Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search *

Jiaoyang Li
Univ. of Southern California
jiaoyanl@usc.edu

Daniel Harabor
Peter J. Stuckey
Monash University
{daniel.harabor,peter.stuckey}@monash.edu

Ariel Felner
Ben-Gurion University
felner@bgu.ac.il

Hang Ma
Sven Koenig
Univ. of Southern California
{hangma,skoenig}@usc.edu

Abstract

Multi-Agent Path Finding (MAPF) is the planning problem of finding collision-free paths for a team of agents. We focus on Conflict-Based Search (CBS), a two-level tree-search state-of-the-art MAPF algorithm. The standard splitting strategy used by CBS is not disjoint, i.e., when it splits a problem into two subproblems, some solutions are shared by both subproblems, which can create duplication of search effort. In this paper, we demonstrate how to improve CBS with disjoint splitting and how to modify the low-level search of CBS to take maximal advantage of it. Experiments show that disjoint splitting increases the success rates and speeds of CBS and its variants by up to 2 orders of magnitude.

1 Introduction

Multi-Agent Path Finding (MAPF) is defined by a graph $G = (V, E)$ and a set of m agents $\{a_i \dots a_m\}$. Each agent a_i has a start vertex s_i and a goal vertex g_i. Time is discretized into timesteps. At each timestep, 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. We call $\langle a_i, a_j, v, t\rangle$ a *vertex conflict* iff a_i and a_j are at the same vertex v at timestep t, and $\langle a_i, a_j, u, v, t\rangle$ an *edge conflict* iff a_i and a_j traverse the same edge (u, v) in opposite directions between timesteps t and $t + 1$. We focus here on resolving vertex conflicts, except for the experimental section, since edge conflicts can be handled analogously. Our task is to find a set of conflict-free paths (referred to as a *solution*) which move all agents from their start vertices to their goal vertices while minimizing the sum of the costs of their paths.

Such problems appear in many application areas, including warehouse logistics (Wurman, D'Andrea, and Mountz 2008), office robots (Veloso et al. 2015), aircraft-towing vehicles (Morris et al. 2016) and computer games (Ma et al. 2017). Although MAPF is NP-hard to solve optimally (Yu and LaValle 2013; Ma et al. 2016b), numerous optimal and bounded-suboptimal MAPF algorithms have been proposed recently using different algorithmic approaches. Surveys are given in (Ma et al. 2016a; Felner et al. 2017).

In this paper, we dramatically improve a state-of-the-art MAPF algorithm, Conflict Based Search (CBS), by changing the splitting strategy of its high-level search. Currently, when CBS finds a conflict $\langle a_i, a_j, v, t\rangle$, it creates two subproblems by adding negative constraints: one prohibits a_i from being at v at timestep t, and one prohibits a_j from being at v at timestep t. This split is intuitive and removes the possibility of the same conflict re-occurring but can lead to substantial duplicate search effort because the set of paths where neither a_i nor a_j is at v at timestep t are shared by both subproblems. Hence, the split is not disjoint. By also using positive constraints, which force an agent to be at a given vertex at a given timestep, we introduce a disjoint splitting strategy that splits the problem into two disjoint subproblems and thus avoids the duplication of search effort. Empirically, we show that disjoint splitting significantly increases the success rates and speeds up CBS and its variants by up to 2 orders of magnitude.

2 Conflict-Based Search

Conflict-Based Search (CBS) (Sharon et al. 2015) is a two-level optimal MAPF algorithm. At the low level, it performs a best-first search to find a shortest path for each agent. At the high level, it performs a best-first search on a binary *constraint tree* (CT). Each CT node N contains a set of constraints that are used to coordinate agents to avoid conflicts and also a set of paths (referred to as a *plan* in N), one for each agent, that satisfy these constraints. The cost of N is the sum of the costs of the paths in its plan. CBS proceeds from one CT node to the next, checking for conflicts and calling its low-level search to replan paths one at a time. CBS terminates when the plan of the current CT node is conflict-free, which then corresponds to an optimal solution.

Constraints: A constraint is a spatio-temporal restriction introduced by CBS to resolve situations where the paths of two agents are in conflict. Specifically, a *negative constraint* $\overline{\langle a_i, v, t\rangle}$ prohibits a_i from being at v at timestep t.

Splits: When CBS expands a CT node N, it checks for conflicts in the plan of N. If there are none, then CBS termi-

*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.

nates. Otherwise, CBS chooses one of the conflicts (by default, randomly) 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 at the conflicting timestep 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. All other paths remain unchanged. With two child CT nodes per conflict, CBS guarantees optimality by exploring both ways of resolving each conflict.

2.1 Variants of CBS

Optimal variants: Instead of choosing conflicts randomly, *Improved CBS* (ICBS) (Boyarski et al. 2015) chooses cardinal conflicts first. A conflict is *cardinal* iff, when CBS uses this conflict to split a CT node N, the cost of each one of the two resulting child nodes is larger than the cost of N. Additionally, identifying disjoint cardinal conflicts can be used to add admissible heuristics to the costs of CT nodes. The resulting algorithm, called CBSH (Felner et al. 2018), is the state-of-the-art optimal CBS variant.

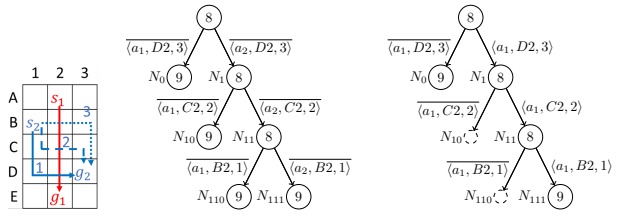
Both ICBS and CBSH use Multi-Valued Decision Diagrams to identify cardinal conflicts. A *Multi-Valued Decision Diagram* (MDD) (Sharon et al. 2013) MDD_i for a_i is a directed acyclic graph that consists of all shortest paths of a_i that satisfy the constraints. All nodes at depth t in MDD_i correspond to all possible vertices at timestep t in these paths. An MDD node is called a *singleton* (Li et al. 2019) iff it is the only node at its depth. Therefore, a conflict is cardinal iff the MDDs of both conflicting agents contain singletons at the conflicting timestep.

Suboptimal variants: Barer et al. (2014) extended CBS to a bounded-suboptimal MAPF algorithm, called *Enhanced CBS* (ECBS). The bounded suboptimality is achieved by using focal search (Pearl and Kim 1982), instead of best-first search, in both the high- and low-level searches of CBS. A focal search maintains a FOCAL list that contains a subset of nodes in the OPEN list such that the costs of the nodes in FOCAL are within a constant factor of the lowest cost of any node in OPEN. The focal search always expands the node with the best user-provided heuristic in FOCAL. ECBS uses the number of conflicts as heuristics in both the high- and low-level focal searches.

3 Inefficiency of CBS Splitting

CBS splitting (also referred to as *non-disjoint splitting*) is intuitive and prevents the possibility of the same conflict reoccurring. It is *complete*, i.e., every candidate conflict-free plan in the parent CT node is contained in at least one of the child CT nodes, but it is not *disjoint*, i.e., some plans may be contained in both child CT nodes. This might lead to a significant amount of duplicate search effort.

Example 1: Figure 1(a) shows a 2-agent MAPF instance, and Figure 1(b) shows its corresponding CT. At the CT root node, a_1 follows the red arrow, and a_2 follows the solid blue arrow. Their paths conflict at $D2$ at timestep 3. CBS resolves this conflict by generating two child CT nodes N_0 and N_1 with the additional constraint $\overline{\langle a_1, D2, 3\rangle}$ and $\overline{\langle a_2, D2, 3\rangle}$,



(a) (b) Non-disjoint splitting (c) Disjoint splitting

Figure 1: (a) A 2-agent MAPF instance on a 4-neighbor grid. (b) and (c) The corresponding CT of CBS with non-disjoint splitting and with disjoint splitting, respectively. Solid nodes are labeled with their costs, and edges are labeled with the constraints they enforce. Dashed nodes are pruned.

respectively. In node N_0, a_1 waits for one timestep. In node N_1, a_2 changes its path to the dashed blue arrow and then conflicts with a_1 at $C2$ at timestep 2. CBS splits N_1 by generating two child CT nodes N_{10} and N_{11} with the additional constraint $\overline{\langle a_1, C2, 2\rangle}$ and $\overline{\langle a_2, C2, 2\rangle}$, respectively. In node N_{10}, a_1 waits for one timestep. In node N_{11}, a_2 changes its path to the dotted blue arrow and then conflicts with a_1 at $B2$ at timestep 1. After CBS splits N_{11} and generates two child CT nodes N_{110} and N_{111}, it finally realizes that the minimum sum of costs of conflict-free paths is 9.

Although the constraints of nodes N_0, N_{10} and N_{110} are different, they all force a_1 to wait for one timestep and allow a_2 to follow its shortest path. In fact, they all include the **same** pair of paths where a_1 follows the red arrow after waiting at $A2$ for one timestep and a_2 follows the dotted blue arrow. If the paths of other agents also conflict with each other, the search below these three nodes will be very similar (creating duplication of search effort).

4 CBS with Disjoint Splitting

To remedy the inefficiency resulting from CBS splitting, we introduce positive constraints. A *positive constraint* $\langle a_i, v, t\rangle$ forces a_i to be at v at timestep t. As a result, any other agent a_j ($j \neq i$) is prohibited from being at v at timestep t and hence has a negative constraint $\overline{\langle a_j, v, t\rangle}$ implicitly. Thus, when adding a positive constraint, we run the low-level search for every agent whose path violates its corresponding negative constraint. We now introduce *disjoint splitting*. We first choose a conflict $\langle a_i, a_j, v, t\rangle$ and one of the two agents a_k ($k \in \{i, j\}$) to split on. We then create two child CT nodes: one where a_k is prohibited from being at v at timestep t (by adding $\overline{\langle a_k, v, t\rangle}$), and one where a_k is forced to be at v at timestep t (by adding $\langle a_k, v, t\rangle$). Here, any plan can only belong to at most one of the child CT nodes, hence their candidate plans are disjoint.

Example 2: We demonstrate disjoint splitting on agent a_1 for the MAPF instance in Figure 1(a). The corresponding CT is shown in Figure 1(c). At the first conflict, CBS generates CT node N_0 where a_1 cannot be at $D2$ at timestep 3 and CT node N_1 where a_1 must be at $D2$ at timestep 3. In node N_0, a_1 waits for one timestep and thus the sum of costs increases. In node N_1, CBS replans the path of a_2 to follow the dashed

blue arrow and discovers a new conflict at $C2$ at timestep 2. Because a_1 has to reach $D2$ at timestep 3, it has to be at $C2$ at timestep 2. So CT node N_{10} with the additional constraint $\overline{\langle a_1, C2, 2\rangle}$ has no solution and is thus *pruned*. Similarly, when resolving the conflict at $B2$ at timestep 1 in node N_{11}, CBS prunes CT node N_{110} with the additional constraint $\overline{\langle a_1, B2, 1\rangle}$. So, CBS generates only CT node N_{111} with the additional constraint $\langle a_1, B2, 1\rangle$, whose cost is 9. Therefore, the CT has only two leaf nodes instead of four. The same kind of pruning can occur in many other parts of the CT, leading to a significant decrease in the number of CT nodes compared to non-disjoint splitting.

Clearly, disjoint splitting is complete since one of the two constraints must hold for any candidate conflict-free plan in the parent CT node, and it is disjoint since both constraints cannot hold simultaneously for any plan. Also, the tighter positive constraints lead more often to node pruning and thus result in smaller CTs. We summarize these observations in the following proposition.

Proposition 1. *Let Θ be the set of candidate plans in a CT node N with constraints C (i.e., the sets of all plans that satisfy these constraints) and $\Theta^* \subseteq \Theta$ be the set of conflict-free candidate plans in N. Suppose that the plan of N is $\theta \in \Theta$, the chosen conflict is $\langle a_i, a_j, v, t\rangle$ and disjoint splitting splits on a_i. Then, non-disjoint splitting generates two child CT nodes $N_1 \equiv C \cup \{\overline{\langle a_i, v, t\rangle}\}$ and $N_2 \equiv C \cup \{\langle a_j, v, t\rangle\}$, while disjoint splitting generates two child CT nodes N_1 and $N_3 \equiv C \cup \{\langle a_i, v, t\rangle\}$. Let Θ_k be the set of candidate plans in CT node N_k ($k \in \{1, 2, 3\}$). Then (1) $\Theta_1 \cap \Theta_3 = \emptyset$, (2) $\Theta^* \subseteq \Theta_1 \cup \Theta_3$, (3) $\Theta_3 \subseteq \Theta_2$, and (4) $\theta \notin \Theta_k$ ($k \in \{1, 2, 3\}$). Hence, the sets of candidate plans in the child CT nodes after disjoint splitting contain no duplicates, and their union contains all conflict-free candidate plans in N.* □

We next discuss how the addition of positive constraints changes the low- and high-level searches of CBS.

4.1 Low-Level Search

In standard CBS, the low-level search uses time-space A* (Silver 2005) to find a shortest path that satisfies all negative constraints imposed on the agent. While we could still use the same low-level search for CBS with disjoint splitting by regarding each positive constraint $\langle a_i, v, t\rangle$ as a set of negative constraints $\{\overline{\langle a_i, u, t\rangle} \mid u \in V \setminus \{v\}\}$, we introduce a better method. We view the start vertex and all positive constraints as *landmarks*. Since each CT node has only one additional constraint, we only replan the segment of the path between two landmarks or the segment of the path from the last landmark to the goal vertex. (The goal vertex is not a landmark because it can be reached at any timestep.)

4.2 High-Level Search

The two CT branches generated by disjoint splitting are not symmetric since we have not only to select the conflict but also the agent to split on.

We examined several simple and intuitive selection strategies. As in ICBS, we prioritize cardinal conflicts over other conflicts. We select a conflict $\langle a_i, a_j, v, t\rangle$ from all conflicts

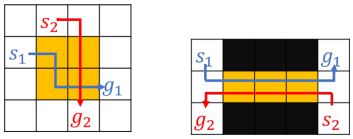


Figure 2: A cardinal-rectangle-conflict instance (left) and a corridor-conflict instance (right).

with the same priority and an agent a_k ($k \in \{i, j\}$) such that: (1) a_k's path involves the most conflicts; (2) a_k has the most constraints imposed; or (3) a_k has the least constraints imposed. However, all these strategies perform empirically similarly to the random strategy Random, which selects a conflict and an agent uniformly at random.

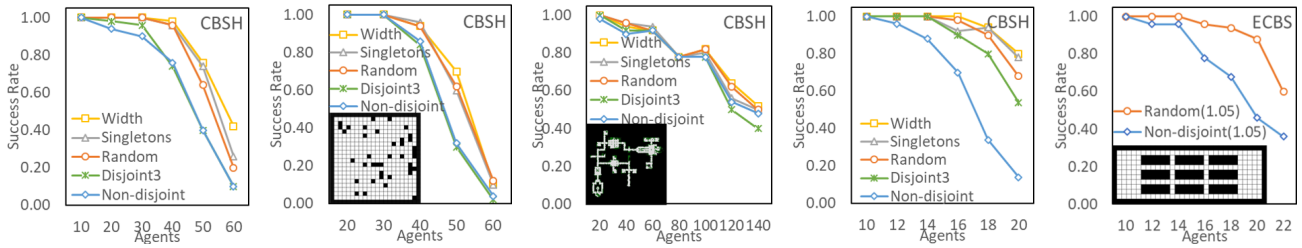
Inspired by the first-fail heuristic (Haralick and Elliot 1980) for constraint satisfaction problems, which picks the most constrained variable, we design the following two selection strategies which select a conflict $\langle a_i, a_j, v, t\rangle$ from all conflicts with the same priority and an agent a_k ($k \in \{i, j\}$) such that: (1) Singletons: MDD_k has the most singletons at depths 1 to t; or (2) Width: MDD_k has the fewest nodes at depth t.

4.3 Related Work

Sharon et al. (2015) already pointed out (in their appendix) that CBS splitting is not disjoint and suggested (but did not implement) 3-way disjoint splitting, that also uses positive and negative constraints, called here Disjoint3. To resolve a conflict $\langle a_i, a_j, v, t\rangle$, Disjoint3 splits a CT node into three child CT nodes with the following additional constraints: C_1: $\{\overline{\langle a_i, v, t\rangle}, \langle a_j, v, t\rangle\}$, C_2: $\{\overline{\langle a_j, v, t\rangle}, \langle a_i, v, t\rangle\}$ and C_3: $\{\overline{\langle a_i, v, t\rangle}, \overline{\langle a_j, v, t\rangle}\}$. Disjoint3 has a larger branching factor than our disjoint splitting and thus could cause redundant search effort. For example, consider a MAPF instance where multiple pairs of agents have conflicts. After CBS has resolved the first conflict, it may have to repeatedly resolve the same second conflict in all child CT nodes. CBS with our disjoint splitting (and even with non-disjoint splitting) has to resolve the second conflict only twice while CBS with Disjoint3 has to resolve it three times. As a result, CBS with Disjoint3 could generate and expand substantially more CT nodes than CBS with our disjoint splitting (and even with non-disjoint splitting). In fact, our disjoint splitting can be understood as merging the second and third child CT nodes into one child CT node with the single additional constraint $\overline{\langle a_j, v, t\rangle}$.

5 Empirical Evaluation

We compare CBSH (a state-of-the-art optimal CBS variant) with non-disjoint splitting and Disjoint3, Random, Singletons and Width disjoint splitting. We also compare ECBS (a state-of-the-art bounded-suboptimal CBS variant) with non-disjoint splitting and Random disjoint splitting. All MAPF instances are on 4-neighbor grids. All algorithms were implemented in C++ and run on a 2.80GHz Intel Core i7-7700 laptop with 8GB RAM and a runtime limit of 5 minutes.



(a) Empty 20×20 grid (b) 10%-blocked 20×20 grid (c) 481×530 game map (d) 10×30 warehouse map (e) 10×30 warehouse map

Figure 3: Success rates (= % of solved instances within 5 minutes) of CBSH and ECBS variants on various maps.

Table 1: The numbers of expanded CT nodes on cardinal-rectangle-conflict and corridor-conflict instances. "Size" and "Length" denote the size and length of the yellow area in the figures in Figure 2, respectively. A suboptimality bound of 1.05 is used for the ECBS variants. We report the number of expanded CT nodes within 5 minutes (with ">" in case of unsolved instances).

CBSH variants on cardinal-rectangle-conflict instances						
Size	4×4	5×5	6×6	7×7	8×8	9×9
CBSH+Non-disjoint	142	1,015	7,447	62,429	573,004	>1,355,201
CBSH+Random	38	110	339	935	2,352	7,757
CBSH and ECBS variants on corridor-conflict instances						
Length	10	12	14	16	18	20
CBSH+Non-disjoint	2,048	8,192	32,768	131,072	524,288	>1,282,531
CBSH+Random	492	1,457	4,373	13,121	39,365	118,097
ECBS+Non-disjoint	1,854	4,332	18,041	88,556	287,220	666,191
ECBS+Random	271	1,024	3,147	8,529	21,365	26,578

Cardinal rectangle conflicts and corridor conflicts: We first experiment on MAPF instances where two agents are involved in two typical types of conflicts that are usually difficult to resolve for CBS: (1) a *cardinal rectangle conflict* (Figure 2(left)) (Li et al. 2019), where all shortest paths of two agents conflict with each other and the optimal solution requires one agent to wait for one timestep, and (2) a *corridor conflict* (Figure 2(right)), where two agents traverse a narrow corridor in opposite directions and the optimal solution requires one agent to wait until the other agent reaches its goal vertex. Table 1 shows the number of expanded CT nodes by non-disjoint splitting and Random. Random reduces the number of CT nodes expanded by CBSH on both types of instances by up to 2 orders of magnitude. It also significantly reduces the number of CT nodes expanded by ECBS on corridor-conflict instances. Random does not speed up ECBS on cardinal-rectangle-conflict instances (not reported) since ECBS finds suboptimal paths at the low level and thus can resolve a cardinal rectangle conflict in a single node expansion.

To show the benefits of the low-level search with landmarks, we also compare the average number of expanded CT nodes per low-level search. The low-level search of CBSH with non-disjoint splitting expands 48 nodes on cardinal-rectangle-conflict instances and 135 nodes on corridor-conflict instances, but only 27 and 22 nodes, respectively, with Random. Similarly, the low-level search of ECBS with non-disjoint splitting expands 148 nodes on corridor-conflict instances but only 108 nodes with Random.

Grids with and without randomly blocked cells: We compare the CBSH variants on an empty 20×20 grid and a 20×20 grid with 10% randomly blocked cells (Figure 3(b)). We experiment on 50 instances with random start and goal vertices for each number of agents and each grid. The success rates and average runtimes are shown in Figures 3(a) and 3(b) and Table 2(a). On both maps, the performance of CBSH with Disjoint3 is similar to that of CBSH with non-disjoint splitting in most cases, while the performance of CBSH with disjoint splitting is substantially better. Most of the speedup is due to disjoint splitting but some of it is due to the low-level search with landmarks. For example, for instances of 10 agents on the empty grid, the average runtime of CBSH with non-disjoint, Width without landmarks and Width with landmarks are 18.38 milliseconds, 1.82 milliseconds and 1.28 milliseconds, respectively. Using landmarks reduces the runtime by 30%. Of the selection strategies, Width is clearly superior in terms of the success rate, although it is sometimes slightly slower than the alternatives. We also compare the ECBS variants, and they perform about the same (not reported). The benefits of disjoint splitting disappear here because the low-level search of ECBS usually has more subpaths between two landmarks to choose from than that of CBSH and hence the positive constraints are less likely to prune nodes and speed up ECBS.

Large game maps: We run the same experiment for the CBSH variants on the benchmark game map `brc202d` (Sturtevant 2012), which is built on a 481×530 grid (Figure 3(c)). The results are shown in Figure 3(c) and Table 2(b). The success rate of CBSH with Disjoint3 is lower than that of CBSH with non-disjoint splitting because the number of agents is larger and three branches thus cause a large amount of overhead to resolve conflicts between different pairs of agents. The performance of CBSH with disjoint splitting is still better than that of CBSH with non-disjoint splitting, although the improvement is not as large as for the 20×20 grids since large maps require more agents before the frequent interactions between multiple agents cause the problem of duplication of search effort to become apparent. We also compute the ECBS variants on this map, and they again perform about the same (not reported).

Warehouse map: Finally, we run the same experiment for the CBSH and ECBS variants on a 10×30 warehouse map (Figure 3(e)). Warehouses represent an important application of MAPF, since robots are essential to the automation

Table 2: Average runtimes (in seconds) of CBSH and ECBS variants on various maps. The runtime limit of 5 minutes is included for unsolved instances. m represents the number of agents. N, D, R, S and W represent CBSH with non-disjoint splitting, Disjoint3, Random, Singletons and Width, respectively. N(1.05) and R(1.05) represent ECBS with a suboptimality bound of 1.05 with non-disjoint splitting and Random, respectively.

(a) 20×20 grids with/without randomly blocked cells

	Empty grid					10%-blocked grid				
m	N	D	R	S	W	N	D	R	S	W
20	18.0	7.6	4.4	3.7	**0.2**	2.1	0.4	**0.05**	0.1	0.06
30	31.9	20.2	1.7	**1.0**	1.2	4.5	6.8	0.7	**0.4**	0.6
40	85.9	82.9	25.5	**15.5**	19.0	57.3	73.5	26.8	**18.4**	28.0
50	203.2	199.2	121.3	95.1	**92.5**	210.1	231.6	158.7	149.5	**139.7**
60	270.9	271.4	249.6	233.0	**187.6**	290.6	296.5	278.5	275.9	**275.5**

(b) 481×530 game map

m	N	D	R	S	W
40	30.1	25.9	19.0	**13.8**	21.5
60	24.1	24.1	24.1	**22.6**	24.1
80	66.5	66.6	66.4	67.5	**66.3**
100	70.9	74.5	59.9	65.5	**59.8**
120	160.7	164.9	132.7	143.5	**128.4**
140	171.1	182.9	**160.5**	175.9	**160.5**

(c) 10×30 warehouse map

m	N	D	R	S	W	N(1.05)	R(1.05)
10	0.2	0.09	0.05	0.05	**0.04**	0.04	**0.01**
12	14.9	1.31	0.4	**0.2**	0.3	13.6	**0.1**
14	48.2	2.03	1.3	**0.9**	**0.9**	23.0	**1.4**
16	133.9	41.6	15.9	28.7	**15.6**	68.8	**13.5**
18	218.2	90.0	63.3	**36.8**	47.5	107.3	**27.5**
20	268.6	181.7	149.0	**103.6**	117.4	170.4	**68.9**

of pick-and-mix warehouses. We simulate this application by forcing half of the agents to move from left to right and half of the agents to move from right to left. All start and goal vertices are randomly located in the left and right 10×5 open areas. The results for the CBSH variants are shown in Figure 3(d) and Table 2(c). Disjoint splitting is highly beneficial for CBSH in terms of both the successes rates and runtimes. CBSH with Singletons often runs the fastest since many paths involve many singletons due to the narrow corridors. The results of the ECBS variants are shown in Figure 3(e) and Table 2(c). Different from the previous experiments, disjoint splitting is now also highly beneficial for ECBS because this map has many narrow corridors and positive constraints can thus prune nodes more frequently.

6 Conclusions

CBS variants are state-of-the-art optimal and bounded-suboptimal MAPF algorithms. But the splitting of CBS is not disjoint and thus creates duplication of search effort. By using positive constraints, we introduced disjoint splitting, that avoids these issues. Empirically, we showed that disjoint splitting is at least as good as CBS splitting and significantly speeds up CBS variants in many cases. Future work will further investigate disjoint splitting and adapt it to more complex settings than 4-neighbor grids with unit edge costs.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *SoCS*, 19–27.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, 740–746.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SoCS*, 29–37.

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

Haralick, R., and Elliot, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019. Symmetry-breaking constraints for grid-based multi-agent path finding. In *AAAI*.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016a. Overview: Generalizations of multi-agent path finding to real-world scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

Ma, H.; Tovey, C.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016b. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI*, 3166–3173.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, 270–272.

Morris, R.; Pasareanu, C.; Luckow, K.; Malik, W.; Ma, H.; Kumar, S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):392–399.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

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

Veloso, M. M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 4423–4429.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–20.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 1444–1449.