Lifelong Multi-Agent Path Finding in Large-Scale Warehouses

Extended Abstract

Jiaoyang Li
University of Southern California
jiaoyanl@usc.edu

Andrew Tinka
Scott Kiesel
Joseph W. Durham
Amazon Robotics

T. K. Satish Kumar
Sven Koenig
University of Southern California

ABSTRACT

Multi-Agent Path Finding (MAPF) is the problem of moving a team of agents from their start locations to their goal locations without collisions. We study the lifelong variant of MAPF where agents are constantly engaged with new goal locations, such as in 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 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. 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.

KEYWORDS

Agent coordination; multi-agent path finding; multi-agent planning

ACM Reference Format:
Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. 2020. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 INTRODUCTION

Multi-Agent Path Finding (MAPF) is the problem of moving a team of agents in discrete timesteps on a graph from their start locations to their goal locations while avoiding collisions. MAPF has numerous real-world applications, such as autonomous aircraft-towing vehicles [9], office robots [13], video game characters [8], and quadrotor swarms [3]. Today, in autonomous warehouses, mobile robots already navigate autonomously to move inventory pods or flat packages from one location to another [4, 15]. 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* [7] and characterized by agents constantly being assigned new goal locations. In this paper, 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. 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.

Existing methods for solving lifelong MAPF include (1) solving it as a whole [10], (2) decomposing it into a sequence of MAPF instances at every timestep where one replans paths for all agents [2, 14], 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 [5, 7]. 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.

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 once every h timesteps. A *Windowed MAPF* instance is different from a MAPF instance in the following ways: (1) it allows an agent to be assigned a sequence of goal locations, and (2) collisions need to be resolved only for the first w timesteps ($w \geq h$).[1] 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, methods that resolve all collisions within the entire time horizon may often do so unnecessarily since the paths of the agents can change as new goal locations arrive.

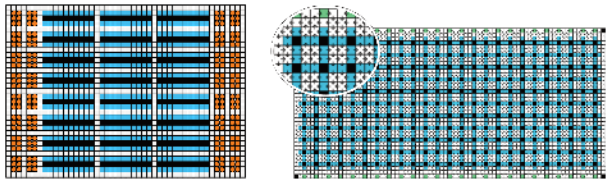
2 FRAMEWORK

Our framework has two parameters w and h ($w \geq h$). w specifies that the Windowed MAPF solver has to resolve collisions within a time window of w timesteps. h specifies that the Windowed MAPF solver has to replan once every h timesteps.

In every Windowed MAPF episode, say, starting at timestep t, we first update the start location and the goal location sequence for each agent. We set the start location of the agent to its location at timestep t. Then, we calculate the distance d from the start location to the first goal location plus the sum of the distances between

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, 1817189, and 1837779, as well as a gift from Amazon.
Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

[1] Resolving collisions only within a window is not a new idea. Silver [12] has already applied this idea to solving MAPF with prioritized planning. He refers to it as WHCA* and empirically shows that, as the length of the window decreases, WHCA* runs faster but also generates longer paths. In this paper, we showcase the benefits of applying this idea to lifelong MAPF and other types of MAPF solvers.



(a) Fulfillment warehouse map. (b) Sorting center map.

Figure 1: Two maps. Black cells represent obstacles, which the agents cannot traverse. Cells of other colors represent empty locations, which the agents can traverse.

Table 1: Throughput and average runtime per run in seconds. Here, a *run* means a call to the (Windowed) MAPF solver.

Agents	Holding endpoints		Dummy paths		Our framework	
	throughput	runtime	throughput	runtime	throughput	runtime
60	2.17	0.01	2.19	0.02	2.33	0.33
100	3.33	0.02	3.41	0.05	3.56	2.04
140	4.35	0.04	4.50	0.17	4.55	7.78

Table 2: Results of our framework using PBS, CA*, CBS, and ECBS. For each algorithm, the top rows report the throughput while the bottom rows report the average runtime per run in seconds. "-" indicates that the runtime of the Windowed MAPF solver exceeds 1 minute per run.

PBS									
Agents	200	300	400	500	600	700	800	900	1000
$w = 5$	6.22	9.28	12.27	15.17	17.97	20.69	23.36	25.79	27.95
$w = 10$	6.27	9.36	12.41	15.43	18.38	21.19	23.94	26.44	28.77
$w = 20$	6.30	9.38	12.45	15.48	18.38	21.24	23.91	-	-
$w = \infty$	6.32	9.36	12.46	15.46	18.40	21.30	-	-	-
$w = 5$	0.13	0.31	0.61	1.12	1.87	3.01	4.73	7.30	10.97
$w = 10$	0.16	0.42	0.89	1.66	2.91	4.81	7.79	12.66	21.31
$w = 20$	0.22	0.61	1.36	2.71	5.11	9.28	17.46	-	-
$w = \infty$	0.28	0.80	1.83	3.84	7.63	16.16	-	-	-

CA*				CBS			ECBS			
Agents	200	300	400	Agents	100	200	Agents	400	500	600
$w = 5$	6.17	9.12	-	$w = 5$	3.17	-	$w = 5$	12.03	14.79	17.28
$w = \infty$	6.30	9.16	-	$w = \infty$	-	-	$w = \infty$	12.28	15.20	-
$w = 5$	0.21	1.07	-	$w = 5$	0.14	-	$w = 5$	1.27	2.37	4.22
$w = \infty$	0.84	3.58	-	$w = \infty$	-	-	$w = \infty$	11.48	23.47	-

consecutive future goal locations in the goal location sequence. d being smaller than h indicates that the agent might finish visiting all its goal locations and being idle before the next planning episode starts at timestep $t + h$. To avoid this situation, the task assigner continually assigns new goal locations to the agent until $d \geq h$. Once we get the start locations and the goal location sequences of all agents, 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.

In order to find a shortest path for an agent to move through a sequence of goal locations, we propose a generalized variant of *Multi-Label A** [2] and use it as the single-agent pathfinding algorithm for our Windowed MAPF solvers. Compared to A*, we add an additional attribute *label* to each node N that indicates the number of goal locations in the goal location sequence that the corresponding path from the root node to N has already visited. The label of the root node is 0. We increase the label of a node by one iff its location is its next goal location. A node is a goal node iff its label equals the cardinality of the goal location sequence. In addition, we set the h-value of a node to the distance from its location to the next goal location plus the sum of the distances between consecutive future goal locations in the goal location sequence.

3 EMPIRICAL RESULTS

We evaluate our framework with various MAPF solvers implemented in C++, namely, CA* [12] (incomplete and suboptimal), PBS [6] (incomplete and suboptimal), ECBS [1] (complete and bounded suboptimal) and CBS [11] (complete and optimal). For comparison, we also implemented two existing realizations of Method (3), namely, holding endpoints [7] and reserving dummy paths [5]. We do not compare against Method (1) since it does not scale beyond 20 agents [10]. We do not compare against Method (2) since its performance in *dense environments* (that have many obstacles and many agents) is similar to that of our framework with $w = \infty$. We simulate 5,000 timesteps for each experiment. All experiments were conducted on Amazon EC2 instances of type "m4.xlarge" with 16 GB memory.

We first use the map in Figure 1(a) from [5] to demonstrate fulfillment warehouse applications. The initial locations of agents are chosen uniformly at random from the orange cells, while the task assigner chooses their goal locations uniformly at random from the blue cells. This map satisfies the structure requirement of Method (3), and we thus compare our framework with both existing realizations of Method (3). 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 at every timestep, as required by Method (3). All methods use PBS as their MAPF solvers. Table 1 reports the throughput and runtime of these methods. In terms of throughput, our method outperforms the two existing realizations of Method (3). In terms of runtime, however, our method is slower per run 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.

We then use the map in Figure 1(b) to demonstrate sorting center applications. Each agent is assigned green cells and blue cells alternately. In particular, the task assigner chooses blue cells uniformly at random and chooses green cells that are closest to the current locations of the agents. This map does not satisfy the structure requirement of Method (3). Table 2 reports the throughput and runtime of our framework for different values of w and $h = 5$ timesteps using PBS, CA*, CBS, and ECBS with a suboptimality bound of 1.1. As expected, the value of 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, the value of w significantly 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.

REFERENCES

[1] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. 2014. 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)*. 19–27.

[2] Florian Grenouilleau, Willem-Jan van Hoeve, and John N. Hooker. 2019. A Multi-Label A* Algorithm for Multi-Agent Pathfinding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*. 181–185.

[3] Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. 2018. Trajectory Planning for Quadrotor Swarms. *IEEE Transactions on Robotics* 34, 4 (2018), 856–869.

[4] Ngai Meng Kou, Cheng Peng, Hang Ma, T. K. Satish Kumar, and Sven Koenig. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*.

[5] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 1152–1160.

[6] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. 7643–7650.

[7] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. 2017. 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)*. 837–845.

[8] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. 2017. 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)*. 270–272.

[9] Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI Workshop on Planning for Hybrid Systems*.

[10] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. 2017. Generalized Target Assignment and Path Finding Using Answer Set Programming. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 1216–1223.

[11] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

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

[13] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 4423–4429.

[14] Qian Wan, Chonglin Gu, Sankui Sun, Mengxia Chen, Hejiao Huang, and Xiaohua Jia. 2018. Lifelong Multi-Agent Path Finding in a Dynamic Environment. In *Proceedings of the 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 875–882.

[15] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*. 1752–1760.