Multi-Label A* to a sequence of goal locations.

Algorithm 1: The low-level search for Windowed MAPF solvers. The difference from state-time A* is shown in blue.

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

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

Algorithm 1 uses the structure of state-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 corresponding path from the root node to node N has already visited. For example, $N.label = 2$ indicates that the corresponding path has 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 13-14]. In the main procedure, Algorithm 1 first creates the root node R with a label of 0 and pushes it into the prioritized queue $open$ [Lines 1-2]. While $open$ is nonempty [Line 4], the node P with the smallest f-value is selected for expansion [Line 5]. If P has reached its current goal location [Line 6], $P.label$ is increased by 1 [Line 7]. If $P.label$ equals the cardinality of the goal location sequence [Line 8], Algorithm 1 terminates and returns the corresponding path [Line 9]. Otherwise, it generates child nodes that respect the given spatio-temporal constraints [Lines 10-11]. The labels of the child nodes equal $P.label$. Checking for duplicates in $open$ 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 resolve collisions for only the first w timesteps. Beyond the first w timesteps, the solvers ignore collisions between agents and assume that each agent follows its individual shortest path to go through all its goal locations, which ensures that the agents still head in the correct directions. Modification details of the various MAPF solvers discussed in Section 2.1 are as follows.

Bounded-Horizon (E)CBS Both CBS and ECBS conduct 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 resolves any existing ones, bounded-horizon (E)CBS only finds collisions among all paths that occur in the first w timesteps and resolves any such existing ones. The remaining parts of (E)CBS stay the same. Since bounded-horizon (E)CBS need to resolve fewer collisions, their high-level trees can be substantially smaller than the high-level trees of standard (E)CBS.

Bounded-Horizon CA* CA* conducts search based on priorities, where an agent avoids collisions with all higher-priority agents. In the bounded-horizon variant of CA*, an agent is required to avoid collisions with all higher-priority agents but only for the first w timesteps. Therefore, when running state-time A* for each agent, we only consider the spatio-temporal constraints of 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*. In fact, 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 for the high level of PBS and incorporate the limited consideration of spatio-temporal constraints for its low level. As a result, bounded-horizon PBS has a smaller high-level tree and runs faster in the low level than standard PBS.

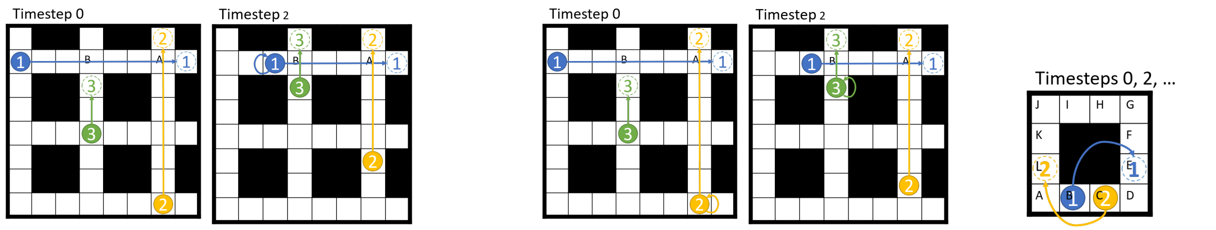
4.3 Behavior of Our Framework

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

Example 1. *Figure 2(a) shows a 3-agent example with $w = 4$ timesteps and $h = 2$ timesteps. At timestep 0, all agents follow their shortest paths as no collisions will occur for the first 4 timesteps. Then at timestep 2, a_3 reaches its goal location and is assigned a new goal location. This time, a_1 and a_3 will collide at cell B at timestep 3. So a_1 is forced to wait for one timestep. However, if we solve this example with $w = 8$ timesteps and $h = 2$ timesteps, as shown in Figure 2(b), we could generate paths that include more wait actions. At timestep 0, the solver finds a collision between a_1 and a_2 at cell A at timestep 7 and thus forces a_2 to wait for one timestep. Then, at timestep 2, the solver finds a collision between a_1 and a_3 at cell B at timestep 3 and forces a_3 to wait for one timestep. The number of total wait actions is 2.*

Similar cases are also found in our experiments: sometimes our framework with smaller w achieves higher throughput than the one with larger w. All of these cases support our claim that, in lifelong MAPF, resolving all collisions in the entire horizon may often do so unnecessarily, which is different from regular MAPF. Nevertheless, the bounded-horizon method also has a drawback - using a too small value for w may generate deadlocks, as shown in Example 2.

Example 2. *consider the example shown in Figure 2(c) with $w = 2$ timesteps and $h = 2$ timesteps and assume that we use*

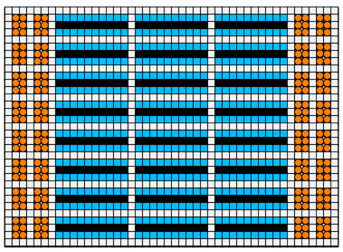


(a) A 3-agent instance with $w = 4$ timesteps. a_3 reaches its goal location at timestep 2 and then gets a new goal location.

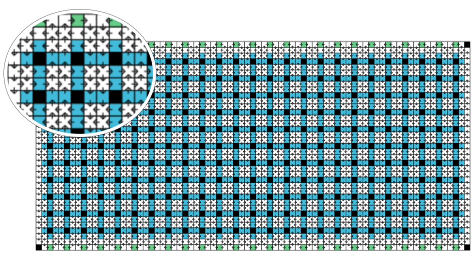
(b) The same instance as shown in (a) with $w = 8$ timesteps.

(c) A 2-agent instance with $w = 2$ timesteps.

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



(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.

an optimal Windowed MAPF solver (e.g., CBS). At timestep 0, the solver returns path [B, B, B, C, D, E] (of length 5) for a_1 *and path [C, C, C, B, A, L] (of length 5) for* 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-enter it, because they are both of flowtime larger than 10. Therefore, at timestep 2, both agents are still waiting at cells B and C. The solver finds exactly the same paths for both agents and force them to wait again. Therefore, the agents will wait at cells B and C forever and never reach their goal locations.*

5 Empirical Results

We implemented our framework in C++ with four Windowed MAPF solvers based on CBS, ECBS, CA* and PBS. We used SIPP [Phillips and Likhachev, 2011], an advanced variant of state-time A*, as the low-level solver for all solvers. For CBS and ECBS, we used Soft Conflict Safe Interval Path Planning (SCIPP) [Cohen *et al.*, 2019], a recent variant of SIPP that facilitates focal search and generally breaks ties in favor of a path with a lower number of collisions. We incorporated CA* with the random restart technique where once CA* fails to find any solutions, we repeatedly restart CA* with a new random priority ordering until it successfully finds a solution or reaches the runtime limit. For comparison, we also implemented two existing realizations of Method (3), namely, holding endpoints [Ma *et al.*, 2017a] and reserving dummy paths [Liu *et al.*, 2019]. We did not compare against Method (1) since it does not work with our online setting. In addition, since we choose dense environments (i.e., environments that have many obstacles and many agents) to stress test various methods, an explicit comparison with Method (2) was not required. This is because the performance of Method (2) in dense environments is similar to that of our framework with an infinite horizon. We simulate $T = 5{,}000$ timesteps for each experiment. All experiments were conducted on Amazon EC2 instances of type "m4.xlarge" with 16 GB memory.

Framework	$m = 60$	$m = 100$	$m = 140$
Our framework	2.33	3.56	4.55
Holding endpoints	2.17 (-6.80%)	3.33 (-6.33%)	4.35 (-4.25%)
Reserving dummy paths	2.19 (-6.00%)	3.41 (-4.16%)	4.50 (-1.06%)
Our framework	0.33 ± 0.01	2.04 ± 0.04	7.78 ± 0.14
Holding endpoints	0.01 ± 0.00	0.02 ± 0.00	0.04 ± 0.01
Reserving dummy paths	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). Numbers in the parentheses characterize throughput differences in terms of percentage compared to our framework. Numbers after "±" indicate standard deviations.

5.1 Order Fulfillment Warehouse Application

In this subsection, we introduce fulfillment warehouse domains that are typically well-formed infrastructures. Fulfillment warehouse problems are commonplace in warehouses and are characterized by horizontal corridors in the center of the map and working stations on the perimeter of the map. In such well-formed infrastructures, Method (3) is applicable and thus we compare our framework with both realizations of Method (3). We use the map in Figure 3(a) from [Liu *et al.*, 2019]. It is a 33×46 grid with 16% obstacles. The initial locations of agents are uniformly chosen at random from the orange cells, while the task assigner chooses the goal loca-

w	$m = 200$	$m = 300$	$m = 400$	$m = 500$	$m = 600$	$m = 700$	$m = 800$	$m = 900$	$m = 1000$
5	6.22 (-1.57%)	9.28 (-0.93%)	12.27 (-1.56%)	15.17 (-1.84%)	17.97 (-2.35%)	20.69 (-2.85%)	23.36	25.79	27.95
10	6.27 (-0.75%)	9.36 (-0.06%)	12.41 (-0.41%)	15.43 (-0.19%)	18.38 (-0.11%)	21.19 (-0.52%)	23.94	26.44	28.77
20	6.30 (-0.22%)	9.38 (+0.16%)	12.45 (-0.07%)	15.48 (+0.12%)	18.38 (-0.11%)	21.24 (-0.26%)	23.91	-	-
∞	6.32	9.36	12.46	15.46	18.40	21.30	-	-	-
5	0.13 ± 0.00	0.31 ± 0.00	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	0.16 ± 0.00	0.42 ± 0.00	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	0.22 ± 0.00	0.61 ± 0.00	1.36 ± 0.01	2.71 ± 0.01	5.11 ± 0.03	9.28 ± 0.06	17.46 ± 0.14	-	-
∞	0.28 ± 0.00	0.80 ± 0.01	1.83 ± 0.01	3.84 ± 0.03	7.63 ± 0.06	16.16 ± 0.17	-	-	-

Table 2: Average throughput (Rows 2-5) and average runtime (in seconds) per run (Rows 6-9) of our framework using PBS. We put "-" when it takes more than 1 minute for the Windowed MAPF solver to find solutions in any run. The numbers in the parentheses characterize throughput differences in terms of percentage compared to our framework with $w = \infty$. The numbers after "±" indicate standard deviations.

ECBS

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

CA*

w	$m = 200$	$m = 300$	$m = 400$
5	6.17 (-0.48%)	9.12 (-0.35%)	-
∞	6.20	9.16	-
5	0.21 ± 0.01	1.07 ± 0.10	-
∞	0.84 ± 0.02	2.58 ± 0.12	-

CBS

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

Table 3: Results of our framework using ECBS, CA* and CBS. All numbers are reported in the same format as in Table 2.

tions for agents uniformly at random from the blue cells.

For our method, we use a time horizon of $w = 20$ timesteps and replan every $h = 5$ timesteps. For the other two methods, we replan 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, our method outperforms the reserving dummy path method, which in turn outperforms the holding endpoints method. This is because, as we discussed in Section 2.2, Method (3) usually generates unnecessary waits or 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). This is 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 on this well-formed map, and are not applicable to all maps.

5.2 Logistic Sorting Center Application

In this subsection, we introduce sorting center domains that are typically not well-formed infrastructures (and thus Method (3) cannot be applied here). They are also commonplace in warehouses and are characterized by uniformly placed chutes in the center of the map and working stations on the perimeter of the map. We use the map in Figure 3(b). It is a 37×77 grid with 10% obstacles. The 50 green cells on the top and bottom boundaries represent working stations where humans put packages on the drive units (agents). The 275 black cells represent the chutes where drive units sit at one of the adjacent blue cells and drop the 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. We use a directed version of this map. The benefit of using a directed map is that MAPF solvers are rendered more efficient since they can avoid having to resolve swapping conflicts. This allows us to focus on the efficiency of the overall framework. Our handcrafted horizontal directions include two rows from left to right alternating with two rows from right to left, and our handcrafted vertical directions include two columns from top to bottom alternating with two columns from bottom to top. We replan paths every $h = 5$ timesteps.

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

6 Conclusions

In this work, we proposed a new framework to solve lifelong MAPF by decomposing it into a sequence of Windowed MAPF instances. We showed how to adapt regular MAPF solvers of different kinds to Windowed MAPF solvers, and we empirically demonstrated the success of our framework on fulfillment warehouse maps and sorting center maps. We achieved scalability for up to 1000 agents while also producing solutions of high quality, significantly outperforming existing Methods (1) and (2). Compared to Method (3), our framework not only works for general graphs but also yields better throughput. Overall, our framework works for general graphs, invokes replanning using a user-specified frequency, and is able to generate pliable plans that can not only adapt to an online setting but also avoid wasting computational effort in anticipating a distant future.

Our framework 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 bounded 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.

References

[Barer *et al.*, 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS)*, pages 19–27, 2014.

[Cáp *et al.*, 2015] Michal Cáp, Jirí Vokrínek, and Alexander Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 324–332, 2015.

[Cohen *et al.*, 2019] Liron Cohen, Tansel Uras, T. K. Satish Kumar, and Sven Koenig. Optimal and bounded-suboptimal multi-agent motion planning. In *Proceedings of the 12th International Symposium on Combinatorial Search (SoCS)*, pages 44–51, 2019.

[Grenouilleau *et al.*, 2019] Florian Grenouilleau, Willem-Jan van Hoeve, and John N. Hooker. A multi-label A* algorithm for multi-agent pathfinding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 181–185, 2019.

[Hönig *et al.*, 2018] Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

[Hönig *et al.*, 2019] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters*, 4(2):1125–1131, 2019.

[Kou *et al.*, 2020] Ngai Meng Kou, Cheng Peng, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Idle time optimization for target assignment and path finding in sortation centers. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[Liu *et al.*, 2019] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1152–1160, 2019.

[Ma *et al.*, 2017a] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 837–845, 2017.

[Ma *et al.*, 2017b] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 270–272, 2017.

[Ma *et al.*, 2019] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 7643–7650, 2019.

[Morris *et al.*, 2016] Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop on Planning for Hybrid Systems*, 2016.

[Nguyen *et al.*, 2017] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. Generalized target assignment and path finding using answer set programming. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1216–1223, 2017.

[Pearl and Kim, 1982] Judea Pearl and Jin H Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):392–399, 1982.

[Phillips and Likhachev, 2011] Mike Phillips and Maxim Likhachev. SIPP: safe interval path planning for dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635, 2011.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Sigurdson *et al.*, 2018] Devon Sigurdson, Vadim Bulitko, William Yeoh, Carlos Hernández, and Sven Koenig. Multi-agent pathfinding with real-time heuristic search. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pages 117–122, 2005.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 173–178, 2010.

[Stern *et al.*, 2019] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th International Symposium on Combinatorial Search (SoCS)*, pages 151–159, 2019.

[Svancara *et al.*, 2019] Jirí Svancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online multi-agent pathfinding. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 7732–7739, 2019.

[Veloso *et al.*, 2015] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4423–4429, 2015.

[Wan *et al.*, 2018] Qian Wan, Chonglin Gu, Sankui Sun, Mengxia Chen, Hejiao Huang, and Xiaohua Jia. Lifelong multi-agent path finding in a dynamic environment. In *Proceedings of the 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 875–882, 2018.

[Wurman *et al.*, 2007] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1752–1760, 2007.

[Yu and LaValle, 2013] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1444–1449, 2013.