Improved Heuristics for Multi-Agent Pathfinding with Conflict-Based Search*

Jiaoyang Li,¹ Ariel Felner,² Eli Boyarski,² Sven Koenig,¹ Hang Ma,¹
¹USC, ²Ben Gurion University of the Negev
jiaoyanl@usc.edu, felner@bgu.ac.il, boyarske@post.bgu.ac.il, skoenig@usc.edu, hangma@usc.edu

Abstract
Conflict-Based Search (CBS) and its enhancements are among the strongest algorithms for Multi-Agent Pathfinding. Recent work introduced an admissible heuristic to guide the high-level search of CBS. In this work, we prove the limitation of this heuristic, as it is based on cardinal conflicts only. We then introduce two new admissible heuristics by reasoning about the pairwise dependency between agents. Empirically, CBS with both new heuristics significantly improves the success rate over CBS with the recent heuristic and reduces the number of expanded nodes and runtime by up to a factor of 50, yielding a new state-of-the-art CBS-based algorithm.

1 Introduction
Multi-Agent Pathfinding (MAPF) is the problem of finding a set of collision-free paths for a given set of agents on a given graph. Although MAPF is NP-hard to solve optimally [Yu and LaValle, 2013a], many MAPF algorithms have been developed in recent years, including reduction-based algorithms [Yu and LaValle, 2013b; Erdem et al., 2016], A*-based algorithms [Standley, 2010; Wagner and Choset, 2011; Goldenberg et al., 2014] and dedicated search-based algorithms [Sharon et al., 2013; Sharon et al., 2015]. Felner et al. [2017] and Ma and Koenig [2017] provide surveys of algorithms and applications for MAPF.

Conflict-Based Search (CBS) [Sharon et al., 2015] is a popular search-based MAPF algorithm which resolves collisions by adding constraints at a high level and computing solutions consistent with those constraints at a low level. It is widely used in many applications, such as warehouse robots [Ma et al., 2017a; Hoenig et al., 2019], quadrotor swarms [Hoenig et al., 2018] and computer game characters [Ma et al., 2017b].

A number of enhancements to CBS have been introduced [Barer et al., 2014; Boyarski et al., 2015; Cohen et al., 2016; Cohen et al., 2018; Li et al., 2019a]. CBSH [Felner et al., 2018] was the first work that introduced an admissible heuristic (called here CG) for the high-level search of CBS by reasoning about cardinal conflicts in the current solution. In this paper, we further develop this direction. We first prove that CG can only offer limited information. We then introduce two new admissible heuristics, DG and WDG, by considering potential conflicts in the future solutions and reasoning about the pairwise dependency between agents. WDG strictly dominates DG, which in turn strictly dominates CG in their h-values. Empirically, the runtime overhead of the new heuristics is reasonable, and WDG improves the success rate of CBS significantly compared to CBS with CG and reduces the number of expanded nodes and runtime by up to a factor of 50, yielding a new state-of-the-art CBS-based algorithm.

2 Background
2.1 Problem Definition
The Multi-Agent Path-Finding (MAPF) problem is specified by an undirected unweighted graph $G = (V,E)$ and a set of $k$ agents $\{a_1 \ldots a_k\}$, where $a_i$ has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. Time is discretized into timesteps. Between successive timesteps, every agent can either move to an adjacent vertex or wait at its current vertex. Both move and wait actions have unit cost unless the agent terminally waits at its goal vertex, which has zero cost. A path of $a_i$ is a sequence of move and wait actions that lead $a_i$ from $s_i$ to $g_i$. A tuple $(a_i, a_j, v, t)$ is a vertex conflict if $a_i$ and $a_j$ are at the same vertex $v$ at timestep $t$, and a tuple $(a_i, a_j, u, v, t)$ is an edge conflict if $a_i$ and $a_j$ traverse the same edge $(u, v)$ in opposite directions between timesteps $t$ and $t+1$. The objective that we focus on in this paper is to find a set of conflict-free paths which move all agents from their start vertices to their goal vertices while minimizing the sum of the costs of these paths.

2.2 Conflict-Based Search (CBS)
CBS has two levels. The high level of CBS searches the binary constraint tree (CT) in a best-first manner according to the costs of the CT nodes. Each CT node $N$ contains: (1) a set of constraints $N.constraints$ where a constraint is either a vertex constraint $(a_i, v, t)$ that prohibits $a_i$ from being at vertex $v$ at timestep $t$ or an edge constraint $(a_i, u, v, t)$ that prohibits $a_i$ from moving from vertex $u$ to vertex $v$ between
conflicts (by default, arbitrarily) and resolves it by splitting and returns. CBS arbitrarily chooses conflicts to split on. However, poor nodes per conflict, CBS guarantees optimality by exploring have any solutions and therefore is pruned. With two child CT constraints (very rare in practice), this child CT node does not If the low-level search cannot find any paths that satisfy the cision Diagram (MDD) [Sharon addresses this issue by prioritizing conflicts at a CT node N into two child CT nodes. In each child CT node, one agent classifies conflicts into three types. A conflict is cardinal iff, when CBS uses it to split N, the cost of each of the two resulting child CT nodes is larger than N.cost. It is semi-cardinal iff the cost of one child CT node is larger than N.cost, but the cost of the other child CT node is equal to N.cost. Finally, it is non-cardinal iff the cost of each of the two child CT nodes is equal to N.cost. CBS must first choose a cardinal conflict (if one exists) when splitting N. For example, in Figure 1(left), the conflict (a1, a2, B2, 1) at the CT root node is non-cardinal as both agents have bypasses that reach their goal vertices at timestep 4 without being at B2 at timestep 1. However, if cells C1 and A3 are blocked, the conflict becomes cardinal. This is because when one of a1 and a2 is prohibited from being at B2 at timestep 1, it has to wait at its start vertex for 1 timestep and thus reaches its goal vertex at timestep 5.

ICBS uses MDDs to classify conflicts. A Multi-Valued Decision Diagram (MDD) [Sharon et al., 2013] MDD\(i\) for a\(i\) at N is a directed acyclic graph that consists of all paths of a\(i\) from \(s_i\) to \(g_i\) within \(\mu\) timesteps that satisfy N.constraints. Nodes at depth \(t\) of MDD\(i\) correspond to all vertices where a\(i\) can be at timestep t along such a path. Here, \(\mu\) should be no smaller than the minimum path of a\(i\) that satisfies N.constraints, \(\mu_i\), otherwise MDD\(i\) is empty. A conflict between a\(i\) and a\(j\) at timestep t is cardinal iff the contested vertex or edge is the only vertex or edge at level t of both MDD\(i\) and MDD\(j\). Figure 1(middle) shows the MDDs MDD\(1\) and MDD\(2\) for a\(1\) and a\(2\) at the CT root node, respectively. Since both MDDs have 2 nodes at timestep 1, the conflict (a\(1\), a\(2\), B2, 1) is non-cardinal.

2.3 Improved CBS (ICBS)

CBS arbitrarily chooses conflicts to split on. However, poor choices can substantially increase the size of its CT and thus its runtime. Improved CBS (ICBS) [Boyarski et al., 2015] addresses this issue by prioritizing conflicts at a CT node N. It classifies conflicts into three types. A conflict is cardinal iff, when CBS uses it to split N, the cost of each of the two resulting child CT nodes is larger than N.cost. It is semi-cardinal iff the cost of one child CT node is larger than N.cost, but the cost of the other child CT node is equal to N.cost. Finally, it is non-cardinal iff the cost of each of the two child CT nodes is equal to N.cost. ICBS must first choose a cardinal conflict (if one exists) when splitting N. For example, in Figure 1(left), the conflict (a1, a2, B2, 1) at the CT root node is non-cardinal as both agents have bypasses that reach their goal vertices at timestep 4 without being at B2 at timestep 1. However, if cells C1 and A3 are blocked, the conflict becomes cardinal. This is because when one of a1 and a2 is prohibited from being at B2 at timestep 1, it has to wait at its start vertex for 1 timestep and thus reaches its goal vertex at timestep 5.

ICBS uses MDDs to classify conflicts. A Multi-Valued Decision Diagram (MDD) [Sharon et al., 2013] MDD\(i\) for a\(i\) at N is a directed acyclic graph that consists of all paths of a\(i\) from \(s_i\) to \(g_i\) within \(\mu\) timesteps that satisfy N.constraints. Nodes at depth \(t\) of MDD\(i\) correspond to all vertices where a\(i\) can be at timestep t along such a path. Here, \(\mu\) should be no smaller than the minimum path of a\(i\) that satisfies N.constraints, \(\mu_i\), otherwise MDD\(i\) is empty. A conflict between a\(i\) and a\(j\) at timestep t is cardinal iff the contested vertex or edge is the only vertex or edge at level t of both MDD\(i\) and MDD\(j\). Figure 1(middle) shows the MDDs MDD\(1\) and MDD\(2\) for a\(1\) and a\(2\) at the CT root node, respectively. Since both MDDs have 2 nodes at timestep 1, the conflict (a\(1\), a\(2\), B2, 1) is non-cardinal.

2.4 CBSH

The high level of CBS always chooses to expand the CT node N with the smallest N.cost. CBSH [Felner et al., 2018] speeds up the high-level search through the addition of an admissible heuristic. The idea is simple: If N.solution contains one cardinal conflict (identified by MDDs), then an h-value of 1 is admissible for N because the cost of any of its descendant CT nodes with a conflict-free solution is at least N.cost + 1. If N.solution contains multiple cardinal conflicts, CBSH builds a conflict graph, whose vertices represent agents and edges represent cardinal conflicts in N.solution. The cost of the path of at least one agent from each cardinal conflict has to increase by at least 1. Thus, the size of a minimum vertex cover (MVC) of the conflict graph (i.e., a set of vertices such that each edge is incident on at least one vertex in the set) is an admissible h-value for N. We refer to this heuristic as the CG heuristic.

3 Limitation of The CG Heuristic

We have just seen that an h-value of 1 for a cardinal conflict is admissible. However, is it possible that a larger h-value for a cardinal conflict is also admissible? In this section, we study some theoretical properties of CG and show its limitation.

All MDDs we discuss in this section are MDDs ignoring all constraints, i.e., they are allowed to include nodes prohibited by constraints. We say a node (x, t) ∈ MDD\(i\) iff MDD\(i\) has vertex x at level t, i.e., there is a path from s\(i\) at timestep 0 to x at timestep t and then to g\(i\) at timestep \(\mu\). We say an MDD MDD\(i\) includes all nodes prohibited by a set of constraints C iff for all vertex constraints (a\(i\), v, t) ∈ C, (v, t) ∈ MDD\(i\), and for all edge constraints (a\(i\), u, v, t) ∈ C, (u, t) ∈ MDD\(i\) and (v, t + 1) ∈ MDD\(i\) with an edge between them.

Let dist(x,y) denote the distance between vertices x and y and P(t) denote the vertex at timestep t on path P. Then MDDs have the following properties:

1. \((x, t) ∈ MDD\(i\) if \(dist(s_i, x) ≤ t\) and \(dist(x, g_i) ≤ μ - t\).
2. \((x, t) ∉ MDD\(i\) if \(dist(s_i, x) ≥ t + 1\) or \(dist(x, g_i) ≥ μ + t + 1\).
3. For any path P of a\(i\), if \((P(t), t) ∉ MDD\(i\), \(P(t′), t′) ∉ MDD\(i\) for all \(t′ ≥ t\).

Property (1) holds because there is a path P from s\(i\) at timestep 0 to x at timestep t and then to g\(i\) at timestep \(μ\) iff \(dist(s_i, x)\) is no larger than \(t\) (the cost of P from s\(i\) to x) and \(dist(x, g_i)\) is no larger than \(μ - t\) (the cost of P from x to g\(i\)). Property (2) is the contrary proposition of Property (1). Property (3) holds because, by contradiction, if \((P(t), t) ∈ MDD\(i\), there is a sub-path P′ in MDD\(i\) from P′ at timestep t to g\(i\) at timestep \(μ\). The path that follows a prefix of P from s\(i\) at timestep 0 to P′ at timestep \(t′\) and then follows P′ to g\(i\) at timestep \(μ\) traverses P at timestep \(t′\), i.e., \((P(t), t) ∉ MDD\(i\). Property (3) tells us that, once a path leaves a MDD, it will never revisit this MDD later.

Lemma 1. Let C be a set of constraints and MDD\(i\) be an MDD that includes all nodes prohibited by C. If a\(i\) has a path
that satisfies all constraints in $C$, then $a_i$ has a path of cost no more than $\mu + 1$ that satisfies $C$.

**Proof.** By assumption, $a_i$ has a path $P$ of cost $T$ that satisfies $C$. If $T \leq \mu + 1$, then we are done. Otherwise, because $(s_i,0) \in MDD_\mu^i$ and $(g_i,T) \notin MDD_\mu^i$, there is a timestep $t < T$ such that $(P(t),t) \in MDD_\mu^i$ and $(P(t+1),t+1) \notin MDD_\mu^i$. Let $x = P(t)$ and $y = P(t+1)$ (Figure 2(left)).

There are now two cases.

**Case 1:** If $(y,t+1) \in MDD_{\mu+1}^i$ (Figure 2(middle)), there is a sub-path $P'$ in $MDD_{\mu+1}^i$ from $y$ at timestep $t+1$ to $g_i$ at timestep $\mu + 1$. From Property (3), $P'$ does not traverse any MDD node in $MDD_\mu^i$, and thus does not violate any constraints. Therefore, a path that follows a prefix of $P$ from $s_i$ at timestep 0 to $y$ at timestep $t+1$ and then follows $P'$ to $g_i$ at timestep $\mu + 1$ is a path of cost $\mu + 1$ that satisfies $C$.

**Case 2:** If $(y,t+1) \notin MDD_{\mu+1}^i$ (Figure 2(right)), then

\[
\begin{align*}
\mu + 2 \leq t + 1 + \text{dist}(y,g_i) & \quad (1) \\
\mu + 2 \leq t + \text{dist}(y,x) + \text{dist}(x,g_i) & \quad (2) \\
\mu + 1 + \text{dist}(y,x) + \text{dist}(x,g_i) & \leq t + \text{dist}(x,g_i) + 2 \leq \mu + 2. & \quad (3) \\
\mu + 1 + \text{dist}(y,x) + \text{dist}(x,g_i) & \leq t \leq \mu. & \quad (4)
\end{align*}
\]

Inequality (1) is obtained by applying $(y,t+1) \notin MDD_{\mu+1}^i$ to Property (2). Inequality (2) is based on the triangle inequality. Inequality (3) is from $\text{dist}(y,x) \leq 1$. Inequality (4) is obtained by applying $(x,t) \in MDD_\mu^i$ to Property (1). By comparing the first line and the last line of these inequalities, all lines are actually equal. Specifically, for Inequality (4), $t + \text{dist}(x,g_i) = \mu$. So, from $(x,t) \in MDD_\mu^i$ and Properties (1) and (2), we know $(x,t+1) \notin MDD_\mu^i$ and $(x,t+1) \in MDD_{\mu+1}^i$. Hence, there is a sub-path $P''$ in $MDD_{\mu+1}^i$ from $x$ at timestep $t+1$ to $g_i$ at timestep $\mu + 1$. From Property (3), $P''$ does not traverse any MDD node in $MDD_\mu^i$, and thus does not violate any constraints. Therefore, a path that follows a prefix of $P$ from $s_i$ at timestep 0 to $x$ at timestep $t$, waits for one timestep, and then follows $P''$ to $g_i$ at timestep $\mu + 1$ is a path of cost $\mu + 1$ that satisfies $C$. \qed

Therefore, if both child CT nodes of $N$ have solutions, a conflict can be regarded as an admissible $h$-value of at most 1. Since CT nodes always have solutions in practice, the size of the MVC is the best admissible heuristic for $N$ that can be obtained from the conflict graph. So, if we want to obtain better heuristics, new directions should be explored.

### 4 The DG Heuristic

The CG heuristic only considers cardinal conflicts in $N\text{.solution}$. To improve on that we need to also consider conflicts in future solutions, i.e., solutions of $N$‘s descendant CT nodes. For example, in Figure 1(left), if CBS resolves the non-cardinal conflict $(a_1,a_2,B2,1)$ by adding a constraint to one of the agents, a new conflict will occur no matter what new cost-minimal path the agent picks. In fact, any two individual cost-minimal paths of the two agents conflict in one of the 4 cells in the middle (B2.B3.C2.C3). Therefore, an $h$-value of 1 is admissible here. This is not captured by CG because the conflicts are initially non-cardinal. Inspired by this example, we generalize the conflict graph described above to a pairwise dependency graph whose edges between two vertices reflect that all individual cost-minimal paths of the corresponding two agents have conflicts.

#### 4.1 Pairwise Dependency Graph $G_D$

Formally, we define a pairwise dependency graph $G_D = (V_D,E_D)$ for each CT node $N$. Each agent $a_i$ induces a vertex $v_i \in V_D$. An edge $(v_i,v_j) \in E_D$ if $a_i$ and $a_j$ are dependent, i.e., all their individual cost-minimal paths that satisfy $N\text{.constraints}$ have conflicts. Similarly to the conflict graph, for each edge $(v_i,v_j) \in E_D$, the cost of the path of at least one agent, $a_i$ or $a_j$, has to increase by at least 1. Hence, the size of the MVC of $G_D$ is an admissible $h$-value for $N$. We refer to this heuristic as the DG heuristic. DG strictly dominates CG because the conflict graph is a sub-graph of $G_D$. We use the same algorithm in [Feln et al., 2018] to solve MVC optimally, whose complexity is $O(2^{|V_D|})$ where $q$ is the size of the MVC.
between every pair of agents. Let $\mu_i$ denote the cost of the path of $a_i$ in $N\.solution$. We first classify all pairs of agents into three categories based on conflicts in $N\.solution$: (1) The two agents do not have any conflicts; (2) They have at least one cardinal conflict; (3) They have only semi- or non-cardinal conflicts. If $a_i$ and $a_j$ are in Category (1), they are independent as their paths in $N\.solution$ are conflict-free. Hence $(v_i, v_j) \notin E_D$. If they are in Category (2), by the definition of cardinal conflicts, they are surely dependent. Hence $(v_i, v_j) \in E_D$. If they are in Category (3), we do not know whether they are dependent or independent. To answer this, we try to merge $MDD_i^\mu$ and $MDD_j^\mu$ into a joint MDD using the method described in [Sharon et al., 2013].

4.2 Constructing $G_D$

To construct $G_D$ for $N$, we need to analyze the dependency between every pair of agents. Let $\mu_i$ denote the cost of the path of $a_i$ in $N\.solution$. We first classify all pairs of agents into three categories based on conflicts in $N\.solution$: (1) The two agents do not have any conflicts; (2) They have at least one cardinal conflict; (3) They have only semi- or non-cardinal conflicts. If $a_i$ and $a_j$ are in Category (1), they are independent as their paths in $N\.solution$ are conflict-free. Hence $(v_i, v_j) \notin E_D$. If they are in Category (2), by the definition of cardinal conflicts, they are surely dependent. Hence $(v_i, v_j) \in E_D$. If they are in Category (3), we do not know whether they are dependent or independent. To answer this, we try to merge $MDD_i^\mu$ and $MDD_j^\mu$ into a joint MDD using the method described in [Sharon et al., 2013].

4.3 Merging The MDDs

The joint MDD of $MDD_i^\mu$ and $MDD_j^\mu$ consists of all combinations of cost-minimal conflict-free paths of $a_i$ and $a_j$ that satisfy $N\.constraints$. Nodes at depth $t$ of the joint MDD correspond to all joint states where $a_i$ and $a_j$ can be at timestep $t$ along such a pair of paths. The merging procedure starts at the joint state $(s_i, s_j)$ at level 0 and is built level by level. Suppose that we already have a joint state $(v_i, v_j)$ at level $t$ and want to add its child nodes at level $t+1$. Each pair in the cross product of the child nodes of $v_i$ at level $t$ in $MDD_i^\mu$ and the child nodes of $v_j$ at level $t$ in $MDD_j^\mu$ should be examined. Only conflict-free pairs are added. $a_i$ and $a_j$ are dependent (i.e., $(v_i, v_j) \in E_D$) iff the joint MDD is empty, i.e., it does not contain state $(g_i, g_j)$ at level $\max\\{\mu_i, \mu_j\}$.

Figure 1(right) shows an example of merging the MDDs. The joint MDD starts from $(B1, A2)$ at level 0. At level 1, we try all combinations of vertices at level 1 in both MDDs and add all of them to the joint MDD except the pair $(B2, B2)$ which represents a conflict (as both agents cannot be at B2 at the same time). We repeat this procedure at levels 2 and 3 until all branches of the joint MDD reach conflicting states and cannot be further developed. Therefore, in this example, the joint MDD is empty.

Since each CT node has only one additional constraint imposed on one agent, we only need to look at the dependency between this agent and all other agents and directly copy edges for other pairs of agents from $G_D$ for the parent CT node. Of course, at the CT root node, we still need to look at the dependency between all pairs of agents. Notice that $G_D$ already built MDDs to classify conflicts. So, for DG, we get $MDD_i^\mu$ and $MDD_j^\mu$ for free. The only extra overhead of DG over CG comes from merging the MDDs.

5 The WDG Heuristic

Following the notation from [Sharon et al., 2015], for a CT node $N$ and a subset of agents from $\{a_1, \ldots, a_k\}$, we refer to the sum of the minimum costs of their individual paths (ignoring other agents) that satisfy $N\.constraints$ as their SIC and the minimum sum of the costs of their conflict-free paths that satisfy $N\.constraints$ as their SOC. Let $\Delta = SOC - SIC \geq 0$. For all $k$ agents, $N\.cost$ is their SIC, and $N\.solution$ is conflict-free iff their $\Delta = 0$. For any two agents, they are dependent iff their $\Delta > 0$.

Although $G_D$ captures the information about whether or not $\Delta > 0$ for any pair of agents, it does not capture the information about how large the value of $\Delta$ is. Thus, when $\Delta > 0$, the DG heuristic only uses 1 (which is a lower bound of $\Delta$) as an admissible heuristic. However, in some cases, the $\Delta$ for a pair of agents could be larger than 1. For instance, in Figure 3, $\Delta = 4$ because one of the agents must wait 4 timesteps at its start vertex. Therefore, we introduce the WDG heuristic, which is interested in not only the pairwise dependency between agents but also the cost that each pair of dependent agents can contribute to the total cost.

5.1 Weighted Pairwise Dependency Graph $G_{WD}$

We generalize the pairwise dependency graph to a weighted pairwise dependency graph $G_{WD} = (V_D, E_D, W_D)$ for $N$. It uses the same vertices and edges as in $G_D$. Every edge $(v_i, v_j) \in E_D$ has a positive integer weight $w_{ij} \geq 1$ which is set to the $\Delta$ of $a_i$ and $a_j$. We also generalize the MVC to an edge-weighted minimum vertex cover (EWMVC) which is an assignment of non-negative integer values $x_1, \ldots, x_k$, one for each vertex, such that $x_i + x_j \geq w_{ij}$ for all $(v_i, v_j) \in E_D$ while minimizing the sum of $x_i$. $x_i$ can be interpreted as a lower bound on the minimum increase in the cost of the path of $a_i$ to get conflict-free paths. The sum of $x_i$ of the EWMVC of $G_{WD}$ is an admissible $h$-value for $N$ since, for each edge $(v_i, v_j) \in E_D$, the sum of the costs of the paths of agents $a_i$ and $a_j$ has to increase by at least $w_{ij}$. We refer to this heuristic as the WDG heuristic. It strictly dominates the DG heuristic. EWMVC is NP-hard since MVC is NP-hard and is a special case of EWMVC where the weights of all edges are 1. To solve EWMVC, we divide $G_D$ into multiple connected components, and run a branch and bound algorithm on each. The EWMVC of $G_{WD}$ is the union of the EWMVCs of all components. Similar dependency graphs and EWMVCs for heuristics guidance were used in the context of MAPF for large agents [Li et al., 2019b], heuristic search for sliding tile puzzles [Felner et al., 2004] and cost-optimal planning [Pomerening et al., 2013].

5.2 Constructing $G_{WD}$

We first construct vertices and edges in $G_{WD}$ for $N$ using the same method in Section 4. For each edge $(v_i, v_j) \in E_{WD}$, we run a MAPF algorithm to find the SOC for $a_i$ and $a_j$ and assign their $\Delta$ to $w_{ij}$. Here, the pathfinding problem is a 2-agent problem with the constraints from $N\.constraints$ on the two agents. Most optimal MAPF algorithms can be adapted to satisfy these constraints. We tried three algorithms in our experiments: CBSH (i.e., CBS with the WDG heuristic [Felner et al., 2018]), EPEA* [Goldenberg et al., 2014] and ICTS [Sharon et al., 2013], and CBSH is the strongest.

One enhancement we use in CBSH is that we set the $h$-value of the CT root node to 1. 1 is admissible because for $a_i$
and \( a_j \) their \( \Delta = w_{ij} \geq 1 \). This can help CBSH to efficiently resolve cardinal rectangle conflicts [Li et al., 2019a] or other symmetric conflicts. Figure 1(left) shows an example of a cardinal rectangle conflict. The cost of the optimal solution is 9. As CBSH searches in a best-first manner, it has to expand all CT nodes of cost 8, even if it has already generated a CT node of cost 9 with an optimal solution. However, if the \( h \)-value of the CT root node is 1, with a good tie-breaking rule (such as depth-first), CBSH can quickly generate a CT node of cost 9 with an optimal solution and return this solution immediately. Empirically, this speeds up CBSH for the 2-agent problem by up to 3 orders of magnitude.

Similarly to Section 4.2, for each CT non-root node, we need to find the edges and calculate the weights for only one agent (the one that has a new constraint) and copy the rest of the edges and the weights from the parent CT node.

### 6 Runtime Reduction Techniques

DG and WDG usually have a larger \( h \)-value than CG. However, computing these heuristics incurs extra overhead per CT node. In this section, we introduce some simple techniques to reduce the runtime overhead for heuristics calculation.

**Lazy Computation of Heuristics.** The high-level search of CBSH resembles an A* search, so techniques to speed up A* can also be applied here. Lazy A* [Tolpin et al., 2013] improves A* by evaluating expensive heuristics lazily. Instead of computing the slower heuristic \( h_2 \) immediately after generating a new node \( N \), it first computes a faster but less informed heuristic \( h_1 \) (or even uses zero as the heuristic) and inserts \( N \) into OPEN. Only when \( N \) emerges from OPEN, it computes \( h_2 \) for it and re-inserts it into OPEN.

Here, each of the CG, DG or WDG heuristics is treated as \( h_2 \), and we define \( h_1 \) for a child CT node \( N' \) of \( N \) as follows. \( N'.h = \max\{N.h - \max_{j \in E_D} \text{w}_{ij}, N.cost + N.h - N'.cost, 0\} \)

where \( i \) is the index of the agent that is re-planned at \( N' \). Here, for simplicity, we view both the conflict graph and the pairwise dependency graph as an edge-weighted pairwise dependency graph whose weights of all edges are 1. \( N.h - \max_{j \in E_D} \text{w}_{ij} \) is admissible because it is a lower bound on the sum of \( x_i \) of the EWMVC of \( G_{WD} = (V_D \setminus \{v_i\}, E_D \setminus \{(i, j) : \forall j, (i, j) \in E_D, W_D\}). N.cost + N.h - N'.cost \) is admissible because the \( f \)-value (i.e., \( N.cost + N.h \)) is non-decreasing. Empirically, the runtime overhead of OPEN operations (e.g., insert or pop a node) is negligible.

**Memoization.** Memoization is an optimization technique to speed up algorithms by storing the results of expensive function calls and returning the cached result when the same inputs occur again. Here, we use memoization to store the results of merging the MDDs and 2-agent pathfinding. The input is the indices of two agents and the set of constraints imposed on them. The output is the existence of the corresponding edge and the edge weight if it exists. Empirically, the memory overhead of caching and the runtime overhead of storing and querying results are both negligible. Memoization can also be applied to save runtime of building MDDs.

| Table 1: \( h \)-values of the CT root node. obs represents the percentage of cells that are randomly blocked on a 20 \( \times \) 20 grid. |
|---|---|---|---|---|---|---|---|---|
| k | CG | DG | WDG | CG | DG | WDG | obs |
| 30 | 0.2 | 1.0 | 1.2 | 16 | 5.9 | 7.9 | 11.6 |
| 40 | 0.5 | 1.7 | 2.0 | 20 | 4.8 | 4.8 | 15.2 |
| 50 | 0.1 | 1.0 | 1.3 | 24 | 6.9 | 7.0 | 22.2 |

**Lazy Computation of Heuristics.** Figure 4 shows the runtime breakdown per expanded CT node, averaged over 300 instances with different numbers of agents. The CBS runtime (yellow) of the three CBSH solvers is much larger than both of them because CBS is the 2-agent pathfinding algorithm. Our code is written in C++, and our experiments are conducted on a 2.80 GHz Intel Core i7-7700 laptop with 8 GB RAM.

#### 7.1 Small Maps

We first use 4-neighbor 20 \( \times \) 20 grids. Specifically, we focus on an empty map which is a 20 \( \times \) 20 grid with no blocked cells and a dense map which is a 20 \( \times \) 20 grid with 30% randomly blocked cells. For each solver, map and number of agents, we use 50 instances with random start and goal vertices, each with a runtime limit of 1 minute.

**h-values of the CT root node.** Table 1 shows the average \( h \)-values of the CT root node. On the empty map, DG is much larger than CG while WDG is only slightly larger than DG because agents on the empty map have many bypasses, and thus the \( \Delta \) for a pair of agents is 0 or 1 in most cases. However, on the dense map, DG is only slightly larger than CG while WDG is much larger than both of them because most conflicts are cardinal and the map contains many narrow corridors, which induce a large \( \Delta \) for pairs of agents. The last four columns show the results of 20 agents on grids with increasing obstacle densities, which show more details of the transition between empty grids to dense grids.

**Runtime overhead of heuristics calculation.** Figure 4 shows the runtime breakdown per expanded CT node, averaged over 300 instances with different numbers of agents. The CBS runtime (yellow) of the three CBSH solvers is slightly different because the different heuristics cause CBS to expand different sets of CT nodes. The runtime of constructing \( G_D \) and \( G_{WD} \) (blue) is small, due to the memoization technique, which saves more than 90% of edge and weight computation time. Although we use simple algorithms to solve the NP-hard problems MVC and EWMVC, their runtime (red) is also small due to the small sizes of \( G_D \) and \( G_{WD} \). Lazy computation of heuristics also contributes to the reduction in the runtime overhead as the stronger heuristics are computed for only 65% of generated CT nodes.

**Overall performance.** Figure 5 shows the success rate, the average number of expanded CT nodes and the average...
of-the-art CBS-based algorithm in previous research. The results show that all the new algorithms beat previous algorithms. The rectangle reasoning technique slightly speeds up WDG in most cases. Therefore, WDG+R is the strongest algorithm among them and therefore is the new state-of-the-art CBS-based algorithm.

**Results with longer time limits.** Figure 7(left) shows the success rate on 50 instances of 100 agents on the large map with different time limits. As the time limit increases, the benefit of using WDG and DG over CG increases as well. In general, it is worth spending a “constant” extra time on each CT node to obtain a better heuristic, since a larger heuristic value usually leads to an exponential reduction in the number of CT nodes. Figure 7(right) shows the results with a time limit of 30 minutes. Although DG and WDG have a larger runtime overhead compared to small maps, WDG significantly outperforms DG which in turn significantly outperforms CG in terms of both success rate and runtime. For example, compared with CG, WDG improves the success rate by a factor of 2 and runs faster by a factor of 50.

### 7.3 Comparing with The Perfect Heuristic

Table 2 shows the average $h$-values and the average $h^*$-value (=optimal cost) of the CT root node. On the dense map, WDG is much smaller than $h^*$ because agents are deeply coupled and reasoning about the pairwise dependency between agents is not enough. However, on the empty map or on the large map, WDG is close to $h^*$ because agents are less coupled and reasoning about the pairwise dependency between agents is relatively accurate in many cases. This explains why WDG has the largest $h$-value improvement on the dense map over all three maps but the smallest node reduction factor on the dense map over all three maps (as shown in Figures 5 and 7).

### 8 Conclusions and Future Work

In this paper, we analyzed the limitation of the heuristic used in the high-level guidance for CBS, a state-of-the-art algorithm for Multi-Agent Pathfinding. We proposed two new admissible heuristics by reasoning about the pairwise dependency between agents. They always dominate the old heuristic in their $h$-values and only incur a small runtime overhead.
per node. Empirically, they increase the success rates and speeds of CBS with the old heuristic by up to a factor of 50.

There are several interesting directions for future work. (1) Study heuristics for sub-optimal CBS-based algorithms [Barer et al., 2014]. (2) Apply similar heuristics techniques to other MAPF algorithms, such as ICTS [Sharon et al., 2013] or MDD-SAT [Surynek et al., 2016]. (3) Generalize these heuristics to groups larger than pairs of agents, e.g., to triples and quadruples.

References


