Multi-Agent Path Finding for Precedence-Constrained Goal Sequences

Han Zhang
University of Southern California
Los Angeles, CA, USA
zhan645@usc.edu

Jingkai Chen
Massachusetts Institute of Technology
Cambridge, MA, USA
jkchen@csail.mit.edu

Jiaoyang Li
University of Southern California
Los Angeles, CA, USA
jiaoyanl@usc.edu

Brian C. Williams
Massachusetts Institute of Technology
Cambridge, MA, USA
williams@csail.mit.edu

Sven Koenig
University of Southern California
Los Angeles, CA, USA
skoenig@usc.edu

ABSTRACT

With the rising demand for deploying robot teams in autonomous warehouses and factories, the Multi-Agent Path Finding (MAPF) problem has drawn more and more attention. The classical MAPF problem and most of its variants focus on navigating agent teams to goal locations while avoiding collisions. However, they do not take into account any precedence constraints that agents should respect when reaching their goal locations. Planning with precedence constraints is important for real-world multi-agent systems. For example, a mobile robot can only pick up a package at a station after it has been delivered by another robot. In this paper, we study the Multi-Agent Path Finding with Precedence Constraints (MAPF-PC) problem, in which agents need to visit sequences of goal locations while satisfying precedence constraints between the goal locations. We propose two algorithms for solving this problem systematically: Conflict-Based Search with Precedence Constraints (CBS-PC) is complete and optimal, and Priority-Based Search with Precedence Constraints (PBS-PC) is incomplete but more efficient in finding near-optimal solutions in practice. Our experimental results show that CBS-PC scales to dozens of agents and hundreds of goal locations and precedence constraints, and PBS-PC scales to hundreds of agents, around one thousand goal locations, and hundreds of precedence constraints.

KEYWORDS

Multi-Agent Path Finding; Precedence Constraints

ACM Reference Format:
Han Zhang, Jingkai Chen, Jiaoyang Li, Brian C. Williams, and Sven Koenig. 2022. Multi-Agent Path Finding for Precedence-Constrained Goal Sequences. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

In intelligent warehouse and factory systems, large teams of robots are expected to complete constantly dispatched tasks effectively. One typical example is the Kiva (now: Amazon Robotics) warehouse system, in which hundreds of Kiva robots are coordinated to transport movable shelving units on the fly without human intervention [17]. The Multi-Agent Path Finding (MAPF) problem is the problem of navigating a team of agents from their start locations to their goal locations while avoiding collisions. Due to the rising demand for developing such multi-robot systems, MAPF has drawn more and more attention, and MAPF algorithms are regarded as fundamental techniques for coordinating the motions of robot teams.

Although classical MAPF algorithms can find effective plans for navigating mobile robots in autonomous warehouses, they only plan for agents to reach single goal locations. In real-world systems, we often need to coordinate robots that fulfill streams of tasks with precedence constraints over relatively long time horizons. For example, a mobile robot needs to move to several stations to deliver different packages, and another mobile robot can only pick up a package after it has been delivered to the corresponding station.

However, existing MAPF algorithms do not consider precedence constraints between goals when planning the path to reach a sequence of goal locations for each agent [5]. This motivates us to study the Multi-Agent Path Finding with Precedence Constraints (MAPF-PC) problem, in which agents need to complete sequences of goals (by reaching the goal locations) while satisfying precedence constraints between the goals. We present two algorithms for solving MAPF-PC: Conflict-Based Search with Precedence Constraints (CBS-PC) and Priority-Based Search with Precedence Constraints (PBS-PC), which generalize the state-of-the-art MAPF algorithms CBS [14] and PBS [10], respectively. We also propose several improvements to CBS-PC. Like CBS and PBS, CBS-PC is complete and optimal, and PBS-PC is incomplete but more efficient in obtaining near-optimal plans in practice.

We benchmarked CBS-PC and PBS-PC on MAPF-PC instances with different numbers of agents, goals, and precedence constraints. The results show that the most advanced CBS-PC variant scales to dozens of agents and hundreds of goal locations and precedence constraints, and PBS-PC scales to hundreds of agents, around one thousand goal locations, and hundreds of precedence constraints.

2 PRELIMINARIES

In this section, we introduce MAPF, CBS, prioritized planning, and PBS to provide the necessary background for the MAPF-PC problem and our MAPF-PC algorithms.

Han Zhang and Jingkai Chen contributed equally to this work.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

2.1 MAPF

The MAPF problem is defined by an undirected graph $G = (V, E)$ and a set of m agents $\{a_1 \dots a_m\}$. Each agent a_i has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. In each timestep, an agent either moves to a neighboring vertex, waits at its current vertex, or terminates at its goal vertex (that is, does not move anymore). Both move and wait actions have unit cost, and terminate actions have zero cost. A *path* of an agent is a sequence of actions that leads it from its start vertex to its goal vertex and ends with a terminate action. The *path cost* of a path is the accumulated cost of all actions in this path. A *vertex conflict* happens when two agents stay at the same vertex simultaneously, and an *edge conflict* happens when two agents traverse the same edge in opposite directions simultaneously. A *solution* is a set of conflict-free paths of all agents. A solution is an *optimal solution* iff there is no other solution with a smaller objective value. Two common objectives for MAPF are the Sum of path Costs (SoC) and the makespan. The *SoC* is the sum of the path costs of the paths of all agents, and the *makespan* is the maximum path cost of the paths of all agents. Solving MAPF optimally is NP-hard for either objective [12, 18].

2.2 CBS

CBS [14] is a complete and optimal two-level MAPF algorithm. On the high level, CBS performs a best-first search on a *Constraint Tree* (CT). Each CT node contains (1) a set of constraints[1] and (2) a set of paths, one for each agent, that satisfy all these constraints. The cost of a CT node is the SoC or makespan of all its paths, depending on the objective of the MAPF problem. CBS starts with the root CT node, which has an empty set of constraints and a path for each agent that has the minimum path cost when ignoring conflicts. When expanding a CT node, CBS returns the paths of it as a solution if the paths are conflict-free. Otherwise, CBS picks a conflict to resolve, splits the CT node into two child CT nodes, and adds a constraint to each child CT node to prohibit either one or the other of the two conflicting agents from using the conflicting vertex or edge at the conflicting timestep. CBS then calls its low level to replan the path of the newly constrained agent in each child CT node. On the low level, for a given CT node and a given agent, CBS finds a path for the agent that has the minimum path cost while satisfying all constraints of the CT node but ignoring conflicts.

2.3 Prioritized Planning and PBS

Prioritized planning is a simple-yet-effective MAPF algorithm that plans the agents according to a predefined total priority ordering. A *priority ordering* $\prec$ is a strict partial order on $\{a_1 \dots a_m\}$ where $a_i \prec a_j$ indicates that agent a_i is of higher priority than agent a_j. A *total* priority ordering $\prec$ satisfies that, for any two agents a_i and a_j, we have either $a_i \prec a_j$ or $a_j \prec a_i$. Prioritized planning plans for agents in the order from highest priority to lowest priority. For each agent, it finds a path that has the minimum path cost among all paths that avoid conflicts with the paths of all higher-priority agents. Whether prioritized planning finds a solution often depends on the predefined priority ordering, and it is not always easy to find a priority ordering that works.

PBS [10] is a two-level MAPF algorithm which systematically searches for such a priority ordering. On the high level, it performs a depth-first search on a *Priority Tree* (PT). Each PT node N contains (1) a *priority ordering* $\prec_N$ and (2) a set of paths, one for each agent, that *respects* its priority ordering, i.e., the paths of any two agents a_i and a_j with $a_i \prec_N a_j$ are conflict-free. PBS starts with the root PT node, which has an empty priority ordering (that is, no agent is of higher priority than another) and thus a path for each agent that has the minimum path cost when ignoring conflicts. When expanding a PT node, PBS picks a pair of conflicting agents a_i and a_j and splits the PT node into two child PT nodes, each extending the priority ordering of its parent PT node with either $a_i \prec a_j$ or $a_j \prec a_i$. PBS then calls its low level to replan the paths of the child nodes so that their paths respect their priority orderings. On the low level, for each PT node, PBS uses topological sorting according to the priority ordering to order the agents and plans paths for them in that order. Like prioritized planning, PBS plans a path for each agent that has the minimum path cost among all paths that avoid conflicts with the paths of higher-priority agents. A PT node is pruned if PBS cannot find such a path for at least one agent. When generating a PT node, PBS returns the paths of it as a solution if the paths are conflict-free. PBS is neither optimal nor complete, but existing work shows that it often finds solutions that are close to optimal and scales well to large numbers of agents [9, 10].

[1]The constraints in a CT are added by CBS to solve the MAPF instance. They are different from precedence constraints, which characterize a MAPF-PC instance and are thus part of the input.

3 PROBLEM DEFINITION

The MAPF-PC problem is defined by an undirected graph $G = (V, E)$, a set of m agents $\{a_1 \dots a_m\}$, and a set of precedence constraints $\mathcal{T}$. Each agent a_i has a start vertex $s_i \in V$ and a sequence of l_i *goals* $[g_i^1, g_i^2 \dots g_i^{l_i}]$. Each goal g_i^j corresponds to a *goal vertex* $g_i^j.loc \in V$. When agent a_i is at $g_i^j.loc$, it can (but is not required to) *complete* goal g_i^j. Complete actions take zero timesteps and have zero cost. We use $\tau(g_i^j)$ to denote the completion timestep of g_i^j. Each *precedence constraint* $\langle g_i^j, g_{i'}^{j'} \rangle \in \mathcal{T}$ is a tuple of two goals g_i^j and $g_{i'}^{j'}$ and means that g_i^j must be completed before $g_{i'}^{j'}$. An agent must complete its goals in the order of the goal sequence and terminates when it completes its last goal. The completion timesteps of all goals must satisfy the precedence constraints as well. Besides vertex and edge conflicts, we consider a new type of conflict called *precedence conflict*. A precedence conflict happens when there exists a pair of goals g_i^j and $g_{i'}^{j'}$ such that $\tau(g_i^j) \geq \tau(g_{i'}^{j'})$ and $\langle g_i^j, g_{i'}^{j'} \rangle \in \mathcal{T}$. In MAPF-PC, a *path* for an agent also needs to specify the completion timestep of each goal of the agent. A *path segment* for goal g_i^j is a sequence of actions from the completion of g_i^{j-1} (or timestep 0 if $j = 1$) to the completion of g_i^j. A *solution* to a MAPF-PC instance is a set of conflict-free paths for all agents.

The MAPF problem is a sub-class of the MAPF-PC problem where each agent has only one goal and $\mathcal{T} = \emptyset$. Therefore, solving MAPF-PC optimally is also NP-hard.

4 CBS WITH PRECEDENCE CONSTRAINTS

We introduce CBS-PC, a complete and optimal algorithm that solves the MAPF-PC problem. In this paper, we are interested in minimizing the SoC. However, CBS-PC can be adapted to other objectives, such as minimizing the makespan or the sum of goal completion timesteps, by making small modifications to its low level.

4.1 High Level of CBS-PC

On the high level, CBS-PC resolves vertex and edge conflicts in the same way as CBS. Consider the case when CBS-PC picks a precedence conflict between goals g_i^j an $g_{i'}^{j'}$ that violates the precedence constraint $\langle g_i^j, g_{i'}^{j'} \rangle$ (in other words, g_i^j needs to be completed before $g_{i'}^{j'}$, but this is not satisfied by the paths of the CT node). We use t to denote $\tau(g_i^j)$ as specified by the path of agent a_i. CBS-PC splits the CT node into two child CT nodes and resolves the precedence conflict by adding one of the following *completion timestep constraints* to one child CT node and the other one to the other child CT node:

(1) $\tau(g_{i'}^{j'}) > t$: agent $a_{i'}$ must complete $g_{i'}^{j'}$ after timestep t. In the child CT node, the path of $a_{i'}$ is replanned, and $g_{i'}^{j'}$ is thus completed after g_i^j.

(2) $\tau(g_{i'}^{j'}) \leq t$: agent $a_{i'}$ must complete $g_{i'}^{j'}$ no later than timestep t, which is already satisfied by the path of $a_{i'}$. However, due to precedence constraint $\langle g_i^j, g_{i'}^{j'} \rangle$, we have $\tau(g_i^j) \leq t - 1$, which is not satisfied by the path of a_i. In the child CT node, both constraints $\tau(g_{i'}^{j'}) \leq t$ and $\tau(g_i^j) \leq t - 1$ are added, the path of a_i is replanned, and $g_{i'}^{j'}$ is thus completed at least one timestep earlier than before.

When generating a child CT node with the completion timestep constraint in (1), the precedence conflict is immediately resolved since $a_{i'}$ is forced to complete $g_{i'}^{j'}$ after a_i completes g_i^j. When generating a child CT node with the completion timestep constraints in (2), CBS-PC tries to find a path for a_i that completes g_i^j earlier than t, which is the timestep when g_i^j is completed in the parent CT node. Such a path often does not exist as t is often the earliest timestep when a_i can complete g_i^j, in which case CBS-PC prunes the child CT node. However, if such a path does exist, it is possible that the new path of agent a_i still does not complete g_i^j earlier than $\tau(g_{i'}^{j'})$, in which case the two agents still have the precedence conflict. Nevertheless, $\tau(g_i^j)$ is guaranteed to decrease by at least one timestep. So, if CBS-PC continues to try to resolve the precedence conflict between the two agents, it will eventually either prune the branch that involves the repeatedly occurring precedence conflict or generate a child CT node where it is resolved.

One needs to decide which conflict to choose if the CT node to be expanded contains multiple conflicts. Existing work shows that choosing conflicts that increase the path cost in the child CT nodes can improve the efficiency of CBS [1]. CBS-PC follows this principle and prefers precedence conflicts over vertex or edge conflicts (and breaks ties randomly) because the completion timestep constraint (1) often increases the path cost. We have also tried the conflict prioritization method in [1], but, unfortunately, this method turned out to be too computationally expensive for MAPF-PC instances since it needs to find all paths with the minimum path costs (known as MDD [15]) that complete all goals of each agent involved in each conflict.

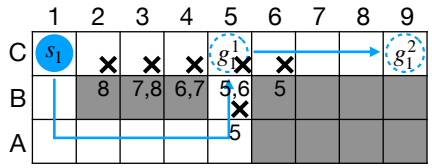


Figure 1: An example of low-level planning in CBS-PC. Agent a_1 has two goals g_1^1 and g_1^2. Crosses represent vertex constraints on a_1, and the numbers below them are the timesteps that a_1 is not allowed to stay at the vertices. For example, a_1 is not allowed to stay at $C5$ at timesteps 5 and 6.

4.2 Low Level of CBS-PC

On the low level, we need to plan a path for an agent that (1) completes all its goals in order, (2) satisfies the constraints imposed by the high level, and (3) minimizes the path cost. One might consider planning the path segments for all goals sequentially instead of planning the entire path at once. The following example shows that planning sequentially can result in a sub-optimal path.

Example 1. *Figure 1 shows an example where agent a_1 has multiple vertex constraints, represented as crosses on their vertices and numbers that specify their timesteps. If we plan path segments sequentially from one goal to the next, we first find a path segment to $C5$ at timestep 4 and then plan the path segment to $C9$. Because of the vertex constraints on $B5$, $C5$, and $C6$ at timestep 5, a_1 can move only to $C4$ at timestep 5. Then, at the next timestep, because of the vertex constraints on $C4$ and $C5$ at timestep 6, a_1 can move only to $C3$, and so on. Eventually, a_1 moves back to $C1$ at timestep 8 and thus can reach $C9$ at timestep 16 the earliest. However, if we plan the entire path of a_1 at once, a_1 can reach $C9$ already at timestep 12 when following the blue arrows.*

CBS-PC uses the Multi-Label A* (MLA*) algorithm [5, 9] to find a minimum-cost path that satisfies all constraints of the CT node. We extend MLA* to support completion timestep constraints: (1) An agent can complete a goal only at a timestep that is larger than the lower bound on the completion timestep of the goal, if provided, and (2) MLA* prunes any low-level search nodes in which the agent can reach a goal vertex only after the upper bound on the completion timestep of the goal, if provided.

4.3 Theoretical Analysis

CBS-PC differs from CBS in how the low level plans paths and how it addresses precedence conflicts. MLA* is complete and optimal [5]. Resolving precedence conflicts with completion timestep constraints does not rule out any solution of a MAPF-PC instance and, for every cost c, there is only a finite number of CT nodes with cost c in CBS-PC. With a proof similar to the one for CBS, we can therefore show that CBS-PC is complete and optimal.

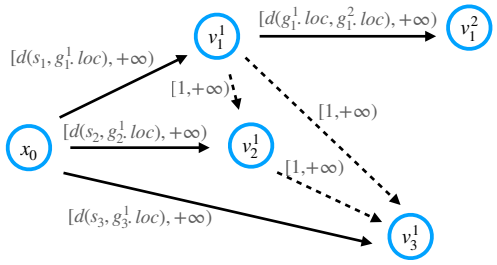


Figure 2: The STN for the root CT node of the MAPF-PC instance in Example 2.

4.4 Improvements

We now present three techniques for improving the efficiency of CBS-PC. One of them is a specialized technique for MAPF-PC, and the other two are adopted from existing work on improving CBS.

Constraint propagation: CBS-PC only adds completion timestep constraints when it picks a precedence conflict to split on. However, additional completion timestep constraints can be inferred from the existing ones in a way similar to how $\tau(g_i^j) \leq t - 1$ is inferred in Completion Timestep Constraint (2).

When generating a CT node, CBS-PC builds a *Simple Temporal Network* (STN) for the CT node [3]. An STN is a directed acyclic graph $\langle V, \mathcal{TC} \rangle$. Each vertex $v \in V$ represents a time point, called an event, and $\tau(v)$ represents the occurrence time of v. Each STN has a reference event $x_0 \in V$ that represents the "beginning of time," and $\tau(x_0)$ is conventionally set to 0. Each edge $\langle v, v' \rangle \in \mathcal{TC}$, annotated with an interval $[LB, UB]$, indicates that v must occur between LB and UB time units after v', that is, $\tau(v') - \tau(v) \in [LB, UB]$. To construct the STN for a CT node, for each goal g_i^j, CBS-PC adds a vertex v_i^j to the STN to represent the completion of g_i^j. CBS-PC adds edges to the STN in three cases:

(1) We use $d(x, y)$ to denote the minimum cost needed to move from x to y in graph G while ignoring constraints and conflicts. For each agent a_i, CBS-PC adds edge $\langle x_0, v_i^1 \rangle$ with interval $[d(s_i, g_i^1.loc), +\infty)$ to the STN, and, for each pair of consecutive goals g_i^j and g_i^{j+1}, CBS-PC adds edge $\langle v_i^j, v_i^{j+1} \rangle$ with interval $[d(g_i^j.loc, g_i^{j+1}.loc), +\infty)$ to the STN.

(2) For each precedence constraint $\langle g_i^j, g_{i'}^{j'} \rangle \in \mathcal{T}$, CBS-PC adds edge $\langle v_i^j, v_{i'}^{j'} \rangle$ with interval $[1, +\infty)$ to the STN.

(3) For each completion timestep constraint $\tau(g_i^j) > t$, CBS-PC adds edge $\langle x_0, v_i^j \rangle$ with interval $[t + 1, +\infty)$ to the STN. Similarly, for each completion timestep constraint $\tau(g_i^j) \leq t$, CBS-PC adds edge $\langle x_0, v_i^j \rangle$ with interval $[0, t]$ to the STN.

The lower and upper bounds on the completion timestep of each goal in the STN can be computed using the Bellman-Ford algorithm. Each lower and upper bound can be converted to a completion timestep constraint. CBS-PC adds these constraints to the generated CT node if the CT node does not contain them already.

Example 2. *Consider a MAPF-PC instance with three agents. Agent a_1 has two goals, and agents a_2 and a_3 both have one goal. $\mathcal{T} = \{\langle g_1^1, g_2^1 \rangle, \langle g_1^1, g_3^1 \rangle, \langle g_2^1, g_3^1 \rangle\}$. Figure 2 shows the corresponding STN for the root CT node. The four solid edges are due to Case (1), and the three dashed edges are due to Case (2). We assume that $d(s_1, g_1^1.loc) > d(s_2, g_2^1.loc) > d(s_3, g_3^1.loc)$.*

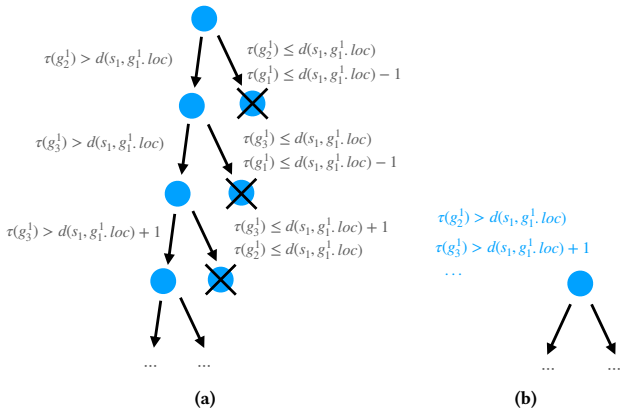


Figure 3: CTs of CBS-PC with and without constraint propagation when solving the MAPF-PC instance in Example 2.

Figure 3a shows the CT of CBS-PC without constraint propagation. The text next to the edges describes the constraints added to the CT nodes. Crossed-out CT nodes are pruned because CBS-PC cannot find paths for some agents. CBS-PC generates multiple CT nodes to resolve the precedence conflicts between g_1^1 and g_2^1, g_1^1 and g_3^1, and g_2^1 and g_3^1, respectively. Figure 3b shows the CT of CBS-PC with constraint propagation. The blue text next to the root CT node describes the constraints generated from constraint propagation, which impose lower bounds on the completion timesteps of the goals. Since the low-level planner is aware of these completion timestep constraints, the paths in the root CT node do not exhibit the previously mentioned precedence conflicts.

Disjoint splitting: Different from the standard splitting rule of CBS, disjoint splitting [8] picks one conflicting agent and then (1) adds a constraint to one child CT node to prohibit this agent from using the conflicting vertex or edge at the conflicting timestep and (2) adds a constraint to the other child CT node to force this agent to use the conflicting vertex or edge at the conflicting timestep, which implies that no other agent can use the conflicting vertex or edge at the conflicting timestep. Since disjoint splitting is able to speed up different variants of CBS significantly, we use it in the context of CBS-PC as well.

Target reasoning: A vertex conflict is a *target conflict* [7] if and only if one of the conflicting agents, denoted as a_i, terminates before the conflicting timestep, denoted as t. It is inefficient for CBS-PC to resolve target conflicts with only vertex and edge constraints. Instead, *target reasoning* [7] uses constraints $\tau(g_i^{l_i}) > t$ (the path of a_i needs to be replanned) and $\tau(g_i^{l_i}) \leq t$ (the path of the other conflicting agent needs to be replanned because only a_i is allowed to occupy $g_i^{l_i}.loc$ at timestep t) to resolve target conflicts. Note that l_i denotes the number of goals of agent a_i.

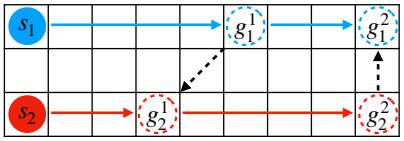


Figure 4: A two-agent MAPF-PC instance. Both agents have two goals. Solid-line arrows represent the sequence of goal vertices that agents need to visit, and dashed-line arrows represent the precedence constraints between goals.

5 PBS WITH PRECEDENCE CONSTRAINTS

We now introduce a suboptimal but more scalable MAPF-PC algorithm, called PBS-PC, which adopts the PBS algorithm for MAPF to solve our MAPF-PC problem. We start with a naïve variant of PBS for MAPF-PC, which assigns priorities to agents to resolve conflicts. We explain why naïve PBS does not find a solution for a simple MAPF-PC instance, which motivates us to introduce an advanced variant, called PBS-PC, which resolves conflicts by assigning priorities to goals.

5.1 Naïve PBS for MAPF-PC

Unlike CBS, PBS can be used to solve the MAPF-PC problem without any change on the high level. On the low level, when PBS plans for an agent, it uses MLA* to find a path that has the minimum cost among all paths that avoid vertex, edge, and precedence conflicts with the paths of all higher-priority agents. However, planning for one agent after another can fail even for a simple MAPF-PC instance.

Example 3. *Consider the two-agent MAPF-PC instance shown in Figure 4, where agents a_1 and a_2 have two goals each and $\mathcal{T} = \{\langle g_1^1, g_2^1\rangle, \langle g_2^2, g_1^2\rangle\}$, that is, the first goal of a_1 must be completed before the first goal of a_2 and the second goal of a_1 must be completed after the second goal of a_2. In the root PT node, which has no priority ordering between a_1 and a_2, PBS plans for each agent individually. We have $\tau(g_1^1) = 5$ and $\tau(g_2^1) = 3$, which is a precedence conflict. PBS splits the root PT node into two child PT nodes. In one child PT node, it extends the priority ordering with $a_1 \prec a_2$, meaning that it plans the path of a_1 first. This path has $\tau(g_1^1) = 5$ and $\tau(g_1^2) = 8$. PBS cannot find a path for a_2 because it is impossible to satisfy $\tau(g_2^1) > 5$ and $\tau(g_2^2) < 8$ simultaneously. In the other child PT node, it extends the priority ordering with $a_2 \prec a_1$, meaning that it plans the path of a_2 first. This path has $\tau(g_2^1) = 3$ and $\tau(g_2^2) = 8$. PBS cannot find a path for a_1 because it is impossible to satisfy $\tau(g_1^1) < 3$. Thus, naïve PBS fails immediately for this simple MAPF-PC instance.*

5.2 PBS-PC

PBS cannot solve the MAPF-PC instance of Example 3 because imposing priority orderings on agents is insufficient for resolving conflicts caused by the precedence constraints among goals. We thus propose PBS-PC, which assigns priority orderings to pairs of goals and plans the path segment for one goal at a time.

Algorithm 1 shows the high level of PBS-PC. We use Γ to denote the list of goals of all agents. Like PBS, PBS-PC performs a depth-first search on the high level and stores all generated but not yet expanded PT nodes in a stack. Unlike PBS, the root PT node of PBS-PC does not always have an empty priority ordering. PBS-PC initializes the priority ordering (Lines 2-5) by (1) adding $g_i^j \prec g_i^{j+1}$ to $\prec_{Root}$ for each pair of consecutive goals of the same agent and (2) adding $g_i^j \prec g_{i'}^{j'}$ to $\prec_{Root}$ for each precedence constraint $\langle g_i^j, g_{i'}^{j'}\rangle \in \mathcal{T}$ between the goals of two different agents. For each node N, $N.paths$ stores the path segment of each goal. PBS-PC begins with an empty path segment for each goal in the root PT node (Line 6). When expanding a PT node, PBS-PC invokes Algorithm 2 to plan path segments (Line 10). Algorithm 2 plans for the goals in a topologically sorted order according to the priority ordering (Line 25) and plans for one goal at a time until:

(1) PBS-PC cannot find a path segment for a goal (Lines 5-6), and the PT node is pruned on Lines 11-12;
(2) PBS-PC finds a conflict among non-empty path segments (Lines 33-35), and the PT node is split into two child PT nodes on Lines 15-23; or
(3) the path segments for all goals are found, and PBS-PC returns a solution for the MAPF-PC instance on Lines 13-14.

Function *FindConflictingGoal*(N, g) returns a goal whose planned path segment has vertex or edge conflicts with the path segment of g or returns *empty* if no such goal exists. When generating a child PT node, PBS-PC extends the priority ordering of the parent PT

Algorithm 1: High-Level Search of PBS-PC

```
1  ≺_Root ← ∅, Root.conflict ← empty
2  ; foreach pair of consecutive goals g_i^j and g_i^{j+1} do
3      ≺_Root ← ≺_Root ∪ {g_i^j ≺ g_i^{j+1}};
4  foreach ⟨g_i^j g_{i'}^{j'}⟩ ∈ T do
5      ≺_Root ← ≺_Root ∪ {g_i^j ≺ g_{i'}^{j'}};
6  Root.paths[g_i^j] ← empty for each goal g_i^j;
7  STACK ← {Root};
8  while STACK is not empty do
9      N ← STACK.pop();
10     succ ← UpdatePath(N);                    // Algorithm 2
11     if succ is false then
12         continue;
13     if N.conflict is empty then
14         return N.paths;
15     (g_i^j, g_{i'}^{j'}) ← N.conflict;
16     foreach g ∈ {g_i^j, g_{i'}^{j'}} do
17         g' ← the goal in {g_i^j, g_{i'}^{j'}} \ {g};
18         N' ← N;
19         ≺_{N'} ← ≺_N ∪ {g ≺ g'};
20         N'.conflict ← empty;
21         foreach g'' ∈ ({g'' | g' ≺_{N'} g''} ∪ {g'}) do
22             N'.paths[g''] ← empty;
23         Insert N' into STACK;
24 return "No Solution";
```

Algorithm 2: UpdatePath(PT node N)

```
25 TopologicalSort(Γ, ≺_N);
26 foreach g ∈ Γ do
27     if N.paths[g] is empty then
28         p ← PlanPath(N, g);                    // Algorithm 3
29         if p is empty then
30             return false;
31         N.paths[g] ← p;
32     g' ← FindConflictingGoal(N, g);
33     if g' is not empty then
34         N.conflict ← (g, g');
35         return true;
36 return true;
```

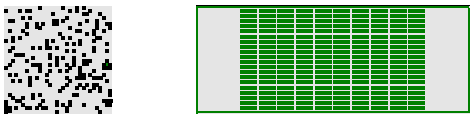
Algorithm 3: PlanPath(PT node N, goal g_i^j)

```
37 P ← {N.paths[g] | g ≺_N g_i^j};
38 if j > 1 then
39     t_0 ← CompletionTimestep(N.paths[g_i^{j-1}]);
40     loc_0 ← g_i^{j-1}.loc;
41 else
42     t_0 ← 0;
43     loc_0 ← s_i;
44 T ← max{CompletionTimestep(N.paths[g]) | ⟨g, g_i^j⟩ ∈ 𝒯}
     or −1 if there is no such g that ⟨g, g_i^j⟩ ∈ 𝒯;
45 p ← a minimum-cost path segment for goal g_i^j that starts at
     vertex loc_0 at timestep t_0, ends at vertex g_i^j.loc after
     timestep T, and does not conflict with any path in P (or
     empty if no such path exists);
46 return p;
```

node with a new pair of goals. Let g' denote the goal of the lower priority in the new ordered pair. PBS-PC empties the path segments of g' and all paths that are of lower priority than it.

Algorithm 3 plans the path segment for goal g_i^j. If $j = 1$, that is, g_i^j is the first goal of the agent, the start timestep and start vertex of search are set to 0 and the start vertex of the agent, respectively. Otherwise, the start timestep and start vertex are set to the completion timestep and goal vertex of the immediate previous goal of g_i^j, respectively. The earliest timestep when a_i is allowed to complete g_i^j can be computed by checking the completion timesteps of all goals that need to be completed before g_i^j (whose path segments have already been planned because goals are planned in a topologically sorted order).

In any PT node of PBS-PC, there is no precedence conflict between any two non-empty path segments because Algorithm 3 only finds path segments that satisfy all precedence constraints. The returned solution does not contain a vertex or edge conflict because, if there is one, the conflict will be found in Line 32, and PBS-PC would not return the set of paths as a solution. Therefore, solutions returned by PBS-PC are conflict-free. Similar to PBS, PBS-PC is neither complete nor optimal.



(a) random-32-32-20 (b) warehouse-10-20-10-2-1

Figure 5: The grid maps of the MAPF-PC instances used in the experimental evaluation.

6 EXPERIMENTAL EVALUATION

In this section, we compare the results of different variants of CBS-PC, PBS, and PBS-PC on MAPF-PC instances with four-neighbor grid maps. The variants of CBS-PC are CBS-PC, CBS-PC-c, CBS-PC-t, CBS-PC-d, and CBS-PC-dct, where c adds constraint propagation, d adds disjoint splitting, and t adds target reasoning. All algorithms were implemented in C++[2] and share the same code base as much as possible. We ran all experiments on t2.large AWS EC2 instances with 8GB of memory. The time limit for solving each MAPF-PC instance was five minutes.

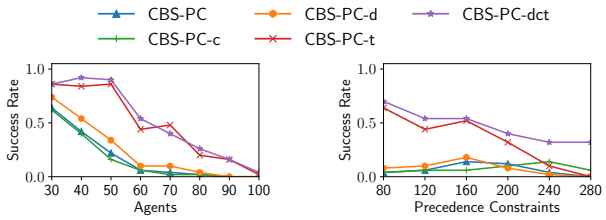
To generate a MAPF-PC instance, we randomly generated the start vertex of each agent and a set of goal vertices. Then, we began with an empty precedence constraint set $\mathcal{T}$, repeatedly picked a random precedence constraint and added it to $\mathcal{T}$ if it was not in $\mathcal{T}$ already and would not introduce cycles, until the number of precedence constraints reached a given number (specified below). Then, we used the Token Passing algorithm [11] to assign goals greedily and generate the goal sequence for each agent.

We picked two grid maps from the MAPF benchmark [16]: *random-32-32-20* and *warehouse-10-20-10-2-1* (shown in Figure 5). For each grid map, we ran two sets of experiments: (1) MAPF-PC instances with different numbers of agents (ranging from 30 to 100), 200 goals, and 120 precedence constraints. (2) MAPF-PC instances with different numbers of precedence constraints (ranging from 80 to 280), 200 goals, and 60 agents. For each number of agents or precedence constraints, we generated 50 random instances.

Comparing variants of CBS-PC: Figures 6 and 7 show the results for the CBS-PC variants. The *success rate* of an algorithm is the percentage of MAPF-PC instances that it solves within the time limit. For the CBS-PC variants without target reasoning, CBS-PC-d almost always had slightly higher success rates than CBS-PC. CBS-PC-c had similar or slightly worse success rates than the other two variants when the number of precedence constraints was less than 200 because the computational overhead outweighs the benefit of the technique. However, it had better success rates than the other two variants when the number of precedence constraints was large enough (namely, at least 240) because constraint propagation significantly reduced the number of precedence conflicts that need to be resolved.

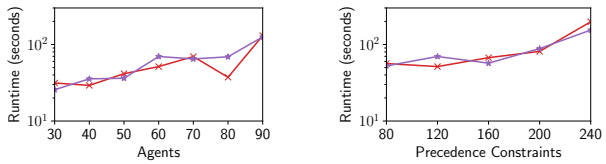
The CBS-PC variants with target reasoning, CBS-PC-t and CBS-PC-dct, had better success rates than the other three variants in most of the experiments. CBS-PC-dct had better success rates than

[2] https://github.com/HanZhang39/MAPF-PC



(a) Success rates for different numbers of agents

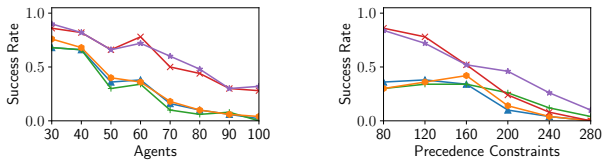
(b) Success rates for different numbers of precedence constraints



(c) Runtimes for different numbers of agents

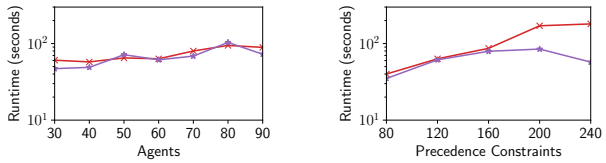
(d) Runtimes for different numbers of precedence constraints

Figure 6: Results for CBS-PC variants on random-32-32-20.



(a) Success rates for different numbers of agents

(b) Success rates for different numbers of precedence constraints



(c) Runtimes for different numbers of agents

(d) Runtimes for different numbers of precedence constraints

Figure 7: Results for CBS-PC variants on warehouse-10-20-10-2-1.

CBS-PC-t when the number of precedence constraints was large but similar success rates otherwise. The runtimes of CBS-PC-dct and CBS-PC-t are averaged over all instances solved by both of them. CBS-PC-dct and CBS-PC-t had similar average runtimes except for some instances with a large number of precedence constraints that CBS-PC-dct solved faster. We omit the runtimes of the other three variants because they solved too few instances within the time limit.

Comparing PBS and PBS-PC: Figures 8 and 9 show the results for PBS and PBS-PC. In general, PBS-PC outperformed PBS in terms of both success rate and runtime. The success rates of PBS stayed similar or even increased when the number of agents increased. However, the success rates of PBS quickly dropped when the number of precedence constraints increased, which shows that PBS did not do a good job at solving MAPF-PC instances with complex precedence constraints.

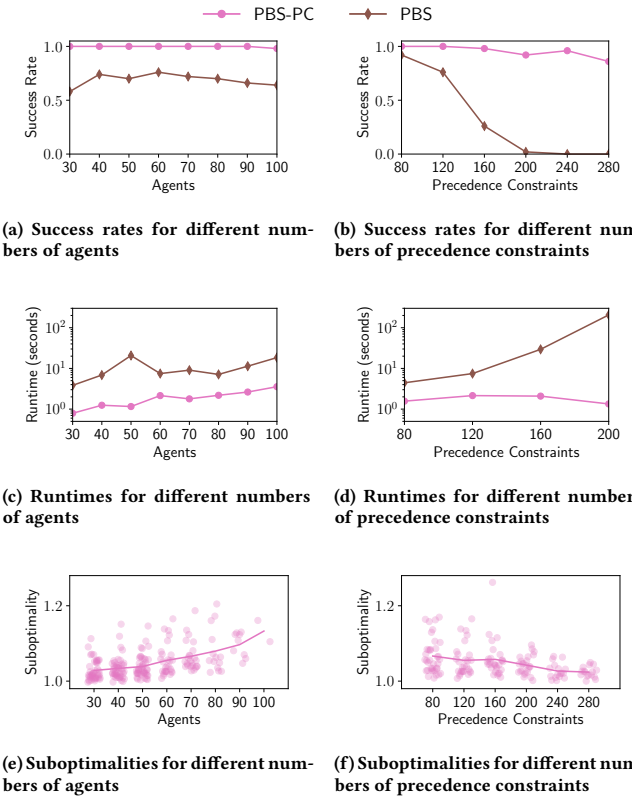
Comparing CBS-PC and PBS-PC: In general, PBS-PC outperformed all CBS-PC variants in terms of both success rate and runtime. In Figures 8e, 8f, 9e, and 9f, we show the suboptimality results for PBS-PC. The *suboptimality ratio* of PBS-PC on an instance is the ratio of the SoC of the solution it finds to the optimal SoC. The dots show the suboptimality ratios of PBS-PC on MAPF-PC instances that were solved by some variant of CBS-PC and the lines show the average suboptimality ratios. The suboptimality ratios of PBS-PC were on average less than 1.1 in most cases and around 1.2 in the worst case. The average suboptimality ratios of PBS-PC increased as the number of agents increased. Interestingly, the average suboptimality ratios of PBS-PC decreased as the number of precedence constraints increased, likely because agents are less likely to have vertex or edge conflicts when they need to wait longer due to the increasing number of precedence constraints.

Scalability of PBS-PC: We ran two additional experiments for only PBS-PC on warehouse-10-20-10-2-1 to see how PBS-PC scales on difficult instances: (1) MAPF-PC instances with different numbers of agents m (ranging from 100 to 500), $2m$ goals, and m precedence constraints. (2) MAPF-PC instances with different numbers of goals n (ranging from 800 to 1800), $0.5n$ precedence constraints were $0.5n$, and 200 agents. Figure 10 shows the results. PBS-PC solved all instances with up to 300 agents in Experiment (1) and all instances with 800 goals in Experiment (2).

7 RELATED WORK

Numerous algorithms have been developed to solve multi-task multi-agent path-finding problems by assigning tasks (in form of goal locations) to agents with the purpose of minimizing the execution time. One representative approach is to formulate the problem as Vehicle Routing Problem with Time Windows (VRPTW) and minimize the execution time over the entire time horizon [2, 6]. A survey of task-assignment algorithms can be found in [4]. However, most of these algorithms ignore collisions and thus cannot be directly used in safety-critical scenarios. Moreover, as our MAPF-PC algorithms are capable of planning with goal vertex sequences and precedence constraints, they can be used in conjunction with most of the aforementioned task-assignment algorithms to generate collision-free paths with respect to the assigned goal vertex sequences.

Among all algorithms that plan collision-free paths for streams of tasks, the algorithms in [2, 13] are able to handle precedence constraints between tasks and thus are most related to ours. [2] presents a four-level algorithm that is able to solve the general Precedence-Constrained multi-agent Task Assignment and Path Finding (PC-TAPF) problem, although it is demonstrated only on assembly scenarios. Its first level iteratively searches for promising task assignments, and the other three levels plan collision-free paths based on the task assignment. As the path planning module of this algorithm solves the same problem as MAPF-PC, it is neither complete nor optimal since it plans each path segment in a myopic way and thus does not consider the feasibility or optimality



(a) Success rates for different numbers of agents

(b) Success rates for different numbers of precedence constraints

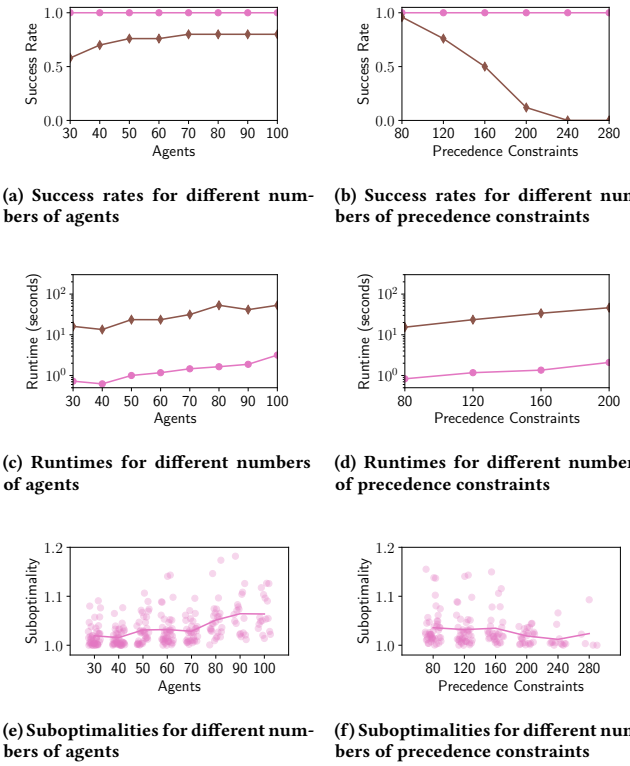
(c) Runtimes for different numbers of agents

(d) Runtimes for different numbers of precedence constraints

(e) Suboptimalities for different numbers of agents

(f) Suboptimalities for different numbers of precedence constraints

Figure 8: Results for PBS-PC and PBS on random-32-32-20.



(a) Success rates for different numbers of agents

(b) Success rates for different numbers of precedence constraints

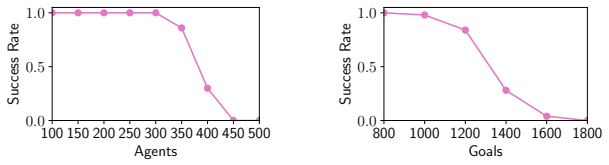
(c) Runtimes for different numbers of agents

(d) Runtimes for different numbers of precedence constraints

(e) Suboptimalities for different numbers of agents

(f) Suboptimalities for different numbers of precedence constraints

Figure 9: Results for PBS-PC and PBS on warehouse-10-20-10-2-1.



(a) Success rates for different numbers of agents

(b) Success rates for different numbers of goals

Figure 10: Scalability results for PBS-PC on warehouse-10-20-10-2-1.

of achieving the subsequent tasks. In comparison, CBS-PC generates provably optimal plans. While both the algorithm in [2] and PBS-PC are suboptimal, PBS-PC is able to plan for 300 agents with 800 goals while the algorithm in [2] can merely plan for 40 agents with 60 tasks in the same runtime. [13] also presents a complete algorithm for solving a multi-task multi-robot path-finding problem with precedence constraints. However, this algorithm relies on the assumption that each task can only have one precedence constraint, which prevents the algorithm from solving realistic scenarios with complex precedence constraints. For example, one needs to introduce several precedence constraints to model a scenario where multiple packages need to be delivered to the same location before a robot picks them up all together.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed the MAPF-PC algorithms CBS-PC and PBS-PC. CBS-PC is complete and optimal, and we proposed several improvements for it. PBS-PC is incomplete and suboptimal but efficient in practice. Our experimental results showed that the most advanced CBS-PC variants scale to dozens of agents and hundreds of goals and precedence constraints and PBS-PC scales to hundreds of agents, around one thousand goals, and hundreds of precedence constraints.

An interesting direction for future work is to extend the MAPF-PC problem with other types of inter-goal constraints, such as simple temporal constraints between the completion timesteps of goals. Another direction is to study different types of MAPF-PC algorithms, such as bounded sub-optimal or anytime algorithms.

9 ACKNOWLEDGMENTS

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712 as well as a gift from Amazon. The research at Massachusetts Institute of Technology was supported by Kawasaki Heavy Industry, Ltd (KHI) under grant number 030118-00001. This article solely reflects the opinions and conclusions of its authors and not the sponsoring organizations, agencies, or the U.S. government.

REFERENCES

[1] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 740–746.

[2] Kyle Brown, Oriana Peltzer, Martin A. Sehr, Mac Schwager, and Mykel J. Kochenderfer. 2020. Optimal Sequential Task Assignment and Path Finding for Multi-Agent Robotic Assembly Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 441–447.

[3] Rina Dechter, Itay Meiri, and Judea Pearl. 1991. Temporal Constraint Networks. *Artificial intelligence* 49 (1991), 61–95.

[4] Maria L. Gini. 2017. Multi-Robot Allocation of Tasks with Temporal and Ordering Constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 4863–4869.

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

[6] Gilbert Laporte and Ibrahim H. Osman. 1995. Routing Problems: A Bibliography. *Annals of Operations Research* 61, 1 (1995), 227–262.

[7] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. 2020. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 193–201.

[8] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. 2019. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 279–283.

[9] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 11272–11281.

[10] 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 AAAI Conference on Artificial Intelligence (AAAI)*. 7643–7650.

[11] 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 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 837–845.

[12] Hang Ma, Craig A. Tovey, Guni Sharon, T. K. Satish Kumar, and Sven Koenig. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 3166–3173.

[13] James Motes, Read Sandström, Hannah Lee, Shawna Thomas, and Nancy M. Amato. 2020. Multi-Robot Task and Motion Planning with Subtask Dependencies. *IEEE Robotics and Automation Letters* 5, 2 (2020), 3338–3345.

[14] 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.

[15] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 195 (2013), 470–495.

[16] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Proceedings of the Symposium on Combinatorial Search (SoCS)* (2019), 151–158.

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

[18] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 1444–1449.