Searching with Consistent Prioritization for Multi-Agent Path Finding[*]

Hang Ma[1], Daniel Harabor[2], Peter J. Stuckey[2], Jiaoyang Li[1], Sven Koenig[1]
[1]University of Southern California
[2]Monash University
hangma@usc.edu, {daniel.harabor,peter.stuckey}@monash.edu, {jiaoyanl,skoenig}@usc.edu

Abstract

We study prioritized planning for Multi-Agent Path Finding (MAPF). Existing prioritized MAPF algorithms depend on rule-of-thumb heuristics and random assignment to determine a fixed total priority ordering of all agents a priori. We instead explore the space of all possible partial priority orderings as part of a novel systematic and conflict-driven combinatorial search framework. In a variety of empirical comparisons, we demonstrate state-of-the-art solution qualities and success rates, often with similar runtimes to existing algorithms. We also develop new theoretical results that explore the limitations of prioritized planning, in terms of completeness and optimality, for the first time.

Introduction

Multi-Agent Path Finding (MAPF) is a coordination problem that arises in many applications, such as for aircraft-towing vehicles (Morris et al. 2016), warehouse and office robots (Wurman, D'Andrea, and Mountz 2008; Veloso et al. 2015), game characters (Ma et al. 2017c), and other multi-agent systems (Ma et al. 2017a). The problem is to plan collision-free paths for multiple agents on a given graph from their given start vertices to their given target vertices (Ma and Koenig 2017). The quality of a solution is measured by the flowtime (the sum of the arrival times of all agents at their target vertices) or the makespan (the maximum of the arrival times of all agents at their target vertices). MAPF is NP-hard to solve optimally (Yu and LaValle 2013b; Ma et al. 2016b). It can be solved with reductions to other well-studied combinatorial problems (Surynek 2015; Yu and LaValle 2013a; Erdem et al. 2013) and dedicated MAPF algorithms (Standley and Korf 2011; Luna and Bekris 2011; Goldenberg et al. 2014; Sharon et al. 2013; Wagner and Choset 2015; Sharon et al. 2015), as described in several surveys (Ma et al. 2016a; Felner et al. 2017).

Prioritized MAPF algorithms (Silver 2005; Sturtevant and Buro 2006) are among the most efficient ones for solving MAPF. They are based on the following simple prioritized-planning scheme (Erdmann and Lozano-Pérez 1987): Each agent is given a unique priority and computes, in priority order, a minimum-cost path from its start vertex to its target vertex that does not collide with the (already planned) paths of all agents with higher priorities. Existing (standard) prioritized MAPF algorithms are well known for their small runtimes and are often used as parts of MAPF solvers (Velagapudi, Sycara, and Scerri 2010; Wang and Botea 2011; Cáp, Vokrínek, and Kleiner 2015). However, they determine a predefined total priority ordering of the agents a priori and can thus result in solutions of bad quality or even fail to find any solutions for solvable MAPF instances, where a different total priority ordering could have resulted in solutions of higher quality.

In this paper, we thus generalize the notion of prioritized planning from planning with a fixed total priority ordering on the agents to planning with all possible total priority orderings. Theoretically, we establish, for the first time, a conceptual framework for discussing the limitations of prioritized planning. For example, we identify the set of solutions resulting from prioritized MAPF algorithms and characterize classes of MAPF instances with respect to the completeness and optimality guarantees for different priority orderings. We also develop two prioritized MAPF algorithms that improve the practicality of prioritized planning by systematically exploring "good" priority orderings. Our first algorithm, Conflict-Based Search with Priorities (CBSw/P), is an adaptation of Conflict-Based Search (CBS) (Sharon et al. 2015). It explores the space of all total priority orderings lazily using a systematic best-first search and introduces an ordered pair of agents only when their paths collide. Our second algorithm, Priority-Based Search (PBS), explores the space of all total priority orderings lazily using a systematic depth-first search. It can take a user-specified partial priority ordering as input, dynamically adds new ordered pairs of agents to it, and plans paths that are consistent with the resulting partial priority ordering. We show that standard prioritized MAPF algorithms are a special case of PBS. Empirically, we evaluate our prioritized MAPF algorithms on a large number of MAPF instances. We find that CBSw/P often computes optimal or

[*]The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189 and 1837779 as well as a gift from Amazon. 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.

near-optimal solutions and is more efficient than a state-of-the-art version of CBS (Felner et al. 2018). PBS also computes near-optimal solutions and is much more efficient than CBSw/P. Moreover, PBS remains near-optimal and efficient for MAPF instances with more than one hundred agents, finds solutions for many MAPF instances where standard prioritized MAPF algorithms cannot, and solves well-formed MAPF instances with six hundred agents in less than a minute.

Problem Definition

We formalize MAPF as follows: We are given a connected undirected graph $G = (V, E)$ and M agents $\{a_i \mid i \in [M]\}$ ($[M] = \{1, \ldots, M\}$). Each agent a_i has a unique start vertex $s_i \in V$ and a unique target vertex $t_i \in V$. At each discrete time $t = 0, \ldots, \infty$, each agent either moves to an adjacent vertex or waits at the same vertex. Let $\pi_i(t)$ be the vertex occupied by agent a_i at time t. A *plan* consists of a set of paths, one path $\pi_i = \langle \pi_i(0), \ldots, \pi_i(T_i), \pi_i(T_i + 1), \ldots \rangle$ for each agent a_i, where $\pi_i(0) = s_i$ and $\pi_i(t) = t_i$ for all times $t = T_i, \ldots, \infty$. Specifically, the *arrival time* T_i of agent a_i at its target vertex is defined to be the earliest time when it has reached its target vertex and stops moving. A *vertex collision* is a tuple $\langle a_i, a_j, v, t \rangle$ where agents a_i and a_j occupy the same vertex v at the same time t. An *edge collision* is a tuple $\langle a_i, a_j, u, v, t \rangle$ where agents a_i and a_j traverse the same edge (u, v) in opposite directions at the same time t. A solution is a plan that consists of collision-free paths for all agents. Its quality is measured by the *flowtime* $\sum_{i \in [M]} T_i$, defined to be the sum of the arrival times of all agents.

Prioritized Planning

Prioritized planning (Erdmann and Lozano-Pérez 1987) is a decoupled approach for MAPF where agents are ordered by importance according to a predefined total priority ordering. The idea is simple: One can plan for each agent individually rather than having to compute a plan for all agents simultaneously (as is the case for coupled MAPF algorithms). This means that one plans for the highest priority agent first and computes its individually optimal path; i.e. it avoids only the fixed obstacles. One then plans for lower and lower priority agents and computes for each agent its individually optimal paths that avoids not only the fixed obstacles but also collisions with the (already planned) paths of all higher priority agents (treated as dynamic obstacles). We generalize prioritized planning for MAPF by using partial priority ordering.

Definition 1. A *priority ordering* $\prec$ is a strict partial order on $[M]$. Agent a_i is of higher priority than agent a_j iff $i \prec j$.

It does not offer completeness or optimality guarantees. It is nevertheless popular because of its efficiency. Its main challenge is to determine a good priority ordering $\prec$ since a bad one can result in solutions of low quality or even failures in solving the problem. Global orderings assign fixed priorities to all agents a priori and resolve all collisions before movement begins. Priorities can be assigned arbitrarily (Warren 1990; Bennewitz, Burgard, and Thrun 2002;

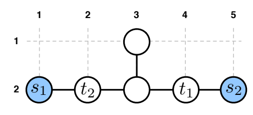


Figure 1: This MAPF instance is not solvable with any fixed priority ordering.

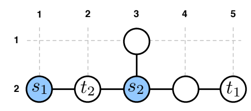


Figure 2: This MAPF instance is P-solvable and can be solved only with priority ordering $\{1 \prec 2\}$.

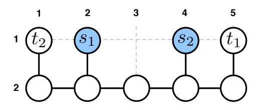


Figure 3: This MAPF instance is *well-formed* and P-solvable for any total priority ordering.

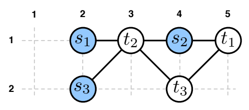


Figure 4: This MAPF instance is OP-solvable but prioritized planning is not guaranteed to find a solution.

Silver 2005) or derived from the problem at hand (Erdmann and Lozano-Pérez 1987). For example, priorities can also be computed using heuristics such as the distances to their target vertices (Van Den Berg and Overmars 2005) or by preferring certain types of paths over others (Buckley 1989; Ferrari et al. 1998). Local orderings assign temporary priorities to some agents in order to resolve collisions on-the-fly. Such algorithms require agents to follow their assigned paths and, when an impasse is reached, priorities are assigned dynamically to determine who waits (O'Donnell and Lozano-Pérez 1989; Azarm and Schmidt 1997). Some existing works attempt to reason over the space of all total priority orderings, which is intractable in general as there are $M!$ permutations. Bennewitz, Burgard, and Thrun (2002) explore some of this space by generating several total priority orderings randomly as part of a hill-climbing scheme. Azarm and Schmidt (1997) enumerate all total priority orderings for up to three agents.

Theoretical Results

We now analyze the effectiveness of prioritized planning, in terms of completeness and optimality, on different classes of MAPF instances. We first generalize a well known result (Erdmann and Lozano-Pérez 1987) to priority planning with a partial priority ordering.

Theorem 1. Prioritized planning with an arbitrary priority ordering $\prec$ is incomplete for MAPF in general.

Proof. The only three possible priority orderings for the counter-example shown in Figure 1 are $\{1 \prec 2\}$, $\{2 \prec 1\}$ or $\emptyset$. Prioritized planning for none of them results in a solution. □

Next, we define a class of MAPF instances which we call *P-solvable*. A MAPF instance is in this class iff it has a solution that can be computed with prioritized planning, that is, a fixed priority ordering exists where higher priority agents never wait for lower priority agents.

Definition 2. A solution $L = \{\pi_i \mid i \in [M]\}$ is *consistent* with a priority ordering $\prec$ if, for all pairs of agents where

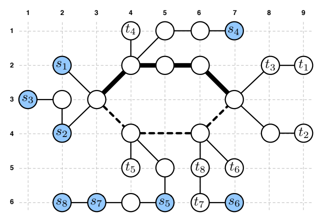


Figure 5: The only flowtime optimal solution for this instance is inconsistent with any fixed priority ordering $\prec$.

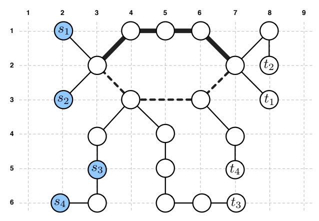


Figure 6: The only makespan optimal solution for this instance is inconsistent with any fixed priority ordering $\prec$.

$i \prec j$, we can never improve the arrival time of a_i at t_i by removing a_j from the set of agents.

Definition 3. A MAPF instance is **P-solvable** iff there exists a solution $L = \{\pi_i \mid i \in [M]\}$ that is *consistent* with some priority ordering $\prec$.

For example, there does not exist any solution that is consistent with any fixed priority ordering for the MAPF instance shown in Figure 1. Thus, the MAPF instance shown in Figure 1 is not P-solvable. The next result says that there exist P-solvable MAPF instances with solutions that are consistent with only one total priority ordering.

Theorem 2. Prioritized planning with any given priority ordering $\prec$ is incomplete for the class of P-solvable MAPF instance.

Proof. Figure 2 shows a P-solvable MAPF instance that has only one optimal solution, namely

$$\pi_1 = \langle\, (1,2), (2,2), (3,2), (4,2), (5,2)\,\rangle$$
$$\pi_2 = \langle\, (3,2), (3,1), (3,2), (2,2)\,\rangle.$$

This solution is only consistent with the total priority ordering $\{1 \prec 2\}$. Prioritized planning for any other priority orderings does not result in a solution. □

The next result focuses on *well-formed* instance, a class of practical and P-solvable MAPF problems that is important for warehouse logistics (Cáp, Vokrínek, and Kleiner 2015; Ma et al. 2017b). Figure 3 shows an example. The distinguishing feature of this class is that each agent can wait indefinitely at its start vertex and target vertex without blocking any other agent.

Theorem 3. Prioritized planning with any given total priority ordering $\prec$ is complete for the class of *well-formed* MAPF instances.

Proof. Given a well-formed MAPF instance and any total priority ordering $\prec$, a consistent solution can always be computed as follows: every agent waits at its start vertex until all higher priority agents arrive at their target vertices. Then, the agent follows an individually optimal path from its start vertex to its target vertex. □

The next result says that, for some P-solvable MAPF instances (including well-formed ones), prioritized planning can produce suboptimal solutions for all given total priority orderings.

Theorem 4. Prioritized planning is suboptimal for the flowtime objective in general for the class of P-solvable MAPF instances.

Proof. (Sketch) The counter-example shown in Figure 5 admits an inconsistent optimal solution L^* with flowtime $=$ 48. We claim that there exists no priority ordering $\prec$ which can produce a consistent prioritized solution with flowtime < 49. We sketch L^* as follows:

- $1 \prec 2$ at location $(3,3)$ at time 1.
- a_1 takes the bold path and arrives at $(6,2)$ at time 4.
- a_2 takes the dashed path and arrives at $(6,4)$ at time 4.
- a_3 takes the bold path and arrives at $(6,2)$ at time 6.
- $2 \prec 1$ at location $(7,3)$ at time 5 (L^* inconsistent)
- a_2 reaches its target vertex at time 7; and a_1 and a_3 reach their target vertices at time 8.
- All other agents follow their individually optimal paths.

The proof is by enumeration but the result only depends on the choice of paths for agents a_1, a_2 and a_3. All other agents follow their individually optimal paths, waiting or not, depending on their interactions with agents a_1, a_2 and a_3. For example, if agent a_1 chooses the dashed (versus) path it avoids another crossing with agent a_2 and speeds up its arrival at $(7,3)$. However, this requires introducing at least two delays due to contention with agents a_5 and a_6. Meanwhile, agent a_2 attempts to follow agent a_1 but this introduces new delays from agents a_5, a_6, a_7 and a_8. Alternatively, agent a_2 can switch to the bold path, which is slower and introduces a delay due to contention with agent a_4. Both choices lead to solutions with flowtime > 48. □

Corollary 5. Theorem 4 holds for the makespan objective.

Proof. (Sketch) The counter-example shown in Figure 6 admits an inconsistent optimal solution and makespan$=7$. The solution requires $1 \prec 2$ at $(3,2)$ and $2 \prec 1$ at $(7,2)$. There exists no priority ordering $\prec$ which can produce a consistent prioritized solution with makespan< 8. The proof is similar to the one of Theorem 4. □

We now focus on a class of MAPF instances for which there exist an optimal prioritized solution that is also optimal in general. We refer to such problems as *OP-solvable*.

Definition 4. A MAPF instance is **OP-solvable** iff: (1) it admits a solution L^* that is consistent with some priority ordering $\prec^*$ and; (2) L^* is optimal among all solutions, whether consistent or not.

The next result says that prioritized planning may not find an optimal solution or indeed any solution, even for problems which are OP-solvable.

Theorem 6. Prioritized planning with any given fixed total order $\prec$ is incomplete in general for the class of problems that are *OP-solvable*.

Proof. The counter-example shown in Figure 4 admits a consistent optimal solution (for both the flowtime and makespan objectives) for only the fixed total order $\{1 \prec 3 \prec 2\}$. The solution is:

$$\pi_1 = \langle\, (2,1), (3,1), (4,2), (5,1)\,\rangle$$
$$\pi_3 = \langle\, (2,2), (2,2), (3,1), (4,2)\,\rangle$$
$$\pi_2 = \langle\, (4,1), (4,1), (4,1), (3,1)\,\rangle.$$

However, the highest priority agent has another individually optimal path available which involves moving to location $(4, 1)$ at time 2, which results in a deadlock for agent a_2. $\square$

To summarize: (1) Some MAPF instances that are solvable are not solvable with prioritized planning. (2) Some MAPF instances that are solvable with prioritized planning are only solvable with prioritized planning for a single total priority ordering. (3) Some MAPF instances that are solvable with prioritized planning are not optimally solvable with prioritized planning for any total priority ordering. (4) Even worse, some MAPF instances that are optimally solvable with prioritized planning require prioritized planning not only to use the correct total priority ordering but also break ties correctly when planning paths for the agents, which—if done incorrectly—can prevent prioritized planning from finding any solution.

Conflict-Based Search with Priorities

(Standard) Conflict-Based Search (CBS) is a two-level algorithm that minimizes the flowtime. Conflict-Based Search with Priorities (CBSw/P) is an adaptation of CBS to prioritized planning. Like CBS, CBSw/P performs a best-first search on the high level to resolve collisions among the agents and thus builds a constraint tree (CT). Each CT node N contains a set of constraints $N.constraints$, a plan $N.plan$ (paths for all agents) that obeys these constraints, and a cost $N.cost$ equal to the sum of the arrival times of all paths in its plan at their target vertices. CBSw/P always expands the CT node with the smallest cost. Unlike CBS, CBSw/P also stores a priority ordering $\prec_N$ in each CT node N and only generates child CT nodes N' whose priority orderings $\prec_{N'}$ extend $\prec_N$ when it expands a parent CT node N.[1]

Definition 5. A priority ordering $\prec_A$ *extends* a priority ordering $\prec_B$ if $\forall i, j \in [M] : i \prec_B j \implies i \prec_A j$, that is, $\prec_A$ maintains all priority information of $\prec_B$.

Algorithm 1 shows the high-level search of CBSw/P. Red lines are not used in CBS. On the high level, CBSw/P starts with the root CT node, that has an empty set of constraints and an empty priority ordering [Lines 1-2]. It performs a low-level search to find an individually optimal path for each agent (without any constraints) independently. The plan of the root CT node thus contains paths for all agents [Line 3], and its cost is the sum of the arrival times of all paths [Line 4]. When CBSw/P expands a CT node N, it checks whether the CT node contains a plan without collisions [Line 9]. If this is the case, N is a goal node and CBSw/P terminates successfully [Line 10]. Otherwise, CBSw/P chooses a collision to resolve [Line 11] (CBSw/p follows the strategy used in (Felner et al. 2018) to choose a collision) and attempts to generate two candidate child CT nodes N_1 and N_2 that correspond to the ordered pairs $j \prec i$ and $i \prec j$, respectively [Line 12]. It actually generates the child CT node N_1 (N_2), iff the priority ordering $\prec_N$ of its parent CT node N does not contain the reversal $i \prec j$ ($j \prec i$)

```
Algorithm 1: High-Level Search of CBSw/P
   Input: MAPF instance
1  $Root.constraints \leftarrow \emptyset$;
2  $\prec_{Root} \leftarrow \emptyset$;
3  $Root.plan \leftarrow$ path for each agent found by a low-level search;
4  $Root.cost \leftarrow$ sum of the arrival times in $Root.plan$;
5  OPEN $\leftarrow \{Root\}$;
6  while OPEN $\neq \emptyset$ do
7      $N \leftarrow \arg\min_{N' \in \text{OPEN}} N'.cost$;
8      OPEN $\leftarrow$ OPEN $\setminus \{N\}$;
9      if $N.plan$ has no collision then
10         return $N.plan$
11     C $\leftarrow$ a vertex or edge collision $\langle a_i, a_j, \dots \rangle$ in $N.plan$;
12     foreach $a_i$ involved in $C$ do
13         if $i \not\prec_N j$ ($a_j$ is the other agent involved in $C$) then
14             $N' \leftarrow$ new node;
15             $N'.plan \leftarrow N.plan$;
16             $N'.constraints \leftarrow N.constraints \cup \{\langle a_i, \dots \rangle\}$;
17             $\prec_{N'} \leftarrow \prec_N$;
18             if $j \not\prec_{N'} i$ then
19                 $\prec_{N'} \leftarrow \prec_{N'} \cup \{j \prec i\}$;
20             Update $N'.plan$ by invoking a low-level search for $a_i$;
21             if a path is returned by the low-level search then
22                 $N'.cost \leftarrow$ sum of the arrival times in $N'.plan$;
23                 OPEN $\leftarrow$ OPEN $\cup \{N'\}$;
24 return "No Solution";
```

of its corresponding ordered pair $j \prec i$ ($i \prec j$) [Line 13]. Each child CT node inherits the plan, all constraints, and the priority ordering from N [Line 15-17]. If the collision to resolve is a vertex collision $\langle a_i, a_j, v, t\rangle$, CBSw/P adds the vertex constraint $\langle a_i, v, t\rangle$ to N_1 (if it is generated) to prohibit agent a_i from occupying v at time t and similarly adds the vertex constraint $\langle a_j, v, t\rangle$ to N_2 (if it is generated). If the collision to resolve is an edge collision $\langle a_i, a_j, u, v, t\rangle$, CBSw/P adds the edge constraint $\langle a_i, u, v, t\rangle$ to N_1 (if it is generated) to prohibit agent a_i from moving from u to v at time t and similarly adds the edge constraint $\langle a_j, v, u, t\rangle$ to N_2 (if it is generated) [Line 16]. For each child CT node, say N_1, CBSw/P adds its corresponding ordered pair $j \prec i$ to its priority ordering $\prec_{N_1}$ if the ordered pair is not in $\prec_{N_1}$ yet [Lines18-19]. CBSw/P uses the same low-level search as CBS (space-time A*) to find an individually optimal path for an agent a_i that respect all constraints in $N_1.constraints$ relevant to agent a_i. If a new path is found, CBSw/P replaces the old path of agent a_i in $N_1.plan$ with the new path returned by the low-level search [Line 20], updates the cost of N_1 accordingly, and thus inserts N_1 into OPEN [Lines 22-23].

Properties. CBSw/P adds a new partially ordered pair to the priority ordering of the child CT node whenever it *splits*, namely generates both child CT nodes of, a parent CT node. Therefore, the number of splitting in any branch of the CT is $O(M^2)$ (the number of all possible ordered pairs). However, the high-level search of CBSw/P is unbounded because the number of all possible collisions between two agents, for each of which a CT node is also expanded, is not finite.

Priority-Based Search

Priority-Based Search (PBS) is a two-level algorithm for prioritized planning. It performs a depth-first search on the high level to dynamically construct a priority ordering and thus builds a priority tree (PT). Conceptually, when it is

[1]We show in the pseudocode how to keep track of $\prec_N$ for each CT node N. However, we do not use the notation $N.\prec_N$ but treat it as a global variable for ease of readability.

Algorithm 2: High-Level Search of PBS (with initial priority ordering $\boldsymbol{\prec}_0$)

```
   Input: MAPF instance, $\boldsymbol{\prec}_0 (= \emptyset$ by default)
1  $\boldsymbol{\prec}_{Root} \leftarrow \boldsymbol{\prec}_0$;
2  $Root.plan \leftarrow \emptyset$;
3  foreach $i \in [M]$ do
4      $success \leftarrow UpdatePlan(Root, a_i)$;   /* success is always true if $\boldsymbol{\prec}_0 = \emptyset$ */
5      if $\neg success$ then
6          return "No Solution";
7  $Root.cost \leftarrow$ sum of the arrival times in $Root.plan$;
8  STACK $\leftarrow \{Root\}$;
9  while STACK $\neq \emptyset$ do
10     $N \leftarrow$ top node in STACK;
11     STACK $\leftarrow$ STACK $\setminus \{N\}$;
12     if $N.plan$ has no collision then
13         return $N.plan$;
14     C $\leftarrow$ first vertex or edge collision $\langle a_i, a_j, \ldots \rangle$ in $N.plan$ ;
15     foreach $a_i$ involved in $C$ do
16         $N' \leftarrow$ new node;
17         $N'.plan \leftarrow N.plan$;
18         $N'.constraints \leftarrow N.constraints \cup \{\langle a_i, \ldots \rangle\}$;
19         $\boldsymbol{\prec}_{N'} \leftarrow \boldsymbol{\prec}_N \cup \{j \prec i\}$;
20         $success \leftarrow UpdatePlan(N', a_i)$;
21         if $success$ then
22             $N'.cost \leftarrow$ sum of the arrival times in $N'.plan$;
23     Insert new nodes $N'$ into STACK in non-increasing order of $N'.cost$;
24 return "No Solution";
25 Function $UpdatePlan(N, a_i)$
26     LIST $\leftarrow$ topological sorting on partially ordered set
           $(\{i\} \cup \{j | i \prec_N j\}, \boldsymbol{\prec}_N)$;
27     foreach $j \in$ LIST do
28         if $j = i$ or $\exists a_k : k \prec_N j$, $a_j$ collides with $a_k$ in $N$.plan then
29             Update $N.plan$ by invoking a low-level search for $a_j$ that avoids
                   colliding with all agents $a_k$ with higher priorities ($k \prec_N j$);
30             if no path is returned by the low-level search then
31                 return $false$;
32     return $true$;
```

faced with a collision, PBS greedily chooses which agent should be given a higher priority. It backtracks efficiently and explores other branches iff there is no solution in the current branch. It thus effectively incrementally constructs a single partial priority ordering until it finds no collisions. PBS shares some similarities with CBSw/P on the high level. Like CBSw/P, PBS splits a PT node and introduces an additional ordered pair only if two agents collide. Unlike CBSw/P, PBS does not store constraints in the PT nodes but maintains the invariant that, if agent a_i has a higher priority than agent a_j ($i \prec_N j$), there are no collisions between them whenever PBS processes a PT node N.

Algorithm 2 shows the high-level search of PBS. We now point out its differences from CBSw/P. On the high level, PBS starts with the root PT node, that contains an initial priority ordering $\boldsymbol{\prec}_0$ [Line 1]. For (standard) PBS, this is the empty priority ordering $\emptyset$. It then calls Function *UpdatePlan*(N, a_i) to find an individually optimal path for each agent a_i [Line 4] and always succeeds with the empty initial priority ordering [Line 5]. When PBS expands a PT node N, PBS always generates two child PT nodes N_1 and N_2 that correspond to the ordered pairs $j \prec i$ and $i \prec j$, respectively [Line 15], since agent a_i does not collide with agent a_j if they are comparable with respect to $\boldsymbol{\prec}_N$ ($i \prec_N j$ or $j \prec_N i$). For each child PT node, say N_1, PBS calls Function *UpdatePlan*(N_1, a_i) to find an individually optimal path for agent a_i that avoids collisions with the paths of all higher priority agents [Line 20]. PBS favors the child PT node with the smallest cost and thus inserts the child PT nodes (if any are generated) into the stack in non-increasing order of their costs [Line 23].

Function *UpdatePlan*(N, a_i). Function *UpdatePlan*(N, a_i) finds an individually optimal path for agent a_i that avoids collisions with the paths of all higher priority agents and maintains the invariant that the path of any agent a_j with a lower priority still obeys $\boldsymbol{\prec}_N$ (finds a new path for such an agent a_j if necessary). To do so, PBS performs a topological sort on i and all j with $i \prec_N j$ [Line 26]. For each j in the topologically sorted order, PBS then performs a low-level search for agent a_j to compute a new individually optimal path that respects priority ordering $\boldsymbol{\prec}_N$ if it collides with an agent a_k with a higher priority [Lines 28-29].

Low-Level Search. On the low level, PBS uses a special low-level search to find an individually optimal path for agent a_j that does not collide with the path of any a_k with a higher priority (i.e., $k \prec_N j$). The number of constraints can be infinite since such agent a_k stays in its target vertex t_k forever after its arrival time T_k. This is different from the low-level search of CBSw/P where the number of constraints is finite. Therefore, there might be no path that reaches the target vertex t_j. PBS thus uses a space-time A* for all times not greater than the maximum of the arrival times of the higher priority agents a_k (i.e., $k \prec_N j$) only and switches to standard A* (without the time dimension and wait actions afterward). CBS breaks ties to favor a path that collides with the fewest paths of other agents, which has been empirically shown to make its high-level search more efficient (Sharon et al. 2015). Similarly, PBS breaks ties first to favor a path that collides with the fewest paths of agents $a_{k'}$ that are not comparable with agent a_j with respect to $\boldsymbol{\prec}_N$ and second to favor a path that collides with the fewest paths of lower priority agents $a_{k''}$ (i.e., $j \prec_N k''$) to avoid having to replan a path for such agents later as much as possible.

Properties. PBS introduces a new ordered pair to the priority ordering of the child PT node whenever it splits a parent PT node. Therefore, the number of splits in any branch of the PT is $O(M^2)$ (the number of all possible ordered pairs). The depth of the PT is also $O(M^2)$ since PBS splits a PT node whenever it expands the PT node. In practice, PBS expands many fewer PT nodes for each MAPF instance where it finds a solution in our experiments.

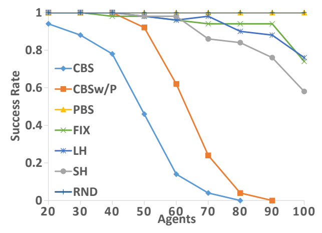
Other Versions of PBS. A non-empty initial priority ordering $\boldsymbol{\prec}_0$ [Line 1] can be given so that the solution found by PBS must respect $\boldsymbol{\prec}_0$. However, there might not exist any solutions that are consistent with $\boldsymbol{\prec}_0$. PBS might thus terminate in the root PT node [Line 6]. For example, for the MAPF instance shown in Figure 2, PBS with the initial priority ordering $\boldsymbol{\prec}_0 = \{2 \prec 1\}$ does not find any solution but standard PBS (the empty initial priority ordering) constructs the priority ordering $\boldsymbol{\prec} = \{1 \prec 2\}$ and finds a solution. A standard prioritized MAPF algorithm is a special case of PBS where the initial priority ordering is a total order.

Experiments

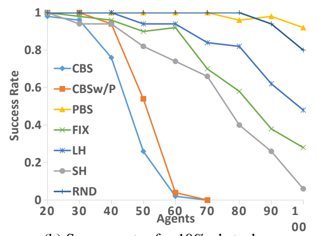
We compare CBSw/P and PBS to CBS and several PBS variants that simulate standard prioritized MAPF algorithms with different total priority orderings. We consider the following algorithms: CBS-H (labeled **CBS**) is a state-of-the-art implementation of CBS (Felner et al. 2018); **FIX** is a PBS variant with a total priority ordering specified by the order of the agents in the MAPF instances, which simulates CA* (Silver 2005); **LH** is a PBS variant with a fixed total priority ordering where the priority of an agent is higher the longer its individually optimal path from its start vertex to its target is, which simulates the heuristic used in in (Van Den Berg and Overmars 2005); **SH** a PBS variant with a fixed total priority ordering where the priority of an agent is higher the shorter its individually optimal path from its start vertex to its target vertex is, which is the opposite of the heuristic of LH; and **RND** is a PBS variant that runs PBS ten times with different randomly generated total priority orderings and picks the solution with the smallest cost, which simulates the randomized strategy in (Bennewitz, Burgard, and Thrun 2002). All experiments were run on a 2.50 GHz Intel Core i5-2450M laptop with 6 GB RAM and a runtime limit of one minute for each MAPF instance for each algorithm except that RND was given a runtime limit of one minute for each of its ten runs. We repeated each experiment 50 times for each number of agents using different randomly generated start and target vertices for the agents, and report the mean values.

Experiment 1: 20 $\times$ 20 grids. We use MAPF instances on 20 $\times$ 20 four-neighbor grids with 0 and 10 percent obstacles (represented by randomly blocked cells) with random start and target vertices, in an empty grid (0% obstacles) and a grid with 10% of the cells filled with obstacles.

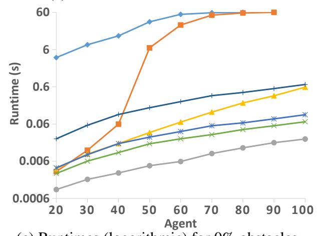
Figures 7(a) and (b) show the success rates of the algorithms, i.e., the percentage of MAPF instances solved within the time limit (RND solves a MAPF instance if it solves the MAPF instance in any of its ten runs within the time limit), as a function of the number of agents. The success rate of CBSw/P is higher than the one of CBS, although they both approach zero as the number of agents increases, due to the increasing size of the CT. PBS and its variants have much higher success rates. FIX, LH, and SH often fail to find a solution but PBS and RND find solutions for all MAPF instances except some "hard" ones with more than 70 agents and 10% obstacles. For the "hard" instances, PBS never reports "No Solution" but reaches the runtime limit on some of them, while RND never reaches the runtime limit for all ten runs but reports "No Solution" for all ten runs on some of them. SH has the lowest success rates among all PBS variants with a total priority ordering because, in SH, the higher priority agents tend to arrive at their target vertices earlier and are thus more likely to block the lower priority agents from reaching their target vertices. Figures 7(c) and (d) show the runtimes of the algorithms. The one minute runtime is averaged in for MAPF instances where the runtime limit was reached. CBSw/P is faster than CBS. RND is slower than the other PBS variants because it solves one MAPF instances multiple times. PBS is almost as fast as the other PBS variants with total priority orderings (within an order of magnitude always) but the runtime difference increases as the number of agents grows.



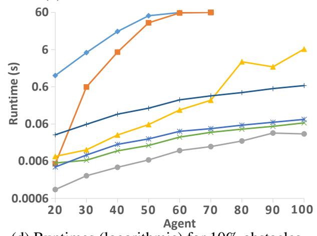
(a) Success rates for 0% obstacles.



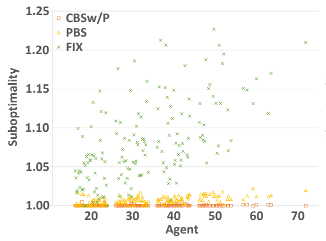
(b) Success rates for 10% obstacles.



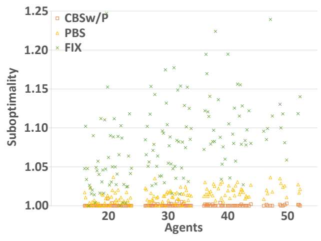
(c) Runtimes (logarithmic) for 0% obstacles.



(d) Runtimes (logarithmic) for 10% obstacles.



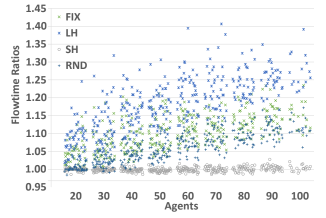
(e) Suboptimality ratios for 0% obstacles.



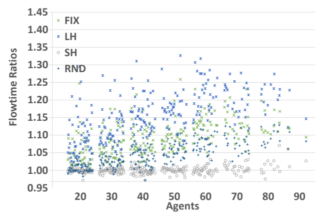
(f) Suboptimality ratios for 10% obstacles.

obs	M	sol	low-level node expansions ($\times$1000)				high-level node expansions			flowtime			
			CBS	CBS w/P	PBS	FIX	CBS	CBS w/P	PBS	CBS	CBS w/P	PBS	FIX
0%	20	47	22.02	0.68	1.04	0.86	268.45	7.74	3.43	252.83	252.89	253.68	265.15
	30	44	369.88	2.69	2.38	1.92	4,319.11	40.98	7.68	399.14	399.20	401.52	428.32
	40	38	353.21	13.50	3.79	2.95	5,430.03	161.68	12.97	519.32	519.53	523.42	566.55
	50	23	778.13	99.97	6.65	5.23	10,813.09	1,167.74	23.74	661.57	661.70	669.00	744.57
	60	6	3,428.58	362.16	9.62	6.50	60,338.67	5,805.00	36.50	741.00	741.83	750.83	850.33
	70	1	13,337.57	3,627.07	19.46	12.04	184,554.00	58,408.00	46.00	854.00	854.00	871.00	1,033.00
10%	20	49	1.30	0.86	1.37	1.02	16.16	9.14	4.69	271.02	271.08	272.90	285.57
	30	47	674.99	51.26	3.25	2.16	5,219.55	454.15	11.38	404.40	404.60	408.81	437.47
	40	36	1,721.40	318.47	5.96	3.22	21,688.75	4,981.56	24.61	539.17	539.39	548.89	595.42
	50	13	5,197.59	680.32	8.37	4.86	63,240.23	9,742.54	37.23	675.92	676.46	691.00	762.77

(g) Results on instances solved by all CBS, CBSw/P, PBS, and FIX.



(h) Flowtime ratios against PBS for 0% obstacles.



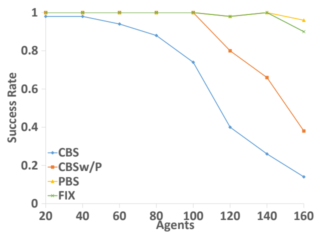
(i) Flowtime ratios against PBS for 10% obstacles.

obs	M	sol	low-level node expansions ($\times$1000)					flowtime				
			PBS	FIX	LH	SH	RND	PBS	FIX	LH	SH	RND
0%	20	50	1.03	0.88	1.29	0.32	8.35	255.84	267.30	277.50	255.56	257.38
	30	50	2.34	1.87	2.99	0.54	18.10	401.26	427.08	447.60	400.58	411.34
	40	49	3.82	3.08	5.44	0.76	32.81	526.96	571.41	612.67	525.84	548.88
	50	48	6.91	4.85	7.84	1.08	45.99	675.27	743.94	792.96	673.04	707.67
	60	46	11.46	6.40	10.87	1.46	63.84	798.33	891.26	965.67	797.78	855.35
	70	41	17.62	7.57	13.99	2.04	85.94	955.41	1,064.02	1,172.61	954.83	1,037.95
	80	37	26.85	9.67	15.17	2.69	94.86	1,076.65	1,222.16	1,330.78	1,077.24	1,177.43
	90	33	41.66	12.62	19.12	3.80	121.57	1,258.42	1,444.24	1,571.06	1,263.67	1,394.91
	100	17	59.28	14.63	24.90	5.08	156.56	1,420.18	1,629.65	1,793.47	1,425.12	1,584.65
10%	20	50	1.37	1.00	1.67	0.38	10.46	273.84	286.26	299.90	273.44	275.66
	30	46	3.16	2.11	3.04	0.67	19.82	410.26	438.65	458.26	409.57	421.26
	40	47	6.67	3.35	5.47	1.09	34.56	552.26	598.30	638.91	550.74	573.49
	50	38	11.24	4.60	7.06	1.64	45.62	703.79	769.53	822.53	703.76	741.26
	60	34	23.19	7.75	11.29	2.72	73.93	843.44	951.12	1,022.79	843.41	905.21
	70	24	42.07	10.16	13.31	4.29	92.90	1,012.79	1,148.13	1,219.33	1,019.88	1,094.63
	80	12	60.07	11.95	16.59	6.22	111.44	1,188.83	1,349.50	1,448.42	1,201.17	1,301.75
	90	3	97.01	14.48	16.20	11.12	128.01	1,406.00	1,562.33	1,638.00	1,451.67	1,524.00

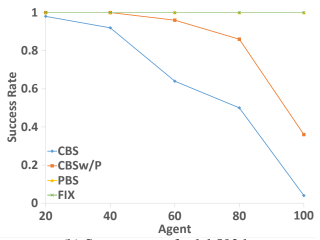
(j) Results on instances solved by all PBS, FIX, LH, SH, and RND.

Figure 7: Results on 20$\times$20 grids.

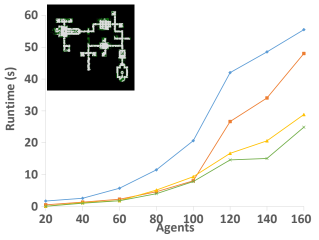
We then consider results for all MAPF instances that CBS, CBSw/P, PBS, and FIX solve. Figures 7(e) and (f) show the ratio of flowtime to the optimal flowtime (computed by CBS). We see that CBSw/P almost always finds optimal



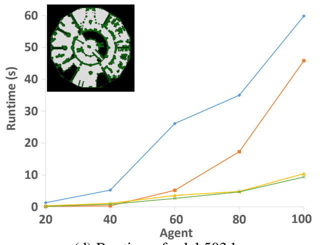
(a) Success rates for brc202d.



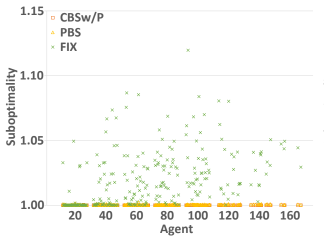
(b) Success rates for lak503d.



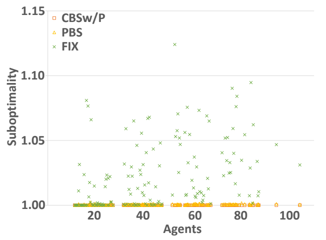
(c) Runtimes for brc202d.



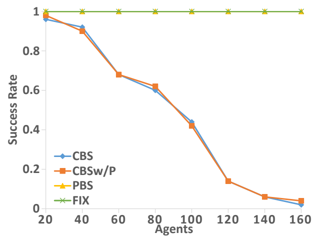
(d) Runtimes for lak503d.



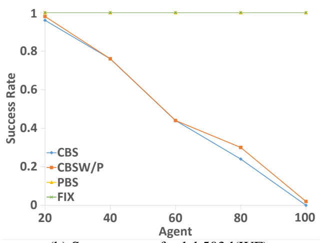
(e) Suboptimality ratios for brc202d.



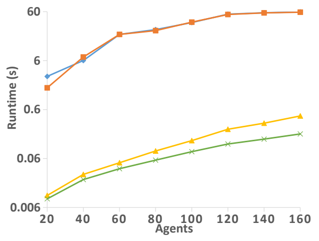
(f) Suboptimality ratios for lak503d.



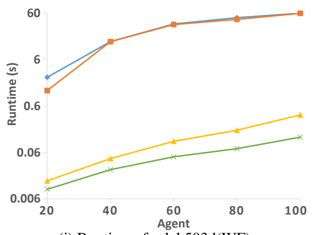
(g) Success rates for brc202d(WF).



(h) Success rates for lak503d(WF).



(i) Runtimes for brc202d(WF).



(j) Runtimes for lak503d(WF).

	M	sol	low-level node expansions (×1000)				high-level node expansions			flowtime			
			CBS	CBS w/P	PBS	FIX	CBS	CBS w/P	PBS	CBS	CBS w/P	PBS	FIX
brc202d	20	49	3.33	3.18	15.64	15.24	0.55	0.55	0.45	2,529.16	2,529.18	2,529.49	2,538.55
	40	49	9.36	8.22	245.53	233.20	4.49	3.18	2.10	5,170.65	5,170.71	5,171.96	5,243.49
	60	47	16.13	15.37	341.22	381.73	6.60	6.64	4.04	7,517.43	7,517.45	7,519.79	7,647.45
	80	44	56.35	28.02	778.00	706.31	27.93	15.73	6.70	9,861.70	9,861.75	9,864.59	10,090.64
	100	37	87.41	59.50	1,294.59	1,326.70	39.70	30.92	10.03	12,344.73	12,344.78	12,349.43	12,740.68
	120	20	195.05	86.61	1,434.54	1,708.10	112.15	56.95	14.10	15,383.25	15,383.30	15,390.55	15,830.15
	140	13	388.29	159.20	1,402.82	1,302.98	114.77	61.69	18.77	16,709.92	16,710.00	16,721.38	17,194.23
	160	5	486.98	120.13	3,025.44	2,469.80	228.20	99.80	24.40	19,982.60	19,982.80	19,991.80	20,841.40
lak503d	20	49	8.09	7.01	87.06	38.49	2.96	3.69	1.47	2,320.71	2,320.73	2,321.27	2,341.14
	40	46	49.12	22.04	232.75	197.86	22.26	11.24	4.20	4,702.54	4,702.59	4,704.35	4,785.83
	60	32	639.95	115.09	536.72	516.34	325.91	70.03	10.03	6,809.72	6,809.81	6,813.78	7,030.59
	80	25	783.31	360.23	844.29	905.59	341.28	251.20	15.00	9,121.60	9,121.76	9,127.88	9,485.76
	100	2	4,129.21	1,190.30	1,497.35	1,708.58	1,321.50	810.50	22.50	12,781.50	12,782.50	12,792.00	13,277.00
brc202d (WF)	20	48	3.31	3.16	3.07	2.74	0.50	0.50	0.31	2,528.23	2,528.25	2,528.60	2,528.60
	40	45	9.31	7.98	7.12	5.93	4.56	3.02	1.22	5,179.53	5,179.56	5,180.82	5,180.82
	60	34	14.29	13.19	10.67	8.85	5.88	5.71	2.06	7,484.32	7,484.32	7,486.35	7,486.65
	80	30	20.80	17.49	14.21	11.49	9.80	8.87	3.17	9,890.00	9,890.00	9,893.03	9,893.30
	100	21	45.64	31.13	20.63	15.48	23.52	17.19	5.10	12,246.86	12,246.86	12,251.90	12,252.24
	120	7	133.41	56.11	27.12	19.05	76.14	35.29	7.57	15,079.71	15,079.71	15,087.29	15,087.43
	140	3	105.78	68.06	31.95	22.61	57.00	37.00	9.67	17,215.67	17,215.67	17,224.33	17,225.33
lak503d (WF)	20	48	7.15	6.37	4.45	3.11	2.54	3.42	1.00	2,324.54	2,324.56	2,325.06	2,325.21
	40	37	33.14	17.09	10.13	6.71	15.03	8.41	2.59	4,648.08	4,648.08	4,649.68	4,650.27
	60	20	321.00	89.70	19.90	11.44	117.05	41.55	6.10	6,744.40	6,744.40	6,748.50	6,749.30
	80	12	580.91	167.59	28.39	14.83	258.00	84.83	9.08	9,033.58	9,033.58	9,040.58	9,041.17

(k) Results on instances solved by all CBS, CBSw/P, PBS, and FIX.

Figure 8: Results on game maps.

solutions. PBS finds solutions that are very close to optimal, getting slightly worse as the number of agents grows, but never more than 4% worse than optimal. In contrast, FIX finds solutions that are more than 5% worse than optimal and often much worse. The results in the graphs are confirmed by the results reported in the table shown in Figure 7(g) where the number of instances (**sol**) that we consider for each number of agents is also reported.

We also consider results for the set of instances that are solved by PBS, FIX, LH, SH, and RND simultaneously, so we can easily compare their behaviour. Figures 7(h) and (i) show that PBS is almost indistinguishable from SH, which is better than RND, which is in turn better than FIX, which is in turn better than LH. Figure 7(j) shows that LH has the largest numbers of low-level node expansions among algorithms with total priority orderings. PBS has large numbers of low-level node expansions because it plans paths for more priority orderings and thus performs more low-level searches. RND has the largest numbers of low-level node expansions because it solves each MAPF instance multiple times. The runtime in Figures 7(c) and (d) behaves like the number of low-level node expansions since the algorithms spend most of their runtime in low-level searches.

Experiment 2: Game Maps. We also use MAPF instances on two standard benchmark maps **brc202d** (a 481 × 530 four-neighbor grid) and **lak503d** (a 192 × 192 four-neighbor grid) of the game Dragon Age: Origins (Sturtevant 2012). We also use MAPF instances on these grids instances labeled **brc202d(WF)** and **lak503d(WF)** where the agents are removed from the grid once they arrive at their target vertices for the first time so that they cannot block other agents afterward.

We first focus on results for brc202d and lak503d. CBSw/P outperforms CBS in terms of success rate (Figures 8(a) and (b)) and running time (Figures 8(c) and (d)), again with almost negligible reduction in solution quality (Figures 8(e) and (f)). PBS outperforms FIX, CBS, and CBSw/P in terms of success rate, and is close to FIX in terms of runtime. PBS is almost always able to find nearly optimal solutions as opposed to FIX (Figures 8(e) and (f)). Figures 8(k) shows results for all MAPF instances that all algorithms solve within the runtime limit. PBS and FIX expand more low-level nodes than CBS and CBSw/P for **brc202d** because the narrow corridors make the constraints of high priority agents more significant than for other maps, especially when those agents arrive at their target vertices early and stop moving, which explains why PBS and FIX reach the runtime limit for some instances for 120 and 160 agents on **brc202d** within the time limit.

We then focus on results for brc202d(WF) and lak503d(WF). These MAPF instances are much easier to solve for PBS and FIX than those on brc202d and lak503d, as the numbers of low-level node expansions show. CBSw/P and CBS are almost indistinguishable with respect to both success rates and runtimes (Figures 8(g) to (j)), which is not the case for brc202d and lak503d.

We further tested the scalability of PBS on brc202d(WF) with large numbers of agents. PBS scales to 600 agents and never reaches the runtime limit (shown in the below table).

M	100	200	300	400	500	600
runtime (s)	0.14	0.86	3.07	8.79	18.51	35.18
PT node expansions	5.16	21.38	46.02	81.84	130.04	182.74

Conclusions

We tackled the main challenge of exploring "good" orderings of agents for prioritized planning from a conceptual and practical perspective. From the conceptual side, we developed a first theoretical framework for discussing the limits of prioritized planning. From the practical side, we developed two new algorithms, CBSw/P and PBS, that search for "good" orderings and thus compute solutions with "good" orderings. Both algorithms order agents lazily by imposing an ordering on two agents only to resolve collisions between them and both of them explore the orderings in a systematic fashion: CBSw/P using best-first search, and PBS using depth-first search.

References

Azarm, K., and Schmidt, G. 1997. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *ICRA*, 3526–3533.

Bennewitz, M.; Burgard, W.; and Thrun, S. 2002. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems* 41(2-3):89–99.

Buckley, S. J. 1989. Fast motion planning for multiple moving robots. In *ICRA*, 322–326.

Cáp, M.; Vokrínek, J.; and Kleiner, A. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *ICAPS*, 324–332.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, 290–296.

Erdmann, M. A., and Lozano-Pérez, T. 1987. On multiple moving objects. *Algorithmica* 2:477–521.

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 *SoCS*, 29–37.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 83–87.

Ferrari, C.; Pagello, E.; Ota, J.; and Arai, T. 1998. Multirobot motion coordination in space and time. *Robotics and Autonomous Systems* 25(3-4):219–229.

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

Luna, R., and Bekris, K. E. 2011. Push and Swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, 294–300.

Ma, H., and Koenig, S. 2017. AI buzzwords explained: Multi-agent path finding (MAPF). *AI Matters* 3(3):15–19.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016a. Overview: Generalizations of multi-agent path finding to real-world scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

Ma, H.; Tovey, C.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016b. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI*, 3166–3173.

Ma, H.; Hönig, W.; Cohen, L.; Uras, T.; Xu, H.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2017a. Overview: A hierarchical framework for plan generation and execution in multi-robot systems. *IEEE Intelligent Systems* 32(6):6–12.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017b. Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS*, 837–845.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017c. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, 270–272.

Morris, R.; Pasareanu, C.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

O'Donnell, P. A., and Lozano-Pérez, T. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *ICRA*, 484–489.

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.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

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

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, 668–673.

Sturtevant, N., and Buro, M. 2006. Improving collaborative pathfinding using map abstraction. In *AIIDE*, 80–85.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Surynek, P. 2015. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*, 1916–1922.

Van Den Berg, J., and Overmars, M. 2005. Prioritized motion planning for multiple robots. In *IROS*, 430–435.

Velagapudi, P.; Sycara, K.; and Scerri, P. 2010. Decentralized prioritized planning in large multirobot teams. In *IROS*, 4603–4609.

Veloso, M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 4423–4429.

Wagner, G., and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219:1–24.

Wang, K., and Botea, A. 2011. MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42:55–90.

Warren, C. W. 1990. Multiple robot path coordination using artificial potential fields. In *ICRA*, 500–505.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–20.

Yu, J., and LaValle, S. M. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 1444–1449.