Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search

Jiaoyang Li[1] , **Ariel Felner**[2] , **Eli Boyarski**[2] , **Hang Ma**[1] and **Sven Koenig**[1]
[1]University of Southern California
[2] Ben Gurion University of the Negev
jiaoyanl@usc.edu, felner@bgu.ac.il, boyarske@post.bgu.ac.il, {hangma, skoenig}@usc.edu

Abstract

Conflict-Based Search (CBS) and its enhancements are among the strongest algorithms for Multi-Agent Path Finding. 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 dependencies between agents. Empirically, CBS with either new heuristic 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.

1 Introduction

Multi-Agent Path Finding (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, 2013b], many optimal MAPF algorithms have been developed in recent years, including reduction-based algorithms [Yu and LaValle, 2013a; Erdem *et al.*, 2013; Surynek *et al.*, 2016; Barták *et al.*, 2017], 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].

Conflict-Based Search (CBS) [Sharon *et al.*, 2015] is a popular two-level search-based MAPF algorithm which resolves collisions by adding constraints at a high level and computing paths consistent with those constraints at a low level. It is widely used in many real-world applications, such as warehouse robots [Ma *et al.*, 2017a; Hönig *et al.*, 2019], quadrotor swarms [Hönig *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; Li *et al.*, 2019b; Gange *et al.*, 2019]. *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 a special type of collisions in the *current solution* (i.e., the paths in the current high-level node). In this paper, we further develop this direction. We first prove that CG can offer only a limited amount of information. We then introduce two new admissible heuristics, DG and WDG, by considering potential collisions in *future solutions* (i.e., the paths in the descendant high-level nodes) and reasoning about the pairwise dependencies between agents. WDG strictly dominates DG, which in turn strictly dominates CG. Empirically, the runtime overhead of calculating the new heuristics is reasonable, and WDG improves the success rate of CBS significantly compared to CG and reduces the number of expanded nodes and runtime by up to a factor of 50.

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 $\{a_1 \dots a_k\}$, where a_i has start vertex $s_i \in V$ and 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 $\langle a_i, a_j, v, t\rangle$ is a *vertex conflict* iff a_i and a_j are at the same vertex v at timestep t, and a tuple $\langle a_i, a_j, u, v, t\rangle$ is an *edge conflict* iff 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* $\langle a_i, v, t\rangle$ that prohibits agent a_i from being at vertex v at timestep t or an *edge constraint* $\langle a_i, u, v, t\rangle$ that prohibits agent a_i from moving from vertex u to vertex v between timesteps t and $t + 1$;

(2) a solution $N.solution$, that consists of a set of k cost-minimal paths, one for each agent, that satisfy $N.constraints$; and

(3) a cost $N.cost$, that is equal to the sum of the costs of the paths in $N.solution$.

The root CT node contains an empty set of constraints.

When CBS chooses a CT node N for expansion, it checks for conflicts in $N.solution$. If there are none, CBS terminates and returns $N.solution$. Otherwise, CBS chooses one of the conflicts (by default, arbitrarily) and resolves it by *splitting* N into two child CT nodes. In each child CT node, one agent from the conflict is prohibited from using the contested vertex or edge by way of an additional constraint. The path of this agent then no longer satisfies the constraints of the child CT node and must be replanned by a low-level search (e.g., a time-space A* search [Silver, 2005]). All other paths remain unchanged. If the low-level search cannot find any path that satisfies the constraints, this child CT node does not have any solution and therefore is pruned. With two child CT nodes per conflict, CBS guarantees optimality by exploring both ways of resolving each conflict.

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 each 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 $N.cost$. (i.e., a conflict is cardinal iff all shortest paths of the two conflicting agents traverse the conflicting vertex/edge at the conflicting timestep). 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 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 $\langle a_1, a_2, B2, 1\rangle$ at the root CT 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 because, when a_1 or a_2 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 only at timestep 5.

ICBS uses MDDs to classify conflicts. A Multi-Valued Decision Diagram (MDD) [Sharon *et al.*, 2013] for a_i at N is a directed acyclic graph that consists of all cost-minimal paths of a_i from s_i to g_i that satisfy $N.constraints$. Nodes at depth t of the MDD for a_i correspond to all vertices where a_i can be at timestep t along one of its cost-minimal paths. 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 the MDDs for both agents. Figure 1(middle) shows the MDDs for a_1 and a_2 at the root CT node, respectively. Since both MDDs have 2 nodes at timestep 1, the conflict $\langle a_1, a_2, B2, 1\rangle$ 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, then an h-value of 1 is admissible

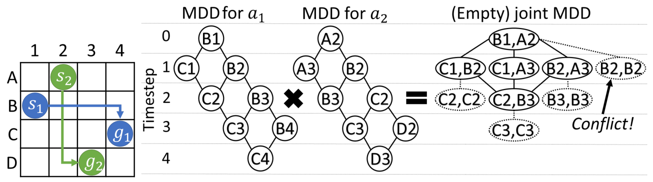


Figure 1: A MAPF instance on a 4-neighbor grid with the corresponding MDDs and joint MDD.

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, then 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 to find a larger h-value for a cardinal conflict? The following theorem answers the question. The proof of this theorem is given in the appendix.

Theorem 1. *Suppose that CBS chooses to resolve a conflict between a_i and a_j at timestep t at a CT node N and both child CT nodes of N, N_1 (with an additional constraint imposed on a_i) and N_2 (with an additional constraint imposed on a_j) have solutions. If the conflict occurs after one of the agents, say a_i, reaches its goal vertex (i.e., $t \geq \mu_i$, where μ_i is the cost of the path of a_i in $N.solution$), then $N_1.cost = N.cost + t + 1 - \mu_i$ and $N_2.cost \in \{N.cost, N.cost + 1\}$. Otherwise (i.e., it occurs before both agents reach their goal vertices), $N_1.cost, N_2.cost \in \{N.cost, N.cost + 1\}$.*

Therefore, if both child CT nodes of N have solutions (which is always true in practice), a conflict can be regarded as an admissible h-value of at most 1. Then 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 need to be explored.

4 The DG Heuristic

The CG heuristic considers only cardinal conflicts in $N.solution$. To improve on that we also need to 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 $\langle a_1, a_2, B2, 1\rangle$ by adding a constraint for one of the agents, a new conflict will occur no matter what new cost-minimal path the agent picks. In fact, any two 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 reflect that all cost-minimal paths of the corresponding two agents have conflicts.

4.1 Pairwise Dependency Graph $\mathbf{G_D}$

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 $(v_i, v_j) \in E_D$ iff a_i and a_j are *dependent*, i.e., all their cost-minimal paths that satisfy $N.constraints$ 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 as in [Felner *et al.*, 2018] to determine an MVC. Its complexity is $O(2^q|V_D|)$, where q is the size of the MVC.

4.2 Constructing $\mathbf{G_D}$

To construct G_D for N, we need to analyze the dependencies between every pair of agents. Let μ_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 their 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-cardinal 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 provide an answer, we try to *merge* the MDDs for the two agents into a joint MDD using the method described in [Sharon *et al.*, 2013]. The two agents are dependent iff their joint MDD is empty. Details of the merging are given in Section 4.3.

Since each CT node has an additional constraint imposed on only one agent, we only need to look at the dependencies between this agent and all other agents and can copy the edges for the other pairs of agents from the G_D for the parent CT node. Of course, at the root CT node, we still need to look at the dependencies for all pairs of agents. CG already build MDDs to classify conflicts. So, for DG, we get these MDDs for free. The only overhead of DG over CG comes from merging the MDDs.

4.3 Merging the MDDs

The *joint MDD* of the MDDs for a_i and a_j at N 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 without conflicts. Let μ_k ($k = i, j$) denote the depth of the MDD for a_k. If $\mu_i \neq \mu_j$, a path of $|\mu_i - \mu_j|$ dummy goal vertices is added to the sink node of the shallower MDD (representing the agent sitting at its goal vertex) so that both MDDs have the same depth. The joint MDD is built level by level. The merging procedure starts at the joint state (s_i, s_j) at level 0. 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 the MDD for a_i and the child nodes of v_j at level t in the MDD for a_j should be examined. Only conflict-free pairs are added. a_i and a_j are dependent iff the joint MDD is *empty*, i.e., does not contain state (g_i, g_j) at level $\max\{\mu_i, \mu_j\}$.

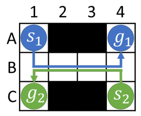


Figure 2: A MAPF instance on a 4-neighbor grid with $\Delta_{12} = 4$.

Figure 1(right) shows an example of merging the MDDs. The joint MDD starts at (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 for the pair (B2, B2), which represents a conflict space. 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, and thus a_1 and a_2 are dependent.

5 The WDG Heuristic

For a CT node N and two agents a_i and a_j, we refer to the difference between the minimum sum of the costs of their conflict-free paths that satisfy $N.constraints$ and the sum of the costs of their paths in $N.solution$ as Δ_{ij} ($\Delta_{ij} \geq 0$). a_i and a_j are dependent iff $\Delta_{ij} > 0$.

Although G_D captures the information whether $\Delta_{ij} > 0$ for any pair of agents, it does not capture the information how large the value of Δ_{ij} is. When $\Delta_{ij} > 0$, the DG heuristic only uses 1 (a lower bound on Δ_{ij}) as an admissible h-value. However, in some cases, Δ_{ij} could be larger than 1. For instance, in Figure 2, $\Delta_{12} = 4$ because one of the agents must wait 4 timesteps at its start vertex. Therefore, we introduce the WDG heuristic, which captures not only the pairwise dependencies between agents but also the extra cost that each pair of dependent agents will contribute to the total cost.

5.1 Weighted Pairwise Dependency Graph $\mathbf{G_{WD}}$

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 G_D. The weight on each edge $(v_i, v_j) \in E_D$ equals Δ_{ij} at N. Here, Δ_{ij} is always larger than 0 as a_i and a_j are dependent. We also generalize the MVC to an *edge-weighted minimum vertex cover* (EWMVC), which is an assignment of non-negative integers $x_1, \ldots, x_k$, one for each vertex, which minimizes the sum of the x_i subject to the constraints that $x_i + x_j \geq \Delta_{ij}$ for all $(v_i, v_j) \in E_D$. x_i can be interpreted as the increase in the cost of the path of a_i. 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 Δ_{ij}. We refer to this heuristic as the WDG heuristic. It strictly dominates the DG heuristic. Calculating the EWMVC is NP-hard since calculating the MVC is NP-hard and a special case of calculating EWMVC where the weights of all edges are 1. To calculate the EWMVC, we divide G_{WD} into multiple connected components and calcualte the EWMVC on each component with a branch-and-bound algorithm that branches on the possible values of each x_i in the component and prunes nodes using the cost of the best result so far. The EWMVC of G_{WD} is the union of the EWMVCs of all components. Similar dependency graphs and EWMVCs for heuristic search were used in the context of MAPF for large agents [Li *et al.*, 2019c], sliding tile puzzles [Felner *et al.*, 2004] and cost-optimal planning [Pommerening *et al.*, 2013].

5.2 Constructing G_{WD}

We first construct the vertices and edges in G_{WD} for N using the same method as in Section 4. To calculate the weight Δ_{ij} for each edge $(v_i, v_j) \in E_D$, we run a MAPF algorithm to find the minimum sum of the costs of the conflict-free paths of a_i and a_j that satisfy $N.constraints$ (ignoring the other agents). Here, the pathfinding problem is a two-agent problem with the constraints from $N.constraints$ imposed on the two agents. Most optimal MAPF algorithms can be adapted to satisfy these constraints.

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

5.3 The Two-Agent Problem

We tried three search-based MAPF algorithms to solve the two-agent problem in our experiments: CBSH [Felner *et al.*, 2018] (i.e., CBS with the CG heuristic), EPEA* [Goldenberg *et al.*, 2014] and ICTS [Sharon *et al.*, 2013], and CBSH is significantly faster than the other two.

One enhancement that we use in CBSH for the two-agent problem is that we set the h-value of the root CT node to 1. One is admissible because Δ_{ij} is at least 1. This can help CBSH to resolve cardinal rectangle conflicts [Li *et al.*, 2019b] or other symmetric conflicts efficiently. 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 root CT 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. In our experiments, this speeds up CBSH for the two-agent problem by up to 3 orders of magnitude.

6 Runtime Reduction Techniques

DG and WDG usually have larger h-values than CG. However, computing these heuristics incurs overhead per CT node. In this section, we introduce a number of simple techniques to reduce the runtime overhead for the calculation of the heuristics.

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 expensive h-value $h_2(N)$ immediately after generating a new node N, lazy A* first computes a cheaper but less informed h-value $h_1(N)$ (or even uses zero) and inserts N into OPEN. Only when N emerges from OPEN, it computes $h_2(N)$ for it and re-inserts it into OPEN.

Here, for simplicity, we view both the conflict graph and the pairwise dependency graph as an edge-weighted pairwise dependency graph whose edges all have weight one. 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 $\max\{N.h - \max_{j:(i,j)\in E_D} \Delta_{ij}, N.cost + N.h - N'.cost, 0\}$, where i is the index of the agent whose path gets re-planned at N'. The first term $N.h - \max_{j:(i,j)\in E_D} \Delta_{ij}$ is a lower bound on the sum of x_i of the EWMVC of the sub-graph of G_{WD} of N without edges incident on vertex v_i. It is admissible because the sum of x_i of the EWMVC of G_{WD} of N' should be no smaller than the sum of x_i of the EWMVC of this sub-graph. The second term $N.cost + N.h - N'.cost$ 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 caching the results of expensive function calls and returning the cached results when the same inputs occur again. Here, we use memoization to store the results of merging the MDDs and solving the two-agent problems. The inputs are the indices of two agents and the set of constraints imposed on them. The output is the existence of the corresponding edge and, if its exists, its edge weight. Empirically, the memory overhead of caching and the runtime overhead of storing and retrieving results are both negligible, and the cached results are used frequently. This is because CBS often repeatedly resolves the same conflict in different branches, and many CT nodes thus have the same set of constraints imposed on the same agent. Memoization can also be used to save runtime for building MDDs.

7 Experimental Results

We experiment with CBS, ICBS and CBSH with the CG, DG and WDG heuristics on 4-neighbor grids. All CBSH solvers use the two improvements discussed in Section 6, and the WDG heuristic uses the CBSH algorithm discussed in Section 5.3 to solve the two-agent problem. We generate 50 instances with random start and goal vertices for each map and each number of agents. 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.

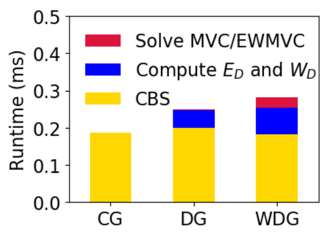
7.1 Small Maps

First, we test the solvers on 20×20 grids. We focus on an *empty map*, which is a 20×20 grid with no blocked cells, and a *dense map*, which is a 20×20 grid with 30% randomly blocked cells. We use a time limit of 1 minute for each solver on each instance.

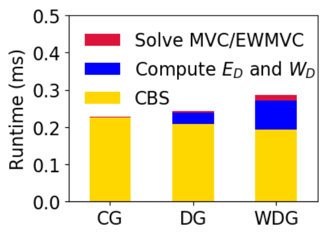
h-values of the root CT node. Table 1 shows the h-values of the root CT node. On the empty map, DG is much larger than CG while WDG is only slightly larger than DG because agents on the empty map usually have many bypasses, and

Empty map				Dense map				20 agents			
k	CG	DG	WDG	k	CG	DG	WDG	obs	CG	DG	WDG
30	0.2	1.0	1.2	16	3.9	3.9	11.6	0	0.1	0.5	0.5
40	0.5	1.7	2.0	20	4.8	4.8	15.2	10	1.0	1.3	2.1
50	0.6	2.3	2.8	24	6.9	7.0	22.2	20	3.0	3.1	6.2

Table 1: Average h-values of the root CT node. k represents the number of agents, and obs represents the percentage of cells that are randomly blocked on a 20×20 grid.

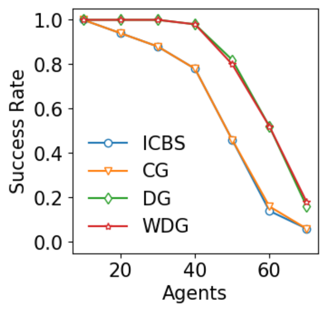


(a) Empty map.

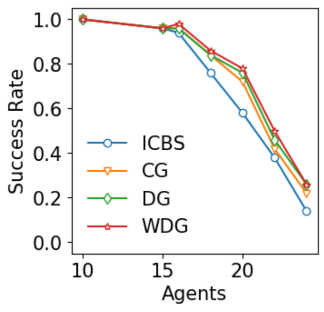


(b) Dense map.

Figure 3: Average runtime per expanded CT node over 300 instances with different numbers of agents.



(a) Empty map.



(b) Dense map.

Figure 4: Success rate (i.e., the percentage of solved instances).

thus Δ_{ij} 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 Δ_{ij}. The last four columns show the results for 20 agents on grids with increasing obstacle densities to provide more details on the transition from empty grids to dense grids.

Runtime overhead of heuristics calculation. Figure 3 shows the runtime breakdown per CT node. The CBS runtimes (yellow) of the three solvers are slightly different because the different heuristics cause CBS to expand different sets of CT nodes. The runtimes of constructing G_D and G_{WD} (blue) are small due to the memoization technique, which saves more than 90% of the edge and weight computation time. Although we use simple algorithms to solve the NP-hard problems MVC and EWMVC, their runtimes (red) are also small due to the small sizes of G_D and G_{WD}. The lazy computation of heuristics also contributes to the reduction in the runtime overhead as the expensive heuristics are computed for only 65% of the generated CT nodes.

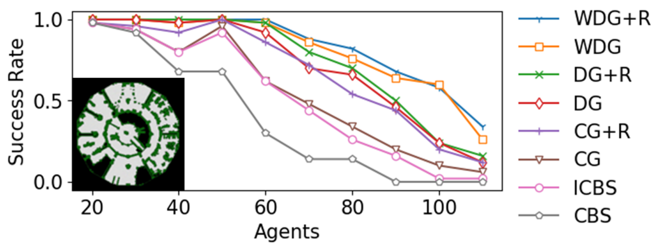


Figure 5: Success rate on on the large map."+R" means that the solver uses the rectangle reasoning technique.

Overall performance. Figure 4 and Table 2 show the success rate, the average number of expanded CT nodes and the average runtime of the solvers. The number of expanded CT nodes is consistent with the computed h-value of the root CT node. That is, a larger h-value usually leads to a smaller number of expanded CT nodes. With respect to success rate and runtime, both DG and WDG outperform CG. In particular, DG runs slightly faster than WDG on the empty map as it has a smaller runtime overhead than WDG, while WDG runs much faster than DG on the dense map as it leads to a larger node reduction than DG. The usefulness of CG heavily depends on the particular instance. It has almost the same performance as ICBS (= CBSH with h-values that are always zero) on the empty map where cardinal conflicts are rare. However, our new heuristics are efficient on both maps and up to 9 times faster than CG (e.g., for 40 agents on the empty map) for the solved instances (which are relatively easy).

7.2 Large Maps

Next, we test the solvers on a *large map*. We use the benchmark game map *lak503d* from [Sturtevant, 2012], which is a 192×192 grid with 51% blocked cells (Figure 5).

Plugging in the rectangle reasoning technique. Li *et al.* [2019b] analyze the symmetries in grid-based MAPF and improve CBSH using a rectangle reasoning technique. This technique is able to find some (but not all) edges in G_D, so it can save some of the effort of merging the MDDs. Therefore, we add the most advanced rectangle reasoning technique RM from [Li *et al.*, 2019b] to all CBSH solvers. Figure 5 shows their success rates within 1 minute. In particular, CG+R is identical to CBSH-RM from [Li *et al.*, 2019b] and is the state-of-the-art CBS-based solver in previous research. The results show that all new solvers beat the previous solvers. The rectangle reasoning technique slightly speeds up WDG in most cases. As a result, WDG+R is the strongest solver among them and therefore is the new state-of-the-art CBS-based solver.

Results with longer time limits. Figure 6(left) shows the success rates over 50 random 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 some extra time per CT node to obtain a better h-value, since a larger h-value usually leads to an exponential reduction in the number of CT nodes. Figure 6(right) shows the results with a time limit of 30 minutes. Although DG and WDG have a larger runtime

Agents	Instances	ICBS	CG	DG	WDG
Nodes ($\times$1000)					
30	44	3.6	2.6	**0.5**	**0.5**
40	39	8.9	7.0	**0.2**	**0.2**
50	23	12.4	10.1	**2.9**	**2.9**
Runtime (s)					
30	44	0.5	0.4	**0.1**	**0.1**
40	39	1.0	0.9	**0.1**	**0.1**
50	23	1.7	1.5	**0.6**	0.7

(a) Empty map.

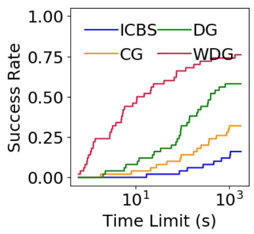
Agents	Instances	ICBS	CG	DG	WDG
Nodes ($\times$1000)					
16	47	20.2	9.6	7.8	**6.1**
20	29	20.2	13.6	10.7	**8.9**
24	7	79.6	47.4	33.2	**15.2**
Runtime (s)					
16	47	7.0	**2.4**	**2.4**	**2.4**
20	29	4.0	3.3	2.1	**1.9**
24	7	17.9	9.6	5.4	**3.0**

(b) Dense map.

Table 2: Average expanded CT nodes and average runtime over instances solved by all solvers.

Empty map					Dense map					Large map				
Agents	CG	DG	WDG	h^*	Agents	CG	DG	WDG	h^*	Agents	CG	DG	WDG	h^*
30	0.2	1.0	1.2	1.7	16	3.9	3.9	11.5	18.6	60	3.6	4.0	6.7	7.6
40	0.5	1.6	2.0	3.3	20	4.7	4.7	14.0	23.2	80	5.7	6.5	10.9	12.2
50	0.5	2.2	2.6	4.7	24	6.5	6.5	18.9	28.5	100	8.6	9.2	15.6	18.0

Table 3: Average h-values and average h^*-values of the root CT node over instances of which the h^*-value is known, i.e., instances solved by at least one CBSH solver.



	CG	DG	WDG
All 50 instances			
h-value of the root node	9.4	10.1	**17.0**
Runtime per node (ms)	**16.1**	21.7	21.9
Success rate	0.32	0.58	**0.76**
16 instances solved by all CBSH solvers			
Nodes ($\times$1000)	19.9	6.9	**0.4**
Runtime (s)	319	141	**6**

Figure 6: Results for 100 agents on the large map. The right table uses a time limit of 30 minutes.

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 3 compares the average h-values of the root CT node by different CBSH solvers with the average h^*-values of the root CT node (i.e., the optimal solution cost minus the cost of the root CT node). On the dense map, WDG is significantly smaller than h^* because agents are deeply coupled and reasoning about the pairwise dependencies between agents is not enough. However, on the empty map or the large map, WDG is close to h^* because agents are less coupled and reasoning about the pairwise dependencies between agents is enough in many cases. In other words, h/h^* is closer to 1 on the empty map or the large map than on the dense map. This explains why, compared to CG, WDG has the largest h-value improvement on the dense map over all three maps (as shown in Table 1) but the smallest node reduction factor on the dense map over all three maps (as shown in Table 2 and Figure 6).

8 Conclusions and Future Work

In this paper, we analyzed the limitations of the heuristic used to provide high-level guidance for CBS, a state-of-the-art algorithm for multi-agent path finding. We proposed two new admissible heuristics by reasoning about the pairwise dependencies between agents. They always dominate the old heuristic 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 admissible or inadmissible heuristics for sub-optimal CBS-based algorithms [Barer *et al.*, 2014]. (2) Apply similar heuristics 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.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189 and 1837779 as well as a gift from Amazon. The research was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2017692.

A Proof of Theorem 1

In order to help explain the proof, we define another type of MDDs, called *extended MDDs*, that ignore constraints, i.e., they are allowed to include nodes prohibited by constraints. Formally, an extended MDD MDD_i^μ for a_i is a directed acyclic graph that consists of *all* paths of a_i from s_i to g_i within μ timesteps. In particular, MDD_i^μ is empty iff μ is smaller than the minimum cost of the path of a_i. All MDDs we discuss in this section are extended MDDs. We say that a node $(x, t) \in MDD_i^\mu$ iff MDD_i^μ has vertex x at level t, i.e., there is a path from s_i at timestep 0 to x at timestep t

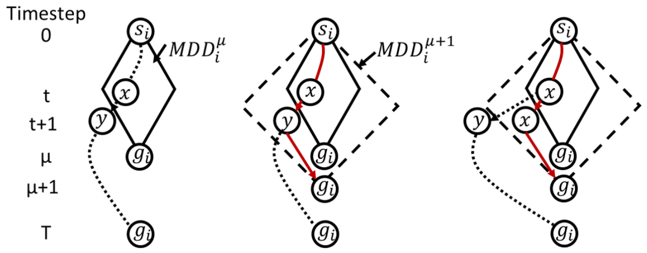


Figure 7: Constructing a path of cost $\mu + 1$. The solid and dashed diamonds include all nodes in MDD_i^{μ} and $MDD_i^{\mu+1}$, respectively.

and then to g_i by timestep μ. 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, the extended MDDs have the following two properties:

(1) $(x, t) \in MDD_i^{\mu}$ iff $dist(s_i, x) \leq t$ and $dist(x, g_i) \leq \mu - t$.

(2) For any path P of a_i, if $(P(t), t) \notin MDD_i^{\mu}$, then $(P(t')), t') \notin MDD_i^{\mu}$ for all $t' \geq t$.

Property (1) can be obtained from the definition. Property (2) holds because, by contradiction, if $(P(t')), t') \in MDD_i^{\mu}$, then there is a sub-path P' in MDD_i^{μ} from $P(t')$ at timestep t' to g_i at timestep μ. The path that follows a prefix of P from s_i at timestep 0 to $P(t')$ at timestep t' and then follows P' to g_i at timestep μ traverses $P(t)$ at timestep t, i.e., $(P(t), t) \in MDD_i^{\mu}$. Property (2) tells us that, once a path *leaves* an extended MDD, it will never revisit this extended MDD later.

We say that MDD_i^{μ} includes all nodes prohibited by a set of constraints C iff MDD_i^{μ} has vertex v at level t for any vertex constraint $\langle a_i, v, t\rangle \in C$ and edge (u, v) from level t to level $t + 1$ for any edge constraint $\langle a_i, u, v, t\rangle \in C$.

Lemma 2. *Let C be a set of constraints and μ be a large enough integer such that MDD_i^{μ} is not empty and includes all nodes prohibited by C. If a_i has a path that satisfies 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 that satisfies C. Let T be the cost of P. If $T \leq \mu + 1$, then we are done. Otherwise, as shown in Figure 7, we will construct a path of cost $\mu + 1$ by letting a_i (1) first follow the prefix of P to the border of MDD_i^{μ}, (2) then cross the border to a node in $MDD_i^{\mu+1}$, and (3) last follow a path in $MDD_i^{\mu+1}$ to g_i at timestep $\mu+1$. This new path satisfies C because P satisfies C and any nodes or edges outside MDD_i^{μ} also satisfy C. The existence of such a path is proved as follows.

Node $(s_i, 0)$ (the first node of P) is in MDD_i^{μ}. Node (g_i, T) (the last node of P) is not in MDD_i^{μ} (because $T > \mu$). So, there is a timestep $t < T$ such that $(P(t), t) \in MDD_i^{\mu}$ and $(P(t + 1), t + 1) \notin MDD_i^{\mu}$. Let x and y represent $P(t)$ and $P(t+1)$, respectively (Figure 7(left)). There are now two cases, and we show how to construct a path of cost $\mu + 1$ that satisfies C in each case.

Case 1: $(y, t + 1) \in MDD_i^{\mu+1}$ (Figure 7(middle)). There is a sub-path P' in $MDD_i^{\mu+1}$ from y at timestep $t + 1$ to g_i at timestep $\mu + 1$. By applying $(y, t + 1) \notin MDD_i^{\mu}$ to Property (2), P' does not traverse any MDD node in MDD_i^{μ}, 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: $(y, t + 1) \notin MDD_i^{\mu+1}$ (Figure 7(right)). Since node $(y, t + 1)$ is on P, $dist(s_i, y) \leq t + 1$. Then, from Property (1), $dist(y, g_i) > (\mu + 1) - (t + 1)$. Rearranging the terms yields

$$\begin{aligned}\mu \leq& t + dist(y, g_i) - 1 && (1)\\ \leq& t + dist(y, x) + dist(x, g_i) - 1 && (2)\\ \leq& t + dist(x, g_i) && (3)\\ \leq& \mu. && (4)\end{aligned}$$

Inequality (2) is based on the the triangle inequality. Inequality (3) is from $dist(y, x) \leq 1$. Inequality (4) is obtained by applying $(x, t) \in MDD_i^{\mu}$ to Property (1). By comparing the first line and the last line of these inequalities, all lines are actually equal. Specifically, for Inequality (4),

$$t + dist(x, g_i) = \mu. \qquad (5)$$

Since node (x, t) is on P, $dist(s_i, x) \leq t$. From Equation (5) and Property (1), we know that $(x, t + 1) \notin MDD_i^{\mu}$ and $(x, t + 1) \in MDD_i^{\mu+1}$. Hence, there is a sub-path P'' in $MDD_i^{\mu+1}$ from x at timestep $t + 1$ to g_i at timestep $\mu + 1$. From Property (2), P'' does not traverse any MDD node in MDD_i^{μ}, 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. □

Finally, we prove Theorem 1 based on Lemma 2.

Proof. Without loss of generality, we only focus on N_1. Every constraint in $N.constraints$ imposed on a_i was generated due to a conflict that occurred on one of the old paths of a_i, i.e., paths of a_i in the solutions of any ancestor CT nodes of N. The cost of any old path of a_i is no larger than μ_i. Therefore, $MDD_i^{\mu_i}$ includes all nodes prohibited by $N.constraints$.

If the chosen conflict occurs before a_i reaches g_i (i.e., $t < \mu_i$), then $MDD_i^{\mu_i}$ includes all nodes prohibited by $N_1.constraints$. From Lemma 2, we know that a_i has a path of cost no more than $\mu_i + 1$ that satisfies $N_1.constraints$. So, the cost-minimal path of a_i is of cost at most $\mu_i + 1$, and $N_1.cost \in \{N.cost, N.cost + 1\}$.

If the chosen conflict occurs after a_i reaches g_i (i.e., $t \geq \mu_i$), then the chosen conflict is $\langle a_i, a_j, g_i, t\rangle$ and the additional constraint added to N_1 is $\langle a_i, g_i, t\rangle$. MDD_i^{t} includes all nodes prohibited by $N_1.constraints$, and thus, by Lemma 2, a_i has a path of cost no more than $t + 1$ that satisfies $N_1.constraints$. On the other hand, since a_i is prohibited from being at g_i at timestep t, its cost-minimal path is of cost at least $t+1$. Therefore, the cost-minimal path of a_i is of cost $t + 1$, and $N_1.cost = N.cost + t + 1 - \mu_i$. □

References

[Barer *et al.*, 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *SoCS*, pages 19–27, 2014.

[Barták *et al.*, 2017] Roman Barták, Jirı Švancara, and Marek Vlk. Scheduling models for multi-agent path finding. In *MISTA*, pages 189–200, 2017.

[Boyarski *et al.*, 2015] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.

[Cohen *et al.*, 2016] Liron Cohen, Tansel Uras, T. K. Satish Kumar, Hong Xu, Nora Ayanian, and Sven Koenig. Improved solvers for bounded-suboptimal multi-agent path finding. In *IJCAI*, pages 3067–3074, 2016.

[Cohen *et al.*, 2018] Liron Cohen, Glenn Wagner, David M. Chan, Howie Choset, Nathan Sturtevant, Sven Koenig, and T. K. Satish Kumar. Rapid randomized restarts for multi-agent path finding solvers. In *SoCS*, pages 1909–1911, 2018.

[Erdem *et al.*, 2013] Esra Erdem, Doga Gizem Kisa, Umut Öztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, pages 290–296, 2013.

[Felner *et al.*, 2004] Ariel Felner, Richard E. Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

[Felner *et al.*, 2018] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Keonig. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, pages 83–87, 2018.

[Gange *et al.*, 2019] Graeme Gange, Daniel Harabor, and Peter J. Stuckey. Lazy CBS: Implicit conflict-based search using lazy clause generation. In *ICAPS*, 2019.

[Goldenberg *et al.*, 2014] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan R. Sturtevant, Robert C. Holte, and Jonathan Schaeffer. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

[Hönig *et al.*, 2018] Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

[Hönig *et al.*, 2019] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters*, 4(2):1125–1131, 2019.

[Li *et al.*, 2019a] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Ariel Felner, Hang Ma, and Sven Keonig. Disjoint splitting for multi-agent path finding with conflict-based search. In *ICAPS*, 2019.

[Li *et al.*, 2019b] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Symmetry-breaking constraints for grid-based multi-agent path finding. In *AAAI*, 2019.

[Li *et al.*, 2019c] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *AAAI*, 2019.

[Ma *et al.*, 2017a] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS*, pages 837–845, 2017.

[Ma *et al.*, 2017b] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, pages 270–272, 2017.

[Pommerening *et al.*, 2013] Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In *IJCAI*, pages 2357–2364, 2013.

[Sharon *et al.*, 2013] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, pages 173–178, 2010.

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

[Surynek *et al.*, 2016] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, pages 810–818, 2016.

[Tolpin *et al.*, 2013] David Tolpin, Tal Beja, Solomon Eyal Shimony, Ariel Felner, and Erez Karpas. Toward rational deployment of multiple heuristics in A*. In *IJCAI*, pages 674–680, 2013.

[Wagner and Choset, 2011] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, pages 3260–3267, 2011.

[Yu and LaValle, 2013a] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. In *ICRA*, pages 3612–3617, 2013.

[Yu and LaValle, 2013b] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, pages 1444–1449, 2013.