Online Multi-Agent Path Finding: New Results

Jonathan Morag, Ariel Felner, Roni Stern, Dor Atzmon, Eli Boyarski

Ben-Gurion University of the Negev
moragj@post.bgu.ac.il, felner@bgu.ac.il, {sternron, dorat, boyarske}@post.bgu.ac.il

Abstract

Online MAPF extends the classical Multi-Agent Path Finding problem (MAPF) by considering a more realistic problem in which new agents may appear over time. As online solvers are not aware of which agents will join in the future, the notion of snapshot-optimal was defined, where only current knowledge is considered. In this paper, we perform an extensive comparison between oracle-optimal solutions (where the solver is preinformed of future agents), snapshot-optimal solutions, and suboptimal solutions obtained by prioritised planning.

Introduction

In the *Multi-agent path finding* problem (MAPF) the task is to find a set of non-conflicting paths for multiple agents from their start vertex to their goal vertex on a given graph. MAPF can be found in real-world applications such as routing autonomous vehicles on road networks and multi-robot systems (Ma et al. 2019; Čáp, Vokřínek, and Kleiner 2015; Dresner and Stone 2008; Ho et al. 2019). Many efficient algorithms that solve the MAPF problem optimally for a large number of agents exist (Li et al. 2019; Lam et al. 2019; Gange, Harabor, and Stuckey 2019; Felner et al. 2017).

The standard MAPF problem assumes that all agents start moving at the same time and finish their task when arriving at their goals. However, in real-life applications, it is most likely that new agents will appear while some agents are still executing their paths, e.g., a new autonomous vehicle can get on the road. *Online MAPF* is a MAPF setting in which agents can appear at different times. A prominent work on this topic was done by Švancara et al. (2019). Hereafter, we refer to that paper as `S19`. Because of the online nature of this problem, a solver does not know in advance where and at what time new agents will appear. As a result, an existing agent may be routed in a way that seems efficient at the current point in time, but eventually results in many conflicts with agents that appear in the future. An *oracle-optimal* solution (also called offline-optimal) is the minimal cost solution among all possible solutions while having full knowledge about the schedule and location of appearance of new agents in the future. This definition is inadequate for online MAPF because the knowledge about the future is not available. Consequently, no online solver can guarantee to return an oracle-optimal solution (`S19`).

`S19` suggested a new type of optimality for online MAPF called *snapshot optimality*. A solution is *snapshot-optimal* if it is optimal for the current known agents, i.e., assuming no new agent will appear.[1] `S19` developed several algorithms for finding snapshot-optimal solutions. The quality of snapshot-optimal solutions in terms of the cost function, when compared to oracle (offline) optimal, has been analysed in term of the competitive ratio (asymptotic ratio of costs) (Ma 2021). However, it has not been empirically evaluated. So, it is an open question whether finding snapshot-optimal solutions is a good idea or not. This paper aims to answer this question. We evaluate the quality of snapshot-optimal solutions compared to oracle-optimal solutions both theoretically and empirically, showing the pros and cons of snapshot-optimality. Since even snapshot-optimality does not guarantee the best solution, sub-optimal algorithms may also be considered, as they are typically much faster than optimal algorithms. One such algorithm is Replan Single (`S19`) which is based on prioritised planning. We also compare the quality of Replan Single solutions with oracle-optimal and snapshot-optimal solutions.

A closely related setting to online MAPF is *multi-agent pickup and delivery* (Ma et al. 2017), called the *warehouse model* by Stern et al. (2019). In this online setting agents cannot join or leave the problem space. Instead, new pickup-and-delivery tasks appear over time.

Definitions and Background

The input to the *Multi-Agent Path Finding* problem (MAPF) is a tuple $\langle G, A \rangle$, where $G = (V, E)$ is a unit cost graph, and $A = \{a_1, ..., a_k\}$ is a set of k agents, where each agent a_i is associated with a start vertex $s_i \in V$ and a goal vertex $g_i \in V$ (Stern et al. 2019). A solution to a MAPF problem is a list $\pi = \{\pi_1, \ldots, \pi_k\}$ of individual agent paths such that each agent $a_i \in A$ is associated with a single path π_i from its start to its goal. π_i is a sequence of vertices such that $\pi_i(t) \in V$ is the planned vertex for agent a_i at time t, where time is discrete. The length of a path π_i is marked $|\pi_i|$ and defined

[1] This is reminiscent of the *free space assumption* (Koenig and Likachev 2002) which assumes that a cell in a grid that is not particularly known as an obstacle is considered free.

as the number of vertices (non-unique) in the path, minus 1. A MAPF solution is *valid* only if none of the paths within it conflict. Two paths π_i and π_j conflict if at any time t the two agents are planned to occupy the same vertex ($\pi_i(t) = \pi_j(t)$), which is called a vertex conflict, or are planned to swap their vertices ($\pi_i(t) = \pi_j(t-1) \wedge \pi_i(t-1) = \pi_j(t)$), which is called a swapping conflict (Stern et al. 2019). A common cost function in MAPF is the *sum of costs* (SOC). SOC is defined as the sum of the lengths of all individual paths, $SOC(\pi) = \sum_{i=1}^{k} |\pi_i|$. A solution is *optimal* if it is valid and has the minimal cost among all valid solutions.

Online MAPF

Most previous works on MAPF focused on the *offline MAPF* setting, where paths are found before the agents start their movement. As soon as a solution is chosen, it is assumed that the agents can execute that solution, without any modification during execution. In *online MAPF* (S19), new agents may appear over time and wish to join the problem space while existing agents are still executing their paths. This problem is relevant to real-world problems, such as autonomous intersections, robot warehouses, airport traffic etc.

The input for online MAPF $\langle G, \mathcal{A} \rangle$ contains a graph G, similarly to offline MAPF, but instead of a set of agents, it contains a stream of sets of agents $\mathcal{A} = (A_0, \ldots, A_n)$. At each time step t, a set A_t of 0 or more new agents (coupled with their start and goal vertices) is produced from the stream. n is the time of appearance of the last set of new agents. Each agent a_i is associated with a time of appearance $TOA(a_i)$. Accordingly, each path π_i in an online MAPF solution starts at that agent's time of appearance.

S19 defined two types of online MAPF: the *warehouse model* and the *intersection model*. In this work, we focus on the intersection model, where new agents appear over time, and when they reach their goal they disappear. Appearing agents are only revealed to an online solver at the time of their appearance. At $TOA(a_i)$ agent a_i may either immediately enter the underlying graph G at s_i or choose to wait a few time steps outside the problem space. This *wait* action incurs a cost similar to waiting inside G. This is akin to agents appearing from within a private garage at their start vertex, and leaving into one at their goal vertex. Practically, this can be implemented by having each agent's start and goal vertices changed to private vertices connected to the original graph with directed edges. While the stream of new agents $\mathcal{A}$ is not exhausted, time is incremented by 1. For each time step t the agents that are already in the problem space are advanced to their planned vertices at t, and $\mathcal{A}$ is queried to reveal the set A_t of new agents appearing at t. If A_t is not empty, a new solution π^t (that starts at time t) that contains paths for both the existing agents (from their current positions) and the new agents, must then be found.

A solution to online MAPF is a sequence of all the solutions found at the different times when new agents appeared $\Pi = \{\pi^0, ..., \pi^n\}$, where π^t is the solution found at time $0 \leq t \leq n$ (within π^t, π^t_i denotes the path for agent a_i found at time t). A partial solution $\pi[x : y]$ is the part of a solution π that is planned for time steps $x, x+1, ..., y$. The executed solution $Ex(\Pi)$ is the paths that the agents ended up following. It is derived from Π using partial paths. Formally $Ex(\Pi) = \pi^0[0 : t'_1 - 1] \circ \pi^{t'_1}[t'_1 : t'_2 - 1] \circ ... \circ \pi^n[n : \infty]$, where $\circ$ is the concatenation of partial paths and the t'_i sequence is the times where at least one agent appears.

An online MAPF solution Π is valid iff none of the paths in $Ex(\Pi)$ have a conflict. The cost of an online solution is the sum of costs of its executed solution ($SOC(Ex(\Pi))$).

Snapshot Optimal Online MAPF

An online solver for online MAPF does not know when and which agents may appear in the future. Consequently, a solution with a lower cost than that of the executed solution may exist (S19). An algorithm that always returns a solution for the current set of agents that is optimal, assuming no new agents appear in the future, is called *snapshot-optimal* (S19). Two snapshot-optimal algorithms for online MAPF have been suggested by S19:

(1) *Replan All*. Whenever new agents appear, a new solution is found by optimally re-planning for all agents from their current positions and all existing paths are discarded.

(2) *Online Independence Detection* (S19) (OID) is based on Independence Detection by Standley (2010). OID finds snapshot-optimal solutions, while attempting to minimise the number of times agents are re-planned for.

Sub-Optimal Online MAPF

Perhaps the simplest algorithm for solving online MAPF problems is *Replan Single* (S19). Replan Single involves finding an individual path for each agent as they appear, while avoiding the paths of all previous agents. This is similar to Prioritised Planning (Latombe 1991).

Windowed MAPF (Li et al. 2020) is another approach to solving Online MAPF suboptimaly. The solver provides solutions that are valid up to a certain time horizon. As time progresses, the time horizon is advanced accordingly.

Oracle-Optimal Online MAPF

Any online MAPF problem may be converted into an equivalent offline problem. This is done by informing the solver of all the agents that will appear in the future, rather than hiding them until their time of appearance. By optimally solving the equivalent offline problem, we can find the lowest cost possible for a solution to the online problem. We designate such a solution as an *oracle-optimal* solution. Naturally, online problems can not be solved offline in practice. However, the cost of the oracle-optimal solution is useful for theoretical and practical comparisons. We are interested in studying how close snapshot-optimal is to oracle-optimal.

Figure 1 demonstrates a scenario where the oracle-optimal solver would find a better solution than a snapshot-optimal solver. Agent a_1 appears at time 0 and wishes to travel from vertex s_1 to g_1. Agent a_2 appears at time 2, and wishes to travel from s_2 to g_2 ($= s_1$). The oracle-optimal solver would know about both agents in advance, and choose the path that goes around the obstacle for agent a_1 (π_{or1}, defined below, marked by a blue dashed arrow), and the direct path for agent a_2 (π_{or2}, marked by a solid red arrow).

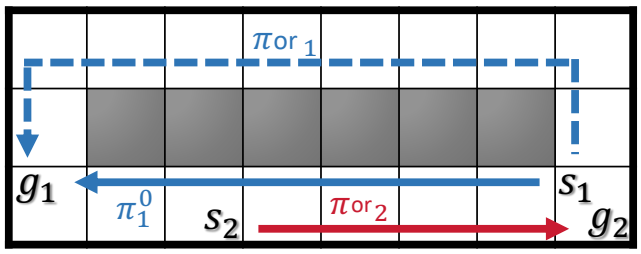

Figure 1: An illustrative example

Note that $|\pi_{or1}| = 11$, and $|\pi_{or2}| = 5$, resulting in an oracle-optimal solution π_{or} where $SOC(\pi_{or}) = 16$. The snapshot-optimal solver does not know about agent a_2 at time 0, and so it chooses the direct (shortest) path for a_1 (π_1^0 marked by a solid blue line, with $|\pi_1^0| = 7$). At time 1, agent a_1 has moved one cell to the left. At time 2, agent a_2 appears. Now, a new snapshot-optimal solution is chosen. a_2 waits outside the problem space until a_1 passes vertex s_2 ($|\pi_2^2| = 10$), while a_1 executes the remainder of its existing path ($|\pi_1^2| = 5$), so $SOC(\pi^2) = 15$. However, $SOC(Ex(\Pi)) = 17$ as it includes the steps taken by a_1 before a_2 appeared, meaning the snapshot-optimal solution has a higher cost than the oracle-optimal solution in this case.

Theoretical Comparison

Next, we present three observations regarding differences in costs of snapshot-optimal and oracle-optimal solutions. Their proofs are omitted and can be found in the supplementary materials [2].

Let P be an online MAPF problem. Let $\Pi_{or}(P)$ and $\Pi_{sn}(P)$ be the solutions to P generated by an oracle-optimal solver and snapshot-optimal solver, respectively. Let P^+ be an online MAPF problem that is identical to P except that there is an additional new agent a_i and $\forall a_j \neq a_i$ ($TOA(a_i) > TOA(a_j)$). Let $t^+ := TOA(a_i)$. Let π_i^* be a path for agent a_i that has minimal length while ignoring all other agents. Let $\Delta(P)$ be the difference in SOC between $\Pi_{sn}(P)$ and $\Pi_{or}(P)$, that is,

$$\Delta(P) = SOC(\Pi_{sn}(P)) - SOC(\Pi_{or}(P)).$$

Note that if $\Delta(P^+) > \Delta(P)$ it means that adding a new agent caused the cost difference to increase.

Observation 1. If $\Delta(P^+) > \Delta(P)$ then for every minimal-length path for the new agent there exists a conflict with the paths of the other agents in any $\Pi_{sn}(P)$.

Observation 2. If $\Delta(P^+) > \Delta(P)$ then for every snapshot-optimal solution $\Pi_{sn}(P^+)$ there exists at least one agent whose plan was made longer. For an old agent it means its new path is longer than its old path ($\Pi_{sn}(P)$). For a new agent a_i, it would mean its path is longer than π_i^*.

A solution Π is a prefix of Π' up to time t if all agents occupy the same vertices in both solutions at every time $t' < t$ meaning $\forall_i \forall_{t'<t} (Ex(\Pi)_i(t') = Ex(\Pi')_i(t'))$. This includes agents that have already disappeared.

[2] https://github.com/J-morag/Papers/blob/main/OnlineMAPF_SoCS22_Supplementary_Materials.pdf

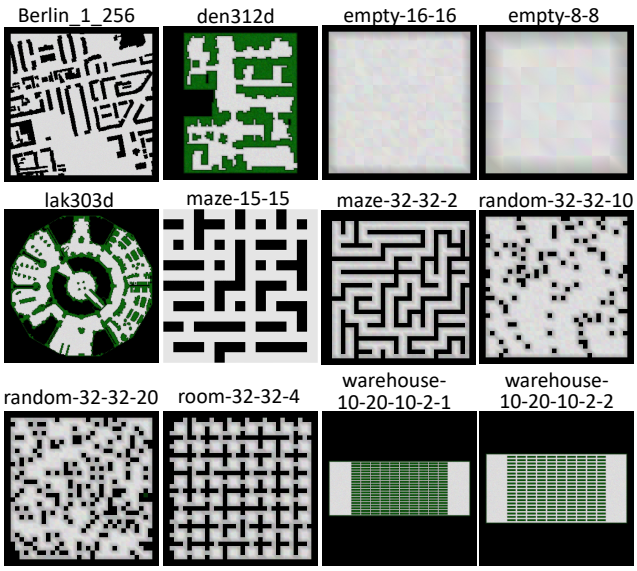

Figure 2: Maps used for experiments

Observation 3. If $\Delta(P^+) > \Delta(P)$ then for every oracle-optimal solution $\Pi_{or}(P^+)$, it holds that $\Pi_{sn}(P)$ is not a prefix of $\Pi_{or}(P^+)$ up to time step t^+.

The scenario in Figure 1 exemplifies these observations: (1) π_2^* conflicts with π_1^0. (2) Path $|\pi_2^2|$ is longer than $|\pi_2^*|$. (3) π^0 is not a prefix of the only oracle-optimal solution for this example (π_{or}) up to time 2.

From these observations we conclude that for a significant difference in quality to exist between an oracle-optimal solution and a snapshot-optimal solution for an online MAPF problem, that problem would have to contain many situations where the scenarios implied by the observations are met. We hypothesize that they are rare in practice, and conduct experiments to examine that hypothesis.

Experimental Comparison

We compared the quality of oracle-optimal, snapshot-optimal, and Replan Single solutions[3]. We used a diverse set of maps shown in Figure 2, representing the various map types and sizes from a common MAPF benchmark described in Stern et al. (2019), which contains a set of 4-connected grid-maps and a set of (offline) MAPF scenarios for each map. Each scenario had a list of start-goal pairs. We also used a 15x15 maze, for which we generated offline instances in a manner similar to the offline instances in the benchmark.

We created online MAPF problem instances based on these offline instances. We used two methods of selecting the start-goal pairs for the agents. **(1)** *Repeating method* where agents repeatedly used a fixed set of start-goal pairs. This simulates for example, gate-runway pairs at an airport. **(2)** *Uniform method* where agents were uniformly scattered. This simulates for example, cars driving in a city. The repeating method was done as follows:

[3] Our implementation is available at: https://github.com/J-morag/MAPF/tree/Online_MAPF

Map	Dist	λ	#	OO	SO	RS	0Δ	$\mu\Delta$	$M\Delta$
Berlin_1_256	R	0.05	49	12,025	12,025	12,025	98	0.01	0.01
		0.3	43	11,595	11,595	11,595	93	0.01	0.03
		0.6	45	11,640	11,641	11,641	96	0.03	0.05
		1	38	11,504	11,504	11,504	97	0.01	0.01
	U	0.05	49	11,459	11,459	11,460	98	0.01	0.01
		0.3	39	11,378	11,378	11,379	100	-	-
		0.6	39	11,462	11,462	11,463	95	0.01	0.01
		1	39	11,404	11,404	11,406	100	-	-
den-312d	R	0.05	45	2,879	2,879	2,880	93	0.06	0.10
		0.3	32	2,514	2,514	2,515	97	0.03	0.03
		0.6	24	2,485	2,485	2,487	96	0.05	0.05
		1	23	2,368	2,368	2,371	96	0.10	0.10
	U	0.05	47	3,143	3,143	3,146	96	0.06	0.07
		0.3	7	2,761	2,762	2,770	57	0.08	0.15
		0.6	2	2,464	2,466	2,473	0	0.08	0.12
		1	0	-	-	-	-	-	-
empty-8-8	R	0.05	50	230	230	230	94	0.34	0.39
		0.3	50	232	233	233	84	0.76	2.17
		0.6	49	232	233	235	78	1.38	3.73
		1	38	245	246	250	79	2.06	5.66
	U	0.05	50	235	235	235	96	0.45	0.47
		0.3	50	235	236	237	68	0.54	1.00
		0.6	50	236	237	239	42	0.74	1.38
		1	50	237	239	242	38	0.86	2.27
maze-15-15	R	0.05	50	570	571	571	94	0.34	0.44
		0.3	42	562	564	566	74	0.86	3.16
		0.6	21	532	536	543	71	2.23	6.97
		1	12	477	482	488	83	6.76	12.50
	U	0.05	50	604	604	606	70	0.26	0.87
		0.3	27	601	603	612	41	0.62	1.69
		0.6	0	-	-	-		-	-
		1	0	-	-	-		-	-
random-32-32-20	R	0.05	50	1,146	1,146	1,146	100	-	-
		0.3	50	1,148	1,148	1,150	86	0.12	0.19
		0.6	45	1,116	1,116	1,118	73	0.16	0.48
		1	37	1,109	1,111	1,115	70	0.34	0.95
	U	0.05	50	1,221	1,221	1,221	90	0.12	0.17
		0.3	49	1,226	1,227	1,230	49	0.18	0.41
		0.6	48	1,226	1,227	1,231	27	0.19	0.49
		1	43	1,210	1,212	1,218	28	0.20	0.61
warehouse-10-20-10-2-1	R	0.05	49	4,496	4,497	4,498	94	0.04	0.06
		0.3	49	4,498	4,499	4,500	90	0.08	0.18
		0.6	46	4,421	4,421	4,423	93	0.05	0.06
		1	43	4,501	4,501	4,504	93	0.02	0.03
	U	0.05	50	4,831	4,832	4,833	84	0.03	0.04
		0.3	45	4,769	4,770	4,774	67	0.06	0.12
		0.6	48	4,814	4,815	4,820	60	0.05	0.09
		1	42	4,828	4,829	4,834	71	0.06	0.12

Table 1: Comparison of the different algorithms

(1) Select an offline benchmark instance. Randomly select 20 start-goal pairs from the list for initializing agents. **(2)** Agents may appear at different frequencies. For every time step, draw the number of new agents that appear from a Poisson distribution. We experimented with different rates of appearance (average appearances per time step), which is expressed as different values for the λ parameter of the Poisson distribution. **(3)** Every time a new agent appears, draw a start-goal pair from a standard normal distribution over the fixed set of 20 pairs.

For the *uniform method* the selection was done differently. Step (1) was canceled. Step (2) was done in a similar manner. For step (3), draw a start-goal pair from a uniform distribution over *all* start-goal pairs from the list.

We compared the quality of oracle-optimal (OO), snapshot-optimal (SO), and Replan Single (RS) solutions on 50 problem instances per map and appearance rate. We tried appearance rates (λ) of 0.05, 0.3, 0.6, and 1 agents per time step (average). In each instance, appearance of new agents was stopped after a total of 50 agents had appeared. We set a time limit of 300 seconds to solve each instance. We tried both the repeated (R) and uniform (U) start-goal pair selection methods. The results for six representative maps are presented in Table 1 (results for more maps are available in the supplementary materials). For the instances solved by all solvers (sums shown under #), we report the average cost of solutions found by each solver. We also report the percentage of instances where no cost difference was observed between oracle-optimal and snapshot optimal solutions (0Δ), and the average ($\mu\Delta$) and maximal ($M\Delta$) Δ as a percent of the oracle-optimal cost, out of instances with $\Delta > 0$.

Most importantly, the cost of snapshot-optimal solutions was on average very close to the oracle-optimal cost, and in many cases (0Δ) was very large. This was true for both selection methods. Generally, as λ increases, the agents are more dense and therefore 0Δ decreases. Note that the set of solved instances (#) shrinks when increasing λ . Thus, the average solution cost may also decrease.

The repeating method was less dense because agents may follow the same pattern as their predecessors from the same start-goal pair. Thus, in general (depending on the map), its success rate and its 0Δ rate were higher than the uniform method where no such patterns exist. By contrast, the repeated method might suffer from large conflicts occurring repeatedly. Thus, its $\mu\Delta$ and $M\Delta$ were sometimes larger.

Additionally, we observe that Replan Single achieves average costs that are close to snapshot-optimal. In accordance with our previous observations, these costs are also close to oracle-optimal. The largest cost difference between Replan Single and oracle-optimal was 12 (3.3%), on maze-15-15 with $\lambda = 1.5$. A narrower experiment performed by `S19` showed similar results. These observations make Replan Single an appealing practical solver for online MAPF, as it is simple to understand and implement, has polynomial computational complexity (`S19`), and usually produces solutions that are close to oracle-optimal.

Summary and Future Work

The experimental results validate our claim that the scenarios implied by the observations above can be very rare. In view of these results, we conclude that snapshot-optimal algorithms seem very powerful as they are practical to implement and their solution quality is very close to that of the oracle (offline) optimal solution, which is the best possible solution. We also conclude that Replan Single is a practical alternative to snapshot-optimal when computation time is a primary concern, as it is very close in its solution quality, while also being simple to implement and efficient.

Future work may consider different cost functions that are uniquely relevant to online MAPF, such as fairness.

Acknowledgments

This research was sponsored by the United States-Israel Binational Science Foundation (BSF) under grant numbers 2017692 and 2021643, and by Israel Science Foundation (ISF) under grant number 844/17 and 210/17.

References

Čáp, M.; Vokřínek, J.; and Kleiner, A. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS)*.

Dresner, K.; and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research (JAIR)*, 31: 591–656.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search (SoCS)*.

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

Ho, F.; Geraldes, R.; Goncalves, A.; Rigault, B.; Oosedo, A.; Cavazza, M.; and Prendinger, H. 2019. Pre-flight conflict detection and resolution for UAV integration in shared airspace: Sendai 2030 model case. *IEEE Access*, 7: 170226–170237.

Koenig, S.; and Likachev, M. 2002. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, 476–483.

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

Latombe, J.-C. 1991. Multiple Moving Objects. In *Robot motion planning*, 1–57. Springer.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS*, 279–283.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T.; and Koenig, S. 2020. Lifelong multi-agent path finding in large-scale warehouses. *arXiv preprint arXiv:2005.07371*.

Ma, H. 2021. A Competitive Analysis of Online Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, volume 31, 234–242.

Ma, H.; Hönig, W.; Kumar, T. S.; Ayanian, N.; and Koenig, S. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, 7651–7658.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 837–845.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Švancara, 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)*, volume 33, 7732–7739.