Generalized and Sub-Optimal Bipartite Constraints for Conflict-Based Search

Thayne T. Walker,[1] Nathan R. Sturtevant,[2] Ariel Felner[3]
[1]University of Denver, Denver, USA
[2]University of Alberta, Edmonton, Canada
[3]Ben-Gurion University, Be'er-Sheva, Israel
thayne.walker@du.edu, nathanst@ualberta.ca, felner@bgu.ac.il

Abstract

The main idea of conflict-based search (CBS), a popular, state-of-the-art algorithm for multi-agent pathfinding is to resolve conflicts between agents by systematically adding constraints to agents. Recently, CBS has been adapted for new domains and variants, including non-unit costs and continuous time settings. These adaptations require new types of constraints. This paper introduces a new automatic constraint generation technique called bipartite reduction (BR). BR converts the constraint generation step of CBS to a surrogate bipartite graph problem. The properties of BR guarantee completeness and optimality for CBS. Also, BR's properties may be relaxed to obtain suboptimal solutions. Empirical results show that BR yields significant speedups in 2^k connected grids over the previous state-of-the-art for both optimal and suboptimal search.

1 Introduction

The goal of multi-agent pathfinding (MAPF) is to move multiple agents to their respective goal states while avoiding conflicts between agents. Conflict-based search (CBS) (Sharon et al. 2015), a state-of-the-art algorithm for MAPF, is designed around detecting and resolving conflicts between agents by systematically adding constraints to agents and has been used for many variants of the MAPF problem (Atzmon et al. 2018; Li et al. 2019b; Ma et al. 2018; Li et al. 2019a; Thomas, Deodhare, and Murty 2015; Hönig et al. 2018). Most notably, CBS has been adapted for non-unit cost and continuous time domains (Andreychuk et al. 2019; Cohen et al. 2019). Many of these adaptations introduced new types of constraints or new ways to manage constraints. In each case, rigorous proofs are required to guarantee that the new enhancements preserve completeness and optimality. There is a need for a generalized technique that can automate this process, simplifying the application of CBS to new domains.

This paper has three main contributions. First, a new technique called *bipartite reduction* (BR). This is a general constraint generation technique for CBS that can be applied to new domains and variants which guarantees completeness and optimality. BR converts the constraint generation step of CBS to a surrogate bipartite graph problem. BR only requires a low-level (single-agent) successor generation routine and a conflict detection routine and can return constraints with both spatial and temporal extents. Second, a method of introducing conditional constraints for suboptimal search which exploits constraint relaxations to quickly find suboptimal solutions fast while maintaining completeness. Finally, empirical analysis showing significant improvement over the previous state-of-the-art for both optimal and sub-optimal variants in 2^k-connected grids.

2 Problem Definition

MAPF was originally defined for agents that occupy a single vertex and move on unit-cost edges. This paper demonstrates the new automation technique (BR) on MAPF$_R$ (Walker, Sturtevant, and Felner 2018) a variant of MAPF for real-valued weighted graphs. A MAPF$_R$ problem instance is defined by a tuple (G, A, V_s, V_g). $G = (V, E)$ is a positive-weighted graph, $\forall e \in E, w(e) \in \mathbb{R}_+$. Each $v \in V$ is associated with unique coordinates in a metric space. $A = \{1, ..., k\}$ is a set of k agents. $V_s \subseteq V = \{start_1, ..., start_k\}$ and $V_g \subseteq V = \{goal_1, ..., goal_k\}$ are sets of unique start and goals for each agent where $start_i \neq start_j, goal_i \neq goal_j$ for all $i \neq j$.

A *solution* to MAPF is $\Pi = \{\pi_1, ..., \pi_k\}$, a set of single-agent *paths* composed of *states*. A state $s = (v, t)$ is a pair composed of a vertex $v \in V$ and time $t \in \mathbb{R}_+$. We denote the vertex and time of a state as $v(s)$ and $t(s)$ respectively. A path for agent i is a sequence of d states $\pi_i = [s_i^0, ..., s_i^d]$ where $s_i^0 = (start_i, 0)$ and $s_i^d = (goal_i, t)$. Each $(v(s_i^n), v(s_i^{n+1})) \in E$ and each $t(s_i^{n+1}) = t(s_i^n) + w(e_i^n)$. That is, states on the path are connected by edges, and the weight of an edge is the time it takes to traverse it.

Agents have a physical shape such as spheres, polygons or polygonal meshes which are situated relative to their *reference point* (Li et al. 2019b). Agents move along edges $(v(s^n), v(s^{n+1})) \in E$ — an agent begins with its reference point at $v(s^n)$ at time $t(s^n)$ and its reference point follows a straight, constant velocity motion vector in metric space, ending at $v(s^{n+1})$ at time $t(s^{n+1})$. For simplicity, we assume that the time it takes to traverse an edge is identical to the weight of the edge (more complicated assumptions can be made). The act of traversing an edge is an *action*, and a path is a sequence of actions $\pi_i = [a_i^0, ..., a_i^d]$ where each

$a_i^n = (s_i^n, s_i^{n+1})$. MAPF$_\text{R}$ also allows self-directed edges for *wait* actions.

A *conflict* is denoted by $\langle a_i, a_j \rangle$ – a pair of actions for agent i and j in which their shapes overlap. A *feasible* solution is one in which no two agents come into conflict at any time during their respective paths in Π, i.e., the shapes never overlap. In this paper the objective is to minimize flowtime, the sum of individual path costs. An optimal solution Π^* has minimal cost among all feasible solutions. Finding optimal solutions to the classic MAPF problem is NP-hard (Yu and LaValle 2013). As the classic version is a special case of MAPF$_\text{R}$, MAPF$_\text{R}$ is also NP-hard.

3 Background: The CBS Algorithm

CBS (Sharon et al. 2015) is a two-level algorithm for MAPF. The *high level* searches a conflict tree (CT). Each node $N \in CT$ contains a possible solution $N.\Pi$. Each $\pi \in N.\Pi$ for the root node is constructed using a *low level* search without taking other agents into account. $N.\Pi$ is checked for conflicts between paths. If no conflict is found, N is a goal and CBS terminates. If a conflict is found, CBS performs a *split*, where two child nodes N_i and N_j are generated with constraints c_i and c_j. A constraint blocks an agent from performing one or more actions that caused the conflict. In section 4 we discuss the details for constraints. Next, the low level is invoked for N_i and N_j while adding the new constraints c_i and c_j respectively to re-plan paths π_i and π_j and update N_i and N_j. All CT nodes are placed into an OPEN list which is prioritized by flowtime. The search terminates when a feasible solution is found or when OPEN is empty.

CBS is *solution complete*. That is, it is guaranteed to find a solution only if one exists, otherwise it may run forever. This problem can be mitigated by running a polynomial-time algorithm (Botea, Bonusi, and Surynek 2018) in parallel to determine if a solution exists, although general polynomial-time algorithms may not exist for MAPF$_\text{R}$.

Several optimal enhancements for CBS in the classic problem have been published, (Boyarski et al. 2015b; Felner et al. 2018; Li et al. 2019a; Gange, Harabor, and Stuckey 2019). CBS has also been extended for several new problem variants: large agents (MC-CBS) (Li et al. 2019b), robustness (Atzmon et al. 2018), road maps (Hönig et al. 2018), convoys (Thomas, Deodhare, and Murty 2015), trains (Atzmon, Diei, and Rave 2019) and deadlines (Ma et al. 2018). Sub-optimal algorithms have also been published (Barer et al. 2014; Cohen and Koenig 2016; Walker, Chan, and Sturtevant 2017; Cohen et al. 2018).

Prior Work in MAPF$_\text{R}$ Optimal and complete solvers were published for MAPF$_\text{R}$. The ICTS algorithm was extended for MAPF$_\text{R}$ (Walker, Sturtevant, and Felner 2018) and was shown to outperform A* and CBS with classic *edge constraints* and *vertex constraints*. These constraints block low-level solvers from traversing an edge or a vertex respectively at a specific time. The CCBS algorithm (Andreychuk et al. 2019) and the ECBS-CT algorithm (Cohen et al. 2019) adapt CBS for MAPF$_\text{R}$ via the use of *time-range* constraints (Atzmon et al. 2018) with the SIPP algorithm (Phillips and Likhachev 2011) at the low level. The most significant difference between CCBS and ECBS-CT is that the former uses SIPP verbatim and latter modifies SIPP to reason about conflicts in order to inform tie-breaking and sub-optimal search heuristics. Because of the SIPP reservation table, these algorithms are only valid for discretized spaces such as grid maps or robotic latices.

A time-range constraint $\langle e, [t_{start}, t_{end}) \rangle$ blocks all actions which cause an agent to traverse an edge e during an *unsafe interval* $[t_{start}, t_{end})$. An unsafe interval is the time interval in which agent i, traversing e is guaranteed to collide with another agent j traversing another edge in an overlapping time interval. Computing unsafe intervals can be done using an incremental approach (Andreychuk et al. 2019), or in closed-form in the special case of circular agents (Walker and Sturtevant 2019). Time-range constraints are more powerful than simple edge constraints because they may block multiple actions, resulting in more pruning of the CT.

4 The Bipartite Reduction Technique

CBS with *time-annotated bicliques* (CBS+TAB) utilizes bipartite reduction (BR) and is a generalization of multi-constraint CBS (MC-CBS) (Li et al. 2019b) which allows constraint *sets* C, to be used with CT nodes. MC-CBS was applied to unit-cost domains with large agents which was shown to reduce the number of CT node expansions because more conflicts are resolved per CT node. The pruning by a constraint is positively correlated with the cardinality of A, the set of actions blocked by C. A single constraint c (such as a time-range constraint or vertex constraint) may block a set of actions A_c. Prior work in MAPF$_\text{R}$ used a single time-range constraint per CT node in order to block multiple actions in a time interval. We define this *blocking* as a one-to-many relation between constraints and actions: $c \mapsto A_c$. CBS+TAB uses multiple time-range constraints per CT node. Therefore, the set of blocked actions A for the set of constraints C is the union of blocked action sets: $A = \bigcup_{c \in C} A_c$.

Because $|A|$ is correlated with pruning, it is beneficial to maximize $|A|$. CBS+TAB heuristically maximizes $|A|$ without losing completeness or optimality. The algorithm is general and may be used with or without SIPP at the low level and on discretized or continuous environments.

Ensuring Completeness and Optimality CBS+TAB is designed to use more comprehensive constraints while ensuring completeness and optimality. For completeness, constraint sets C_i and C_j which are created during a split, must be *mutually disjunctive* (Atzmon et al. 2018; Li et al. 2019b). That is, no pair of conflict-free paths for agents i and j can violate both C_i and C_j. C_i and C_j are mutually disjunctive if their corresponding blocked action sets A_i, A_j are *mutually conflicting*. Two sets of actions A_i, A_j are **mutually conflicting** if for all pairs of actions (a_i, a_j) in the Cartesian product $A_i \times A_j$, a_i conflicts with a_j.

For example, in Figure 1(a) action 1 conflicts with 6, 7 and 8; action 2 conflicts with 6, 7 and 8 and action 3 conflicts with 6, 7, 8, 9 and 10. Thus the action sets $\{1, 2, 3\}$, $\{6, 7, 8\}$ are mutually conflicting. Although actions 9 and 10 also

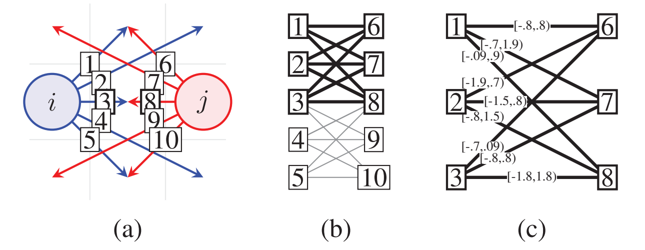
Figure 1: Illustration of (a) actions for two agents, (b) the corresponding BCG and (c) the corresponding TAB.

conflict with action 3, they cannot be included because they do not conflict with actions 1 and 2.

Lemma 4.1. *CBS is complete if all pairs of blocked action sets A_i, A_j affected by a CBS split are mutually conflicting.*

Proof. By contradiction, if $\exists(a_i, a_j) \in A_i \times A_j$ such that a_i does not conflict with a_j, then there may exist a pair of paths π_i^*, π_j^* from a feasible solution Π^* where $a_i \in \pi_i^*$ and $a_j \in \pi_j^*$. Hence, constraints for A_i, A_j would not be mutually disjunctive and would render CBS incomplete. □

The proof of optimality is the same as for the original CBS (Sharon et al. 2015). In short, optimality is guaranteed if no optimal solutions are blocked by constraints (i.e. mutually disjunctive constraints) and both the high and low level OPEN lists are prioritized by flowtime. CBS+TAB reduces the problem of constructing mutually conflicting constraint sets to finding a biclique in a bipartite graph.

Reduction to Bipartite Graphs The conflicts between a pair of action sets A_i and A_j, (shown as arrows in Figure 1(a)), can be represented as a *bipartite conflict graph* (BCG), shown in Figure 1(b). A BCG, $G = (U, V, E)$, has two sets of vertices U and V such that each $u \in U$ represents an action $a_i \in A_i$ and each $v \in V$ represents an action $a_j \in A_j$. E consists of the subset of vertex pairs $(u, v) \in U \times V$ for which the corresponding actions $(a_i, a_j) \in A_i \times A_j$ conflict.

For CBS, it is sufficient to construct a BCG only for the subset of actions which conflict with the *core action pair* which is the actions from the conflict $\langle a_i, a_j \rangle$ that caused a split. In Figure 1(a), the core action pair is $\langle 3, 8 \rangle$, hence only actions which conflict with 3 or 8 are depicted. In this setting, each vertex is guaranteed to be connected to the opposing agent's core action in the BCG.

Although Figure 1 shows biclique construction based only on actions from the start states of the core action pair, in practice, a BCG can include all actions from all states that conflict with an opposing agent's core action. However, it may not be computationally efficient to do so.

Constraint Set Construction Using Bicliques A *biclique* $G' = (U', V', E') \subseteq G$ is a fully bi-connected bipartite graph, that is, $E' = U' \times V'$, meaning all $u \in U'$ are connected via an edge to all $v \in V'$. A BCG may have many bicliques. In order to maximize pruning in the CT, we find a *max-vertex biclique* (MVB) in G which is a biclique with a maximal number of vertices. This can be done in polynomial time

Algorithm 1 ComputeLargestTimeAnnotatedBiclique

```
1. INPUT: A bipartite graph $G = (U, V, E)$
2. Construct $\overline{G}$, the bipartite complement of $G$
3. Find $M$, a maximal matching in $\overline{G}$
4. Construct $K$, a minimum vertex cover of $\overline{G}$ from $M$
5. Take the bipartite complement of $K$ to get $G' \subseteq G$, a max-vertex biclique
6. Annotate all edges $e \in E' \in G'$ with computed unsafe intervals to create a time-annotated biclique $G'_t$:
   For each $e \in E'$, $E'_t \leftarrow E'_t \cup (e, \text{UNSAFEINTERVAL}(e.u, e.v))$
7. Annotate all vertices $U'_t, V'_t$ with the intersection of all unsafe intervals of incident edges:
   For each $u \in U'_t$, $u \leftarrow (u, \bigcap_{e \in \text{INCIDENT}(u)} e.intvl)$; analogously for $V'_t$
8. return $(U'_t, V'_t)$
```

(Garey and Johnson 2002). Algorithm 1, lines 1-5 shows pseudocode for computing a MVB. Because G' is fully bi-connected, U' and V' represent the mutually conflicting action sets suitable for a split, and edge constraints could be used to block these actions. In CBS, edge constraints are only for a single time t. However, given a MVB, unsafe intervals can be computed and time-range constraints can be used (see Section 3).

After extracting G' from G, G'_t, a *time-annotated biclique* (TAB) is constructed (Algorithm 1, line 6) by annotating each edge $e' \in E'$ with its unsafe interval (see Section 3). An example of a TAB is shown in Figure 1(c). Finally each vertex in U'_t, V'_t is annotated with an interval that is *fully included* by the annotated intervals for each $e \in E'_t$ incident to it (line 7). An interval $tr_i = [t_i^{start}, t_i^{end})$ **fully includes** another interval $tr_j = [t_j^{start}, t_j^{end})$ if $tr_i^{start} \leq tr_j^{start}$ and $tr_i^{end} \geq tr_j^{end}$. In set notation, this is denoted $tr_j \subseteq tr_i$.

A time interval tr_i is fully included by a set of time ranges T if $tr_i \subseteq \bigcap_{tr_j \in T} tr_j$. This relation is illustrated in Figure 2 (b) – the interval in blue is fully included by all other intervals. Figure 2(a) illustrates the annotation of a vertex in a TAB. The blue time interval annotation on vertex 1 is the intersection of all intervals annotated on its adjacent edges as shown by the blue interval in part (b). Thus, the result of Algorithm 1 is a TAB where each edge is annotated with an unsafe interval between two actions, and each vertex is annotated with an unsafe interval which is fully included by the intervals of its incident edges.

Thus, for a split we first build the relevant TAB. Then for

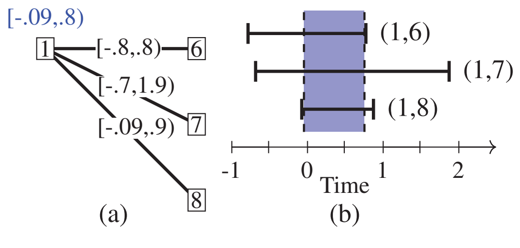
Figure 2: Example of (a) a TAB with (b) corresponding unsafe intervals plotted on a concurrent timeline.

the left node we add the set of time-range constraints C_i that includes $\langle u, tr = [t_{start}, t_{end})\rangle$ for each $u \in U'_t$ where $[t_{start}, t_{end})$ is the unsafe interval associated with it. This is then done analogously for the right node using V'_t. This approach yields constraint sets that guarantee completeness.

Theorem 4.2. *CBS+TAB is complete.*

Proof. First, the action sets $U', V' \in G'$ (line 5 of Algorithm 1) are guaranteed to be mutually conflicting because G' is a biclique. Second, since the annotated unsafe interval for each vertex $u_t \in U'_t$ and $v_t \in V'_t$ is the intersection of all unsafe intervals of incident edges $\in E'_t$ (line 6,7), all time range constraints $c_i \in C_i$ and $c_j \in C_j$ constructed from those intervals (Algorithm 2 lines 13,14) are guaranteed to block only actions that conflict. Hence, C_i, C_j are mutually disjunctive. Thus, per Lemma 4.1 completeness is guaranteed. □

Black portions of Algorithm 2 are the core algorithm for the CBS+TAB split operation. Gray portions are sub-optimal enhancements discussed in the next section and can be disregarded for now. CBS+TAB computes a TAB and creates time range constraints on lines 9-12 then re-plans the agents with those constraints on lines 20, 21.

Additional Variants We define two additional variants which utilize BR: **CBS+MVB**, simply omits the time annotation step (line 9) and uses U' and V' from the MVB to create edge constraints instead of time-range constraints. This variant may be required for some domains in which computing unsafe intervals is not possible or too expensive. **CBS+TMA** (for time-annotated max-biclique approximation), approximates a TAB by assuming that the MVB is a $1 \times N$ biclique, that is, $|U'| = 1$ and $|V'| = N$. For example, using the sets $\{3\}$,$\{6,7,8,9,10\}$ from Figure 1. However, instead of explicitly blocking each action in U'_t and V'_t, the TAB is represented *implicitly*, using only two constraints, one edge constraint c_i for agent i that blocks a_i (this is U'_t), and another constraint c_j for agent j that blocks all actions that conflict with a_i (this is V'_t). c_j in this case is implemented such that it performs a collision check versus a_i during low-level expansions. With this representation, c_i and c_j can be created without constructing a BCG.

5 Sub-Optimal, Complete Constraints

CBS+TAB constraints block large sets of actions in order to maximize pruning of the CT. It is possible to further increase the number of blocked actions by relaxing the mutually disjunctive requirement. For example, by blocking all actions in the BCG. However, doing so may result in incompleteness in two ways: (1) termination at the low level without finding a path or (2) agents being constrained in such a way that each low-level search is successful, but no feasible solution is found. In situation (2) collisions tend to recur over and over at increasingly later times, causing the algorithm to run forever. For completeness, we must detect and avoid these two conditions. For this purpose, we introduce *conditional constraints*.

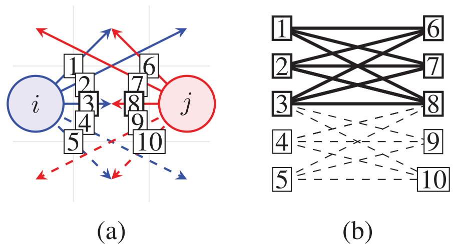

Figure 3: Illustration of (a) sets of available actions for two agents and (b) the corresponding BCG.

Conditional Constraints Constraints for a CT node N apply *permanently* to N and are inherited by all CT nodes in the sub-tree of N. *Conditional* constraints are turned on by default, but may be turned off, meaning they no longer block any actions in N or its sub-tree. A constraint is turned off by omitting it from the low-level re-plan step after a split operation. Per Lemma 4.1, if an action that is not in the biclique is blocked, CBS is no longer complete. To avoid this, mutually-conflicting actions from the MVB are always blocked permanently and other actions not in the MVB are blocked conditionally, so that those actions may be unblocked to avoid incompleteness.

Figure 3(a) and the corresponding BCG in (b) are shown for the same scenario as Figure 1: Actions corresponding to the MVB are permanent and shown with bold lines. All other actions in the BCG are shown with dashed lines – these are the set of conditional constraints.

Algorithm 2 Expand-CT-Node

```
1:  Input: $N$ – a CT node
2:  $\langle a_i, a_j\rangle$ ←find-conflict($N.\Pi$)
3:  if No conflict return $N.\Pi$ as goal
4:  $N_i \leftarrow N; N_j \leftarrow N$ // Copy $N$ to child nodes for split
5:  Get conflict counts $\Delta_i$, $\Delta_j$: the number of conflicts from $N$ to root
6:  Get length of path $d_i$, $d_j$ in CT root node for $i$ and $j$
7:  // Compute BCG and biclique for core action pair
8:  $(U, V, E) \leftarrow CreateBCG(a_i, a_j)$
9:  $(U'_t, V'_t) \leftarrow ComputeMaxVertexTAB(U, V, E)$
10: // Create constraints
11: $N_i.C \leftarrow N_i.C \cup CreatePermanentConstraints(U'_t)$
12: $N_j.C \leftarrow N_j.C \cup CreatePermanentConstraints(V'_t)$
13: $N_i.C \leftarrow N_i.C \cup CreateConditionalConstraints(U \setminus U'_t)$
14: $N_j.C \leftarrow N_j.C \cup CreateConditionalConstraints(V \setminus V'_t)$
15: // Create probabilistically filtered sets
16: $\rho_i \leftarrow \text{MIN}((\Delta_i - 1)/d_i, 1.0)$; $\rho_j \leftarrow \text{MIN}[e]((\Delta_j - 1)/d_j, 1.0)$
17: Remove conditional constraints from $N_i.C$ with probability $\rho_i$
18: Remove conditional constraints from $N_j.C$ with probability $\rho_j$
19: // Re-plan with (filtered) constraint sets
20: $N_i.\Pi \leftarrow Replan(start_i, goal_i, N_i.C)$
21: $N_j.\Pi \leftarrow Replan(start_j, goal_j, N_j.C)$
22: // Check for no path and re-plan without conditional constraints
23: if $N_i.\Pi.\pi_i = \emptyset$ then
24:     Remove all conditional constraints from $N_i.C$
25:     $N_i.\Pi \leftarrow Replan(N.\Pi.\pi_i, N_i.C)$
26: end if
27: if $N_j.\Pi.\pi_j = \emptyset$ then
28:     Remove all conditional constraints from $N_j.C$
29:     $N_j.\Pi.\pi_j \leftarrow Replan(N.\Pi.\pi_j, N_j.C)$
30: end if
31: Add $N_i, N_j$ to OPEN
```

Enhancements for implementing conditional constraints are highlighted in gray in Algorithm 2. After detecting a conflict between two core actions (line 2) child nodes N_i, N_j are created as copies of N (line 4). Then the steps for creating permanent constraints are executed in the same manner as described in Section 4 (lines 8-12). Then conditional constraints are created from $U \setminus U'_t$ and $V \setminus V'_t$ where U and V are from the BCG, and U'_t and V'_t are from the TAB (lines 13,14). Then conditional constraints are turned off according to the two causes of incompleteness as follows:

Situation (1) may occur when a low-level re-plan for an agent returns no path because a conditional constraint may have blocked a feasible path (lines 23, 27). When this occurs, conditional constraints are removed from $N_i.C$ and $N_j.C$ and the re-plan is performed again (lines 25, 29).

Situation (2) is difficult or impossible to detect but intuitively, if a single agent incurs many collisions, it is likely to be in this situation. Because this situation is caused by one of the conditional constraints, we use a strategy to turn them off *probabilistically*. Specifically, they are turned off with an increasing probability $\rho_{\text{off}} = \text{MIN}(1, {}^{(\Delta_i - 1)}/_{d_i})$ (line 16) where d_i (line 6) is the length of the path for agent i in the root CT node and Δ_i (line 5) is the number of conflicts with agent i in CT nodes from N to the root. As the search progresses, if agent i has recurring conflicts, Δ_i will grow relative to d_i increasing ρ_{off}, resulting in a higher proportion of conditional constraints being turned off. Eventually, any conditional constraints causing situation (2) to occur will be turned off, allowing a goal to be found. We call this algorithm **CBS+TCC** (TAB with conditional constraints).

Theorem 5.1. *CBS+TCC is complete.*

Proof. First, no feasible solution is ever blocked by permanent constraints because they will never block a feasible solution per Lemma 4.1. Second, there are two cases to consider for any conditional constraint $c \in C_c$, where $C_c \subset C$ is the set of conditional constraints from N:
Case 1: *c blocks an action in a feasible solution.* If all feasible solutions are blocked, a conflict resulting from situation (1) or (2) will occur. In the case of (1), all conditional constraints are turned off immediately (including c), (lines 23,27) allowing a solution to be found. In the case of (2), if the probabilistic filtering (lines 17,18) does not turn off c at this stage, a new CT node will be created, increasing Δ_i. This situation may be repeated in subsequent CT nodes with increasing ρ_{off} until c is turned off. Because Δ_i is monotonically increasing, ρ_{off} will reach 1 after a finite number of steps, hence c is guaranteed to be turned off after a finite number of steps, (if a goal is not found in a different sub-tree of the CT first) allowing CBS to complete.
Case 2: *c blocks an action that causes a conflict.* If c is turned off before a goal is found, an agent may now be allowed to take an action which re-introduces a conflict into $N.\Pi$. In this case, either a goal node will be found in a different sub-tree, or the resulting conflict will eventually be detected in the sub-tree of N and a permanent constraint to avoid it will be created, allowing CBS to find a goal.

Eventually, in the worst case, all conditional constraints are turned off and the algorithm reduces to CBS+TAB which is guaranteed complete per Theorem 4.2. □

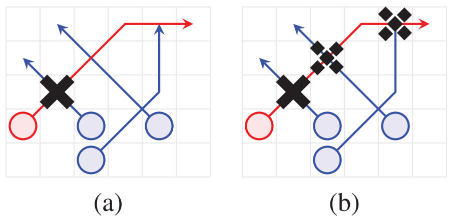
Figure 4: Illustration of (a) regular CBS constraint allocation and (b) allocation with the conflicting paths strategy.

CBS+TCC can yield significant speed-ups over CBS+TAB because it pre-emptively blocks actions that are likely to lead to dead-ends in the CT, resulting in finding a feasible solution sooner. Optimality is not guaranteed because active conditional constraints may block an action $a \in \pi^* \in \Pi^*$ where Π^* is an optimal solution.

The Conflicting Paths Strategy A more powerful blocking strategy called **CBS+TCP** (TAB with conflicting paths) blocks actions that conflict with the paths of all other agents (in addition to agents i and j). This technique has strong resemblances to prioritized planning algorithms (Silver 2005; Van Den Berg and Overmars 2005; Chouhan and Niyogi 2015; 2017). This is done during the CBS feasibility check routine. The first conflict encountered during the check is the *core conflict*. Mutually conflicting actions between agents i and j in the core conflict are blocked using permanent constraints (by computing the TAB for the core action pair). For every conflict between agent i or j and *any other agent* that is encountered thereafter, conditional constraints for all actions in the corresponding BCG are added to C_i or C_j. Figure 4 (a) shows the regular constraint allocation strategy which adds permanent constraints for resolving only one conflict. This is indicated by the black 'x' over the collision area. Diagram (b) shows the CBS+TCP strategy which allocates extra conditional constraints for all conflicts beyond the core conflict as indicated by the dashed 'x's. With CBS+TCP, when agent i (resp. j) is re-planned as part of a split operation, it will attempt to avoid conflicts with all other agents (not just agent j). This technique can result in a significant performance improvement because of aggressive pruning high in the CT.

The same conditions for turning off conditional constraints in CBS+TCC are employed by CBS+TCP, hence it is complete but sub-optimal.

6 Empirical Results

We experimented with CBS+TAB, CBS+TCC and CBS+TCP. All tests were performed on virtual machines with 2.8GHz processors. All implementations use closed-form solutions for conflict detection and unsafe interval computation for circular agents (Walker and Sturtevant 2019) with radius ${}^1/_{2\sqrt{2}}$ which disappear upon reaching their goal.

Table 1: Total problems solved in under 30 seconds on grid MAPF benchmarks

Type	Map	8-Connected						16-Connected						32-Connected					
		ICTS	Classic	Time	MVB	TMA	TAB	ICTS	Classic	Time	MVB	TMA	TAB	ICTS	Classic	Time	MVB	TMA	TAB
City	Berlin_1_256	628	1,626	1,595	1,624	1,785	**1,790**	601	1,356	1,001	1,431	**1,629**	1,570	565	1,104	715	1,212	**1,521**	1,310
	Boston_0_256	623	1,406	1,312	1,446	1,563	**1,574**	559	1,226	858	1,351	**1,463**	1,442	600	855	579	927	**1,205**	974
	Paris_1_256	615	1,545	1,480	1,607	1,653	**1,695**	645	1,207	1,075	1,236	**1,489**	1,393	631	1,076	745	1,124	**1,375**	1,280
DAO	brc202d	363	627	585	**658**	637	**658**	324	460	401	485	485	**505**	329	373	291	386	**473**	413
	den312d	**777**	550	456	558	545	549	450	476	347	477	**507**	502	522	426	273	452	**479**	470
	den520d	941	856	879	880	911	**954**	805	695	670	686	**821**	748	**802**	511	373	521	627	552
	lak303d	520	586	575	594	583	**600**	**542**	369	308	393	449	435	**534**	307	211	326	363	346
	orz900d	227	707	706	736	739	**780**	197	330	320	345	**369**	361	157	260	223	276	**297**	281
	ost003d	**924**	571	615	589	669	687	**622**	468	414	490	541	539	**701**	391	244	421	471	423
Dragon Age 2	ht_chantry	484	638	640	649	**711**	705	318	499	426	529	**585**	559	433	396	282	448	**541**	477
	ht_mansion_n	393	843	774	854	843	**871**	344	561	468	607	**645**	609	377	429	273	511	**561**	513
	lt_gallowstemplar	461	634	633	676	661	**699**	412	568	495	599	607	**625**	364	520	331	549	**571**	553
	w_woundedcoast	322	795	825	865	899	**935**	292	461	411	479	**519**	503	253	326	261	362	**453**	384
Open	empty-8-8	442	451	237	461	485	**493**	384	375	254	387	**445**	386	329	337	134	361	**423**	333
	empty-16-16	429	567	96	592	595	**599**	354	486	259	518	521	**527**	314	412	210	414	**471**	457
	empty-32-32	674	986	70	1,001	1,019	**1,027**	422	832	490	808	**891**	827	438	709	407	735	**841**	762
	empty-48-48	899	1,297	40	1,314	**1,393**	1,307	510	1,196	727	1,204	**1,299**	1,214	484	1,015	506	1,030	**1,205**	1,091
Open+ obstacles	random-32-32-10	487	880	938	910	903	**925**	387	624	358	662	**761**	724	395	575	316	601	**713**	624
	random-32-32-20	305	686	773	699	757	**773**	305	586	354	607	627	**645**	313	518	303	577	605	**615**
	random-64-64-10	656	**1,539**	1,521	1,383	1,483	1,415	519	1,032	632	1,078	**1,203**	1,112	463	857	438	904	**1,085**	953
	random-64-64-20	535	1,068	1,013	1,101	**1,152**	**1,152**	448	732	495	792	853	**882**	480	644	399	662	**811**	774
Maze	maze-32-32-2	239	306	373	308	315	**344**	229	271	251	**291**	271	282	208	232	195	250	259	**260**
	maze-32-32-4	223	297	269	299	291	**304**	170	173	173	184	251	**264**	156	158	152	179	**233**	230
	maze-128-128-10	252	356	309	356	399	**422**	176	244	211	291	**315**	306	180	220	162	250	267	**283**
	maze-128-128-2	232	237	243	250	241	**278**	190	236	190	184	197	**213**	192	203	134	179	**193**	187
Room	room-32-32-4	278	440	347	441	457	**480**	259	382	274	393	395	**426**	267	373	246	378	397	**415**
	room-64-64-16	355	516	426	529	555	**575**	326	405	300	435	485	**513**	281	333	218	367	**443**	424
	room-64-64-8	310	346	299	**399**	371	383	294	291	241	302	315	**345**	267	255	191	263	**309**	306

Table 2: Final size of CT on 16-connected grids

Configuration	Classic	Time	MVB	TMA	TAB
City: Boston	1,422	360	304	180	**174**
DAO: ost003d	2,043	1,197	575	485	**473**
DA2: ht_chantry	2,762	669	607	164	**150**
Open: 16x16	27	25	23	24	**19**
Obstacles: 64x64-20	896	240	602	201	**199**
Maze: maze-32-32-4	13,237	9,332	10,326	6,261	**5,752**
Room: room-64-64-8	1,462	459	1,451	468	**321**

All CBS-based test implementations run in the independence detection framework (Standley 2010). We found that the conflict avoidance table (CAT) (Standley 2010), the bypass enhancement (Boyarski et al. 2015a) and the conflict prioritization enhancement (Boyarski et al. 2015b) were either ineffective or detrimental in 2^k neighborhood environments of 8-connected and higher. Our analysis showed that turning these enhancements off increases average performance. 4-connected grids were not tested because that domain is a planar graph and always yields 1x1 BCGs[1].

Results for Optimal Variants We experiment with CBS+TAB, CBS+MVB and CBS+TMA. CBS+TAB and CBS+MVB use TABs that were computed a-priori and saved in a lookup table. We also experiment with Extended-ICTS (Walker, Sturtevant, and Felner 2018) (denoted *ICTS*), CBS with edge and vertex constraints (Sharon et al. 2015) (denoted *Classic*) and CBS with time-range constraints (Atzmon et al. 2018) – based on CCBS (Andreychuk et al. 2019) and ECBS-CT (Cohen et al. 2019) (denoted *Time*). Our implementation uses A* with a fixed duration of 1 for wait actions at the low level instead of SIPP. Hence, we do not run CCBS and ECBS-CT, but perform a direct comparison of the effectiveness of the time-range constraints which they use.

Table 1 shows results on the MAPF benchmarks (Stern et al. 2019) which consists of 25 tests on each of 28 grid-based maps of various types. Each test consists of up to 1,000 problem instances with increasing numbers of agents. Tests were run by incrementally adding one agent at a time until it becomes unsolvable within the allotted time limit of 30 seconds. The results for each experiment are the sum of the max number of agents solvable per each of the 25 trials. Top scores in each connectivity level of 8-, 16- and 32-connected are in bold.

With the exception of some DAO maps where ICTS is faster, CBS+TAB is the strongest overall algorithm in 8-connected grids, and about equally as strong as CBS+TMA in 16-connected grids. CBS+TMA is consistently stronger in 32-connected settings.

Table 2 shows the size of the CT from sample problems from each category in Table 1. The results are for a number of agents that were solvable by all algorithms in under 30 seconds. CBS+TMA and CBS+TAB show a significant reduction over prior approaches. When comparing the amount of node reduction to the values in Table 1, the improvement is generally less significant – this is due to the low-level performing extra work evaluating constraints. In the case of CBS+TAB and CBS+MVB, a large number of constraints are usually added per CT node. In the case of edge, vertex and time-range constraints, there is only one constraint added per CT node and these constraints are inex-

Code available at https://github.com/thaynewalker/hog2
More benchmarks at http://mapf.info

[1]For agents with diameter smaller than $^1/_2\sqrt{2}$

Table 3: Comparison of solution quality on 4- and 16-connected grids

Configuration	Optimal		Complete			
	CBS[4]	CBS[16]	GCBS[4]	GCBS[16]	GCBS+TCC	GCBS+TCP
Empty 8x8 (25 agents)	116	77 (67%)	132 (114%)	105 (91%)	107 (92%)	107 (92%)
Empty 64x64 (100 agents)	4,277	3,353 (78%)	4,283 (>100%)	3,355 (78%)	3,358 (79%)	3,358 (79%)
den520d (50 agents)	9,025	7,266 (81%)	9,028 (>100%)	7269 (81%)	7292 (81%)	7321 (81%)
brc202d (50 agents)	21,072	18,894 (90%)	21,090 (>100%)	18,899 (90%)	18,980 (90%)	18,922 (90%)
ost003d (50 agents)	7,889	6,148 (78%)	7,899 (>100%)	6,154 (78%)	6,293 (80%)	6,182 (78%)

Table 4: Total problems solved in under 30 seconds

Type	Map	GCBS+Time		GCBS+TCC		GCBS+TCP	
		8	32	8	32	8	32
City	Berlin_1_256	2,473	1,121	4,413	2,970	**4,920**	3,068
	Boston_0_256	3,027	1,073	**6,021**	2,937	5,879	2,983
	Paris_1_256	2,833	1,153	6,115	3,011	**6,745**	3,027
DAO	brc202d	1,701	733	2,703	1,709	**3,385**	2,035
	den312d	849	451	1,885	1,919	**2,549**	2,457
	den520d	1,653	737	2,911	2,783	3,161	**3,235**
	lak303d	1,035	435	2,289	1,883	**2,635**	2,495
	orz900d	1,507	509	2,163	963	**2,393**	1,059
	ost003d	1,139	493	2,341	2,167	2,645	**2,687**
Dragon Age 2	ht_chantry	1,221	577	2,635	2,381	**3,327**	3,217
	ht_mansion_n	1,251	565	2,391	2,515	**2,847**	2,809
	lt_gallowstemplar	1,325	653	2,223	2,213	2,493	**2,497**
	w_woundedcoast	2,031	735	3,277	1,726	**3,853**	2,873
Open	empty-8-8	392	221	800	800	800	800
	empty-16-16	423	223	1,695	1,751	**2,147**	2,121
	empty-32-32	839	431	2,991	3,061	**3,765**	3,737
	empty-48-48	1,079	535	4,043	4,683	5,271	**5,833**
Open+ obstacles	random-32-32-10	689	381	2,787	2,723	3,283	**3,389**
	random-32-32-20	765	401	2,175	1,991	**2,657**	2,435
	random-64-64-10	1,261	595	4,869	5,105	5,891	**6,051**
	random-64-64-20	1,135	681	3,743	3,581	4,105	**4,185**
Maze	maze-32-32-2	447	261	999	917	1,103	**1,123**
	maze-32-32-4	339	222	651	631	**745**	665
	maze-128-128-10	981	435	1,793	1,711	**2,295**	2,155
	maze-128-128-2	601	301	1,117	1,025	**1,223**	1,171
Room	room-32-32-4	489	256	1,205	1,135	**1,395**	1,349
	room-64-64-16	709	317	1,503	1,465	**1,909**	1,677
	room-64-64-8	419	230	1,055	1,005	**1,163**	1,092

pensive to evaluate. In the case of CBS+TMA, there is only one constraint per CT node, however, because the positive constraints perform a collision check when evaluated, they are more costly in terms of runtime. It is often the case that an MVB is a $1 \times N$ biclique (about 56% in 16-connected grids), thus, the set of blocked actions in CBS+TMA constraints is identical to CBS+TAB in many cases.

Results for Sub-Optimal Variants We compare state-of-the-art, Greedy CBS (GCBS) (Barer et al. 2014), an unbounded, suboptimal variant of CBS using the *number of conflicts* heuristic and time-range constraints with GCBS+TCC and GCBS+TCP which are GCBS with the new strategies discussed in Section 5. GCBS low-level prioritization on fewest conflicts with other agents is not performed because (as stated earlier) the CAT enhancement is not effective for 2^k neighborhoods with k of 8 and higher.

Table 4 shows results for the same set of benchmark problems. GCBS+TCP consistently outperforms the other variants. The improvement over GCBS is significant, up to 5x. Figure 5 shows success rate for a subset of the benchmark problems. GCBS+TCP is the strongest overall, with its most significant gains in maps with wide open spaces.

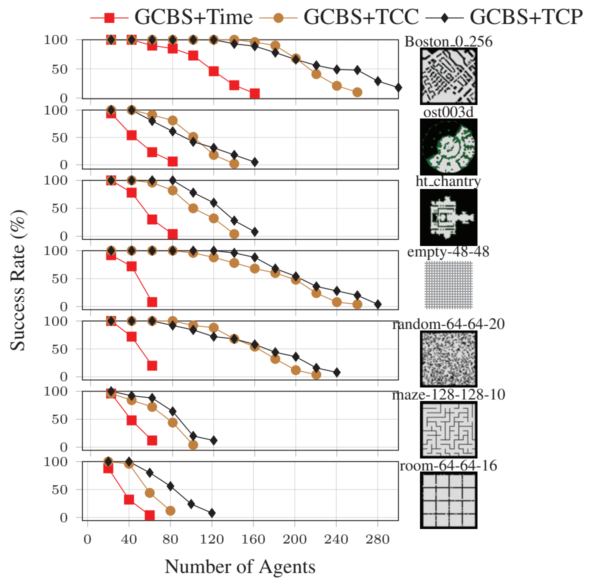

Figure 5: Success rate of sub-optimal variants

Table 3 shows mean solution costs where CBS[4], and GCBS[4] are for 4-connected grids, and CBS[16], and GCBS[16] are for 16-connected grids and GCBS+TCC and GCBS+TCP are also run on 16-connected grids. The solution quality compared to optimal costs in 4-connected grids (the underlined values) is shown next to each statistic as a percentage in parenthesis. Solutions in 8x8 grids show the highest percentages of sub-optimality. This is due to the high agent density. Both strategies do not significantly degrade the overall solution quality when compared to GCBS[16], usually 1% of optimality or less. GCBS+TCP, which shows a significant speedup over GCBS+TCC, does not show any significant degradation in solution quality.

Path quality in 16-connected grids is better than for 4-connected grids (Rivera, Hernández, and Baier 2017), and this phenomenon is reproduced here – CBS[16] consistently yields higher quality solutions than CBS[4], and all sub-optimal variants consistently report better solution quality than CBS[4]. This is a key highlight because it means that if sub-optimal results are acceptable, when given a choice between a low-fidelity, unit-cost movement model and a higher-fidelity non-unit cost movement model, a higher fidelity model can yield both higher quality solutions *and* better runtime performance by using sub-optimal variants.

7 Conclusions

This work introduced a new, systematic approach to implementing constraints using bipartite graphs. Constraints can be extended in both time and space by the use of time-annotated bicliques to significantly increase the efficiency of CBS. This work also formulated new conditional constraints which allow controlled deactivation of constraints in order to significantly increase the performance of Greedy-CBS while guaranteeing completeness.

Acknowlegements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and support from BSF grant #2017692, NSF grant #1815660 and Lockheed Martin Corp.

References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-agent pathfinding with continuous time. In *International Joint Conferences on Artifical Intelligence*, 39–45.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust multi-agent path finding. In *International Conference on Autonomous Agents and Multiagent Systems*, 1862–1864.

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-train path finding. In *Symposium on Combinatorial Search*, 125–129.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Symposium on Combinatorial Search*, 961–962.

Botea, A.; Bonusi, D.; and Surynek, P. 2018. Solving multi-agent path finding on strongly biconnected digraphs. *Journal of Artificial Intelligence Research* 273–314.

Boyarski, E.; Felner, A.; Sharon, G.; and Stern, R. 2015a. Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *International Conference on Planning and Scheduling*, 47–51.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015b. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conferences on Artifical Intelligence*, 223–225.

Chouhan, S. S., and Niyogi, R. 2015. Dmapp: A distributed multi-agent path planning algorithm. In *Australasian Joint Conference on Artificial Intelligence*, 123–135. Springer.

Chouhan, S. S., and Niyogi, R. 2017. Dimpp: a complete distributed algorithm for multi-agent path planning. *Journal of Experimental & Theoretical Artificial Intelligence* 1–20.

Cohen, L., and Koenig, S. 2016. Bounded suboptimal multi-agent path finding using highways. In *International Joint Conferences on Artifical Intelligence*, 3978–3979.

Cohen, L.; Greco, M.; Ma, H.; Hernandez, C.; Felner, A.; Koenig, S.; and Kumar, T. 2018. Anytime focal search with applications. In *International Joint Conferences on Artifical Intelligence*, 1434–1441.

Cohen, L.; Uras, T.; Kumar, T. K. S.; and Koenig, S. 2019. Optimal and bounded sub-optimal multi-agent motion planning. In *Symposium on Combinatorial Search*, 44–51.

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

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy cbs: Implict conflict-based search using lazy clause generation. In *International Conference on Planning and Scheduling*, 155–162.

Garey, M. R., and Johnson, D. S. 2002. *Computers and intractability*, volume 29. W.H. Freeman, New York.

Hönig, W.; Preiss, J. A.; Kumar, T. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics* 34(4):856–869.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Sven, K. 2019a. Symmetry-breaking constraints for grid-based multi-agent pathfinding. In *AAAI Conference on Artificial Intelligence*, 6087–6095.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; and Satish, K. T. 2019b. Multi-agent pathfinding for large agents. In *AAAI Conference on Artificial Intelligence*, 7627–7634.

Ma, H.; Wagner, G.; Felner, A.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2018. Multi-agent path finding with deadlines. In *International Joint Conferences on Artifical Intelligence*, 417–423.

Phillips, M., and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *International Conference on Robotics and Automation*, 5628–5635. IEEE.

Rivera, N.; Hernández, C.; and Baier, J. A. 2017. Grid pathfinding on the 2k neighborhoods. In *AAAI Conference on Artificial Intelligence*, 891–897.

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 *Artificial Intelligence and Interactive Digital Entertainment*, 117–122.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*, 28–29.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Kumar, T. K. S.; Boyarski, E.; and Barták, R. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Symposium on Combinatorial Search*, 151–159.

Thomas, S.; Deodhare, D.; and Murty, M. N. 2015. Extended conflict-based search for the convoy movement problem. *IEEE Intelligent Systems* 30(6):60–70.

Van Den Berg, J. P., and Overmars, M. H. 2005. Prioritized motion planning for multiple robots. In *International Conference on Intelligent Robots and Systems*, 430–435. IEEE.

Walker, T. T., and Sturtevant, N. R. 2019. Collision detection for agents in multi-agent pathfinding. *arXiv preprint arXiv:1908.09707*.

Walker, T. T.; Chan, D.; and Sturtevant, N. R. 2017. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *International Conference on Planning and Scheduling*, 316–324.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2018. Extended increasing cost tree search for non-unit cost domains. In *International Joint Conferences on Artifical Intelligence*, 534–540.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI Conference on Artificial Intelligence*, 1443–1449.