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

Thayne T. Walker¹, Nathan R. Sturtevant², Ariel Felner³
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 connected grids over the previous state-of-the-art for both optimal and suboptimal variants.

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 suboptimal variants in 2ⁿ-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 \( (G, A, V_s, V_g, t) \) where \( G = (V, E) \) 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. \( V_s \subseteq V \) is composed of unique start and goal states. \( V_g = \{goal_1, ..., goal_k\} \) 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. Each path \( \pi_i = (s_0^{i}, ..., s_{d_i}^{i}) \) where \( s_0^i = (start_i, 0) \) and \( s_{d_i}^i = (goal_i, t) \). Each \( (v(s_0^i), v(s_{d_i}^{i+1})) \in E \) and each \( t(s_{d_i}^{i+1}) = t(s_d^i) + w(e^i_d) \). 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 \( (v(s_n^i), v(s_{n+1}^i)) \in E \) — an agent begins with its reference point at \( v(s_n^i) \) at time \( t(s_n^i) \) and its reference point follows a straight, constant velocity motion vector in metric space, ending at \( v(s_{n+1}^i) \) at time \( t(s_{n+1}^i) \). 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 \( \pi_i = [a_0^i, ..., a_d_i^i] \) where each
The high level CBS (Sharon et al. 2015) is a two-level algorithm for MAPF. As the classic version is a special case of 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, MAPF_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 \( N.II \). Each \( \pi \in N.II \) for the root node is constructed using a low level search without taking other agents into account. \( N.II \) is checked for conflicts between paths. If no conflict is found, \( N \) is a goal and CBS terminals. 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 \( \pi_i \) and \( \pi_j \) 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_R.

Several optimal enhancements for CBS in the classic problem have been published, (Boyariski 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_R Optimal and complete solvers were published for MAPF_R. The ICTS algorithm was extended for MAPF_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_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 the 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 \( <a_i, a_j> \) – 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 II, 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, MAPF_R is also NP-hard.

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_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...
number of vertices. This can be done in polynomial time in biclique order to maximize pruning in the CT, we find a vertex that is, \( E \) consists of the subset of vertex pairs \( (u, v) \) with computed unsafe intervals to create a time-annotated biclique \( G' \). For each \( e \in E' \), \( E'_t = E'_t(\cup e.\text{UNSAFEINTERVAL}(e.\text{u}, e.\text{v})) \). Annotate all vertices \( U'_t, V'_t \) with the intersection of all unsafe intervals of incident edges: For each \( u \in U'_t, v \leftarrow (u, \bigcup_{e \in \text{INCIDENT}(u)} e.\text{intvl}) \); analogously for \( V'_t \). return \((U'_t, V'_t)\).

**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 \( \pi_i^*, \pi_j^* \) from a feasible solution \( \Pi^* \) 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 \( (a_i, a_j) \) that caused a split. In Figure 1(a), the core action pair is \((3, 8)\), 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 (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 \( tr_i = \left[ t^\text{start}_i, t^\text{end}_i \right] \) fully includes another interval \( tr_j = \left[ t^\text{start}_j, t^\text{end}_j \right] \) if \( t^\text{start}_i \leq t^\text{start}_j \) and \( t^\text{end}_i \geq t^\text{end}_j \). 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

**Algorithm 1 ComputeLargestTimeAnnotatedBiclique**

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

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 \( (u, tr = [t_{start}, t_{end}]) \) for each \( u \in U' \) where \( [t_{start}, t_{end}] \) is the unsafe interval associated with it. This is then done analogously for the right node using \( V' \). 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 Theorem 4.2. CBS+TAB is complete.

First, the action sets \( U', V' \) are regarded for now. CBS+TAB computes a TAB and creates enhancements discussed in the next section and can be dis-

Black portions of Algorithm 2 are the core algorithm for the CBS+TAB split operation. Red portions are sub-optimal enhancements discussed in the next section and can be dis-regarded 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, instead of explicitly blocking each action in \( U' \) and \( V' \), the TAB is represented implicitly, using only two constraints, one edge constraint \( c_i \) for agent \( i \) that blocks \( a_i \) (this is \( U' \)), and another constraint \( c_j \) for agent \( j \) that blocks all actions that conflict with \( a_i \) (this is \( V' \)). 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 dis-junctive 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.

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 un-blocked 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: \( (a_i, a_j) \leftarrow \text{find-conflict}(N, II) \)
3: if No conflict return \( N, II \) 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: \( (U, V, E) \leftarrow \text{CreateBCG}(a_i, a_j) \)
9: \( (U', V') \leftarrow \text{ComputeMaxVertexTAB}(U, V, E) \)
10: // Create constraints
11: \( N_i.C \leftarrow N_i.C \cup \text{CreatePermanentConstraints}(U') \)
12: \( N_j.C \leftarrow N_j.C \cup \text{CreatePermanentConstraints}(V') \)
13: \( N_i.C \leftarrow N_i.C \cup \text{CreateConditionalConstraints}(U \setminus U') \)
14: \( N_j.C \leftarrow N_j.C \cup \text{CreateConditionalConstraints}(V \setminus V') \)
15: // Create probabilistically filtered sets
16: \( \rho_i \leftarrow \min((\Delta_i - 1)/d_i, 1.0); \rho_j \leftarrow \min(e)((\Delta_j - 1)/d_j, 1.0) \)
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: \( N_i, II \leftarrow \text{Replan}(start_i, goal_i, N_i.C) \)
21: \( N_j, II \leftarrow \text{Replan}(start_j, goal_j, N_j.C) \)
22: // Check for no path and re-plan without conditional constraints
23: if \( N_i, II, \pi_i = \emptyset \) then
24: Remove all conditional constraints from \( N_i.C \)
25: \( N_i, II \leftarrow \text{Replan}(N_i.C, \pi_i, N_i.C) \)
26: end if
27: if \( N_j, II, \pi_j = \emptyset \) then
28: Remove all conditional constraints from \( N_j.C \)
29: \( N_j, II \leftarrow \text{Replan}(N_j.C, \pi_j, N_j.C) \)
30: end if
31: Add \( N_i, N_j \) to OPEN
Enhancements for implementing conditional constraints are highlighted in red 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 \( U \setminus U'_i \) and \( V \setminus V'_i \) where \( U \) and \( V \) are from the BCG, and \( U'_i \) and \( V'_i \) 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 \( \rho_{\text{off}} = \min(1, (\Delta_i - d_i)/d_i) \) (line 16) where \( d_i \) (line 6) is the length of the path for agent \( i \) in the root CT node and \( \Delta_i \) (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, \( \Delta_i \) will grow relative to \( d_i \), increasing \( \rho_{\text{off}} \), 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 \( \Delta_i \). This situation may be repeated in subsequent CT nodes with increasing \( \rho_{\text{off}} \) until \( c \) is turned off. Because \( \Delta_i \) is monotonically increasing, \( \rho_{\text{off}} \) 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 \( N.C \). 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.

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 \( \Pi^* \) 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 \( 1/2\sqrt{\pi} \) which disappear upon reaching their goal.
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 (Boyar et al. 2015a) and the conflict prioritization enhancement (Boyar et al. 2015b) were either ineffective or detrimental in 28 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.

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 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-constraints (Atzmon et al. 2018) – based on CCBS (Andreyevchuk 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 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-
Table 4: Total problems solved in under 30 seconds

<table>
<thead>
<tr>
<th>Type</th>
<th>Map</th>
<th>GCBS+Time 8</th>
<th>GCBS+Time 32</th>
<th>GCBS+TCC 8</th>
<th>GCBS+TCC 32</th>
<th>GCBS+TCP 8</th>
<th>GCBS+TCP 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>Berlin_1,256</td>
<td>2,473</td>
<td>1,121</td>
<td>4,413</td>
<td>2,970</td>
<td>4,920</td>
<td>3,088</td>
</tr>
<tr>
<td></td>
<td>Boston_0,256</td>
<td>1,027</td>
<td>1,073</td>
<td>6,021</td>
<td>2,937</td>
<td>5,879</td>
<td>2,983</td>
</tr>
<tr>
<td></td>
<td>Paris_1,256</td>
<td>2,833</td>
<td>1,153</td>
<td>6,115</td>
<td>3,011</td>
<td>6,745</td>
<td>3,027</td>
</tr>
<tr>
<td>DAO</td>
<td>br202d</td>
<td>1,701</td>
<td>733</td>
<td>2,703</td>
<td>1,799</td>
<td>3,385</td>
<td>2,035</td>
</tr>
<tr>
<td></td>
<td>den312d</td>
<td>849</td>
<td>451</td>
<td>1,885</td>
<td>1,919</td>
<td>2,549</td>
<td>2,457</td>
</tr>
<tr>
<td></td>
<td>den520d</td>
<td>1,653</td>
<td>737</td>
<td>2,911</td>
<td>2,783</td>
<td>3,161</td>
<td>3,235</td>
</tr>
<tr>
<td></td>
<td>lak303d</td>
<td>1,035</td>
<td>435</td>
<td>2,289</td>
<td>1,883</td>
<td>2,635</td>
<td>2,495</td>
</tr>
<tr>
<td></td>
<td>orr900d</td>
<td>1,507</td>
<td>509</td>
<td>2,163</td>
<td>963</td>
<td>2,393</td>
<td>1,059</td>
</tr>
<tr>
<td></td>
<td>out003d</td>
<td>1,139</td>
<td>493</td>
<td>2,341</td>
<td>2,167</td>
<td>2,645</td>
<td>2,687</td>
</tr>
<tr>
<td>Dragon Age 2</td>
<td>ht_chantry</td>
<td>1,221</td>
<td>577</td>
<td>2,635</td>
<td>2,381</td>
<td>3,327</td>
<td>3,217</td>
</tr>
<tr>
<td></td>
<td>ht_mansion_n</td>
<td>1,251</td>
<td>565</td>
<td>2,391</td>
<td>2,515</td>
<td>2,847</td>
<td>2,809</td>
</tr>
<tr>
<td></td>
<td>lt_lowestemple</td>
<td>1,325</td>
<td>653</td>
<td>2,223</td>
<td>2,213</td>
<td>2,493</td>
<td>2,497</td>
</tr>
<tr>
<td></td>
<td>w_woundedcoast</td>
<td>2,031</td>
<td>735</td>
<td>3,277</td>
<td>1,726</td>
<td>3,853</td>
<td>2,873</td>
</tr>
<tr>
<td>Open</td>
<td>empty-8-8</td>
<td>392</td>
<td>221</td>
<td>800</td>
<td>800</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>empty-16-16</td>
<td>423</td>
<td>223</td>
<td>1,695</td>
<td>1,751</td>
<td>2,147</td>
<td>2,121</td>
</tr>
<tr>
<td></td>
<td>empty-32-32</td>
<td>839</td>
<td>431</td>
<td>2,991</td>
<td>3,061</td>
<td>3,765</td>
<td>3,737</td>
</tr>
<tr>
<td></td>
<td>empty-48-48</td>
<td>1,079</td>
<td>535</td>
<td>4,043</td>
<td>4,683</td>
<td>5,271</td>
<td>5,833</td>
</tr>
<tr>
<td>Open+obstacles</td>
<td>random-32-32-10</td>
<td>689</td>
<td>381</td>
<td>2,787</td>
<td>2,723</td>
<td>3,283</td>
<td>3,389</td>
</tr>
<tr>
<td></td>
<td>random-32-32-20</td>
<td>765</td>
<td>401</td>
<td>2,175</td>
<td>1,991</td>
<td>2,657</td>
<td>2,435</td>
</tr>
<tr>
<td></td>
<td>random-64-64-10</td>
<td>1,261</td>
<td>595</td>
<td>4,869</td>
<td>5,105</td>
<td>5,891</td>
<td>6,051</td>
</tr>
<tr>
<td></td>
<td>random-64-64-20</td>
<td>1,135</td>
<td>681</td>
<td>3,743</td>
<td>3,581</td>
<td>4,105</td>
<td>4,185</td>
</tr>
<tr>
<td>Maze</td>
<td>maze-32-32-2</td>
<td>447</td>
<td>261</td>
<td>999</td>
<td>917</td>
<td>1,103</td>
<td>1,123</td>
</tr>
<tr>
<td></td>
<td>maze-32-32-4</td>
<td>339</td>
<td>222</td>
<td>651</td>
<td>631</td>
<td>745</td>
<td>665</td>
</tr>
<tr>
<td></td>
<td>maze-128-128-10</td>
<td>981</td>
<td>435</td>
<td>1,793</td>
<td>1,711</td>
<td>2,295</td>
<td>2,155</td>
</tr>
<tr>
<td></td>
<td>maze-128-128-2</td>
<td>601</td>
<td>301</td>
<td>1,117</td>
<td>1,025</td>
<td>1,223</td>
<td>1,171</td>
</tr>
<tr>
<td>Room</td>
<td>room-32-32-4</td>
<td>489</td>
<td>256</td>
<td>1,205</td>
<td>1,135</td>
<td>1,395</td>
<td>1,349</td>
</tr>
<tr>
<td></td>
<td>room-64-64-16</td>
<td>709</td>
<td>317</td>
<td>1,503</td>
<td>1,465</td>
<td>1,909</td>
<td>1,677</td>
</tr>
<tr>
<td></td>
<td>room-64-64-8</td>
<td>419</td>
<td>230</td>
<td>1,055</td>
<td>1,005</td>
<td>1,163</td>
<td>1,092</td>
</tr>
</tbody>
</table>

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 fi-
delity 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.

Acknowledgements
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


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.


