Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding*

Jiaoyang Li,[1] Daniel Harabor,[2] Peter J. Stuckey,[2] Hang Ma,[1] Sven Koenig[1]
[1]University of Southern California
[2]Monash University
jiaoyanl@usc.edu, {daniel.harabor,peter.stuckey}@monash.edu, {hangma,skoenig}@usc.edu

Abstract

We describe a new way of reasoning about symmetric collisions for Multi-Agent Path Finding (MAPF) on 4-neighbor grids. We also introduce a symmetry-breaking constraint to resolve these conflicts. This specialized technique allows us to identify and eliminate, in a single step, all permutations of two currently assigned but incompatible paths. Each such permutation has exactly the same cost as a current path, and each one results in a new collision between the same two agents. We show that the addition of symmetry-breaking techniques can lead to an exponential reduction in the size of the search space of CBS, a popular framework for MAPF, and report significant improvements in both runtime and success rate versus CBSH and EPEA* – two recent and state-of-the-art MAPF algorithms.

1 Introduction

Multi-Agent Path Finding (MAPF) is the planning problem of finding a set of paths for a team of agents. Each agent is required to move from an initial start location to a specified goal location, while avoiding conflicts with other agents. A conflict (i.e., collision) happens when two agents stay at the same vertex or traverse the same edge at the same time. Such problems appear in a range of 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 (Silver 2005; Ma et al. 2017).

MAPF is known to be NP-hard on general graphs (Yu and LaValle 2013b; Ma et al. 2016b), planar graphs (Yu 2016) and grids (Banfi, Basilico, and Amigoni 2017). Despite these intractability results and due to the substantial interest in applications, numerous optimal MAPF algorithms have been proposed in recent years. Approaches include reducing MAPF to instances of other well known problems (e.g., multi-commodity flow (Yu and LaValle 2013a), satisfiability (Surynek et al. 2016) and Answer Set Programming (Erdem et al. 2013)); solving MAPF with a single integrated A*-search (Standley 2010; Wagner and Choset 2011; Goldenberg et al. 2014); and solving MAPF with a two-level search (Sharon et al. 2013; 2015; Boyarski et al. 2015; Felner et al. 2018), which constructs a plan by keeping track of constraints between agents at a high level and computing paths consistent with those constraints at a low level, one agent at a time. More detailed surveys are given in (Ma et al. 2016a; Felner et al. 2017).

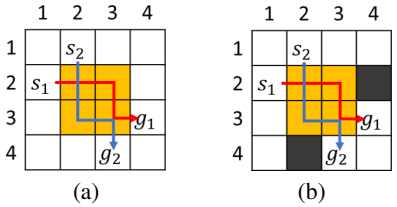


Figure 1: Two situations involving symmetric conflicts between two agents. (a) highlights the problem in general: every shortest path for one agent conflicts with every shortest path for the other agent somewhere in the yellow rectangular area. (b) shows a cardinal conflict, a related class of conflicts which requires that all shortest paths for each agent must pass through a common location at the same timestep: here location $(3, 3)$ at timestep 3.

In this paper, we introduce a new way of reasoning about symmetric conflicts between two agents for MAPF on 4-neighbor grids (which are arguably the most common way of representing the environment for MAPF). Our approach exploits grid symmetries: equivalences between sets of paths or path segments which have the same start and goal locations, the same cost, and which differ only in the order in which grid actions (up, down, left, right, or wait) appear on them. Figure 1 shows two examples. All shortest paths for the two agents conflict somewhere inside the yellow rectangular area. The optimal strategy here is for one agent to wait for the other. We refer to such cases as *cardinal rectangle conflicts*. In this paper, we propose several efficient algorithms to detect cardinal rectangle conflicts as well as two other types of conflicts, semi-cardinal rectangle con-

*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 views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

flicts and non-cardinal rectangle conflicts. We also introduce *barrier constraints* that are able to resolve these rectangle conflicts in a single step and demonstrate, in principle and in practice, that the addition of barrier constraints can achieve an exponential reduction in the number of nodes expanded by Conflict-Based Search (CBS), a popular state-of-the-art MAPF framework.

2 Preliminaries

A MAPF problem is defined by a graph $G = (V, E)$ and a set of m agents $\{a_1, \ldots, a_m\}$. Each agent a_i has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. 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 the tuple $\langle a_i, a_j, v, t\rangle$ a *vertex conflict* iff agents a_i and a_j occupy the same vertex $v \in V$ at the same timestep t, and $\langle a_i, a_j, u, v, t\rangle$ an *edge conflict* iff agents a_i and a_j traverse the same edge $(u, v) \in E$ in opposite directions at the same timestep t. Our task 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 their individual path costs* (SIC). In this paper, graph G is always a 4-neighbor grid whose vertices are unblocked cells and whose edges connect vertices corresponding to adjacent unblocked cells in the four main compass directions.

3 Conflict-Based Search

Conflict-Based Search (CBS) (Sharon et al. 2015) is a two-level search algorithm for MAPF. At the low level, CBS invokes a space-time A* search to find a shortest path for each agent that satisfies some spatio-temporal constraints added by the high level. It break ties by preferring the path that has the fewest conflicts with the paths of other agents. At the high level, CBS performs a best-first search on a binary *constraint tree* (CT). Each CT node contains a set of paths, one for each agent, and also a set of spatio-temporal constraints that are used to coordinate agents and avoid conflicts. The cost of a CT node is the SIC of its current paths. 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 succeeds when the current CT node is conflict-free, which 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 *vertex constraint* $\langle a_i, v, t\rangle$ means that agent a_i is prohibited from occupying vertex v at timestep t. Similarly, an *edge constraint* $\langle a_i, u, v, t\rangle$ means that agent a_i is prohibited from traversing edge (u, v) at timestep t.

Splits: When CBS expands a CT node N, it checks for pairwise conflicts among the current paths. If there are none, then N is a goal CT node and CBS terminates. Otherwise, CBS chooses one of the conflicts and resolves it by *splitting* N into two child CT nodes. In each child CT node, one agent from the conflict is forbidden to use the contested vertex or edge by way of an additional constraint. The path of this agent becomes invalidated and must be replanned by a low-level search. All other paths remain unchanged. With two child CT nodes per conflict, CBS guarantees optimality, exploring both ways of resolving each conflict.

Cardinal, semi-cardinal and non-cardinal conflicts: Boyarski et al. (2015) categorize conflicts into three different types, and they show that prioritizing among conflicts improves performance. The highest priority is given to *cardinal conflicts*, which they define as follows:

> [A conflict] $C = \langle a_i, a_j, v, t\rangle$ is cardinal if all the consistent optimal paths for both [agents] a_i and a_j include vertex v at timestep t.

An example of such a conflict is shown in Figure 1(b). Every possible way of resolving the cardinal conflict $\langle a_1, a_2, (3, 3), 3\rangle$ requires one of the agents to wait for the other or take a detour. That means, when CBS splits on a cardinal conflict, it produces two child CT nodes whose costs are both strictly higher than the current CT node. In this work, we show that there exist other types of conflicts which have the same result when splitting on them but which cannot be detected using the present definition, such as shown in the example in Figure 1(a). We therefore introduce a revised and more general definition:

Definition 1. A conflict C is **cardinal** iff replanning for any agent involved in the conflict increases the SIC.

Once all cardinal conflicts are processed, the next highest priority is given to *semi-cardinal conflicts*, which Boyarski et al. (2015) define as:

> [A conflict] $C = \langle a_i, a_j, v, t\rangle$ is semi-cardinal if all the consistent optimal paths of one agent include vertex v at timestep t, but the other agent has such a path that does not include v at timestep t.

Similarly, we give a revised and more general definition:

Definition 2. A conflict C is **semi-cardinal** iff replanning for one agent involved in the conflict always increases the SIC while replanning for the other agent does not.

Any conflict which is not cardinal or semi-cardinal is said to be **non-cardinal**. These can be processed in any order after the other conflicts, though a popular strategy involves choosing the earliest non-cardinal conflict first.

Admissible heuristics: The high-level of CBS consists of a best-first search that prioritizes for expansion CT nodes with the smallest SIC. Felner et al. (2018) show that the efficiency of the high-level search can be improved through the addition of admissible heuristics. The suggested algorithm, CBSH, proceeds by building a *conflict graph*, whose vertices represent agents and edges represent cardinal conflicts of the current paths. It can be shown that the value of the minimum vertex cover of the conflict graph is an admissible and consistent lower bound on the cost-to-go. The addition of heuristics to the high-level search often produces smaller CTs and decreases the runtime of CBS by a large factor.

4 Inefficiency of CBS and CBSH when Resolving Cardinal Rectangle Conflicts

In this section, we demonstrate how CBS and CBSH resolve cardinal rectangle conflicts, and illustrate the large number

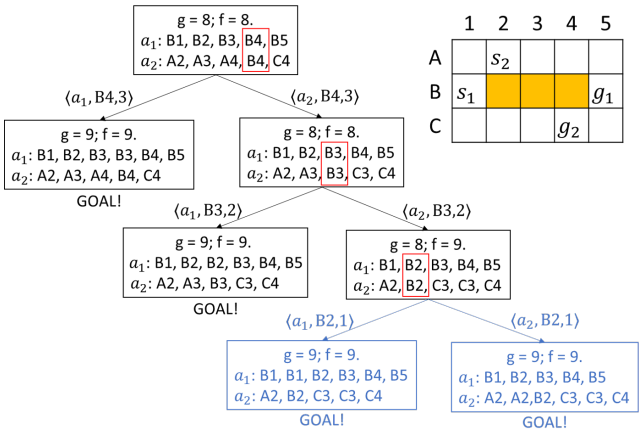


Figure 2: The CT of CBS and CBSH (without the 2 blue CT nodes for CBSH) when resolving a 1×3 cardinal rectangle conflict.

Table 1: Number of CT nodes expanded by CBSH on MAPF instances where 2 agents are involved in one cardinal rectangle conflict. The first column and first row are the width and length of the rectangular area.

	1	2	3	4	5	6	7	8	9
1	1	1	2	3	4	5	6	7	8
2		3	7	14	26	46	79	133	221
3			22	53	116	239	472	904	1,692
4				142	392	1,016	2,651	6,828	17,747
5					1,015	2,971	8,525	23,733	65,236
6						7,447	24,275	78,002	254,173
7							62,429	222,524	795,197
8								573,004	>1,518,151

of CT nodes resulting from it.

Figure 2 illustrates the issue. All shortest paths of agents a_1 and a_2 cross the 1×3 yellow rectangular area. a_1 has 1 shortest path with cost 4 while a_2 has 6 shortest paths with cost 4. Thus, the cost of the root CT node of CBS is 8. However, each of the 6 combinations of these paths has a vertex conflict in one of the yellow cells. Consequently, this is a cardinal rectangle conflict, and the optimal solution cost is 9. The CT of CBS consists of 3 non-goal CT nodes with cost 8 and 4 goal CT nodes with cost 9. The CT of CBSH only saves the last 2 goal CT nodes (in blue).

When the rectangular area is larger, CBS performs worse. Sharon et al. (2015) show that, to resolve the 2×2 cardinal rectangle conflict in Figure 1(a), CBS generates 5 non-goal CT nodes and 6 goal CT nodes (and, CBSH generates 5 CT non-goal nodes and 2 CT goal nodes). To illustrate this issue further, we ran CBSH on MAPF instances where two agents are involved in a cardinal rectangle conflict of different sizes. Surprisingly, the number of expanded CT nodes, as shown in Table 1, is exponential in the length and width of the rectangular area. For a small 8×9 rectangular area, CBSH expands already more than 1 million CT nodes and fails to solve the MAPF instance within 5 minutes.

5 Cardinal Rectangle Reasoning for Entire Paths

In this section, we present a simple algorithm for identifying cardinal rectangle conflicts and introduce a new type of constraints, called barrier constraints, to resolve such conflicts efficiently. We refer to a *node* S as a three-element tuple $(S.x, S.y, S.t)$ corresponding to an agent staying in location $(S.x, S.y)$ at timestep $S.t$. We refer to a *valid path* (or *path* for short) of an agent as a path (sequences of nodes whose locations can repeat and whose timesteps are $0, 1, 2, \dots$) from its start location to its goal location that satisfies its constraints in the CT node but ignores paths of other agents and an *optimal path* of an agent as its shortest valid path.

5.1 Identify Cardinal Rectangle Conflicts

Assume that two agents a_i and a_j have a vertex conflict $\langle a_i, a_j, v, t\rangle$. Let nodes S_i, S_j, G_i and G_j be the corresponding start and goal nodes (Figure 3(a)). We define the *rectangular area* (or *rectangle* for short) as the intersection of the S_i-G_i rectangle and the S_j-G_j rectangle, where S_k-G_k rectangle ($k = i, j$) represents the rectangle whose diagonal corners are in location $(S_k.x, S_k.y)$ and location $(G_k.x, G_k.y)$, respectively. The first two requirements for a cardinal rectangle conflict are intuitive: (1) both agents follow their *Manhattan-optimal* paths, i.e., the cost of each path equals the Manhattan distance from its start node to its goal node, and (2) the distances from each location inside the rectangle to the locations of the two start nodes are equal, which can be simplified to the requirement that both agents move in the same direction in both dimensions (because we already know that the distances from location v to the location of node S_i and the location of node S_j are equal):

$$|S_i.x - G_i.x| + |S_i.y - G_i.y| = G_i.t - S_i.t > 0 \quad (1)$$
$$|S_j.x - G_j.x| + |S_j.y - G_j.y| = G_j.t - S_j.t > 0 \quad (2)$$
$$(S_i.x - G_i.x)(S_j.x - G_j.x) \geq 0 \quad (3)$$
$$(S_i.y - G_i.y)(S_j.y - G_j.y) \geq 0. \quad (4)$$

However, these two requirements do not guarantee that all combinations of optimal paths conflict. Figures 3(b) and 3(c) are two counterexamples where at least one agent has a bypass through which the agent can reach its goal node without entering the rectangle, and thus does not conflict with the other agent. The difference between these two conflicts and the cardinal rectangle conflict in Figure 3(a) is that their goal nodes are located differently compared to their start nodes. Therefore, the third requirement is that the start and goal nodes have opposite relative locations in both dimensions:

$$(S_i.x - S_j.x)(G_i.x - G_j.x) \leq 0 \quad (5)$$
$$(S_i.y - S_j.y)(G_i.y - G_j.y) \leq 0. \quad (6)$$

To sum up, if agents a_i and a_j have a vertex conflict and their corresponding start and goal nodes satisfy Equations (1) to (6), then agents a_i and a_j are involved in a cardinal rectangle conflict.

5.2 Calculate Corner Nodes of the Rectangle

We refer to the four corner nodes of the rectangle as R_s, R_g, R_i and R_j, where R_s and R_g are the corner nodes closest to the start and goal nodes, respectively, and R_i and R_j are the other corner nodes on the opposite borders of S_i and

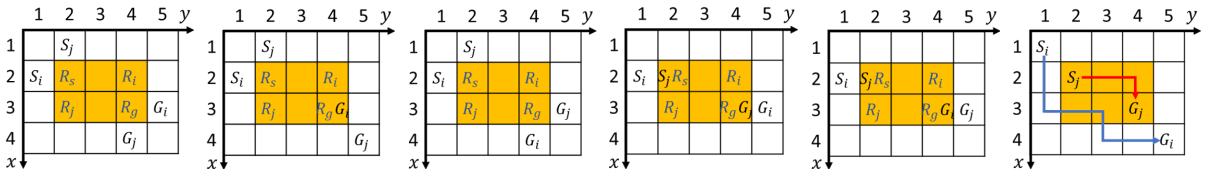


Figure 3: Some examples of rectangle conflicts. The locations of the start and goal nodes are shown in the figures. $G_k.t = S_k.t + |G_k.x - S_k.x| + |G_k.y - S_k.y|, k = i, j$. In (a), (b) and (c), $S_i.t = S_j.t$; in (d) and (e), $S_i.t = S_j.t - 1$; and, in (e), $S_i.t = S_j.t - 2$.

S_j, respectively (Figure 3(a)). The timestep of each node is defined as the timestep when an optimal path of agent a_i or a_j reaches the location of the node. We analyze all combinations of relative locations of start and goal nodes and come up with the following way to calculate them: For the locations of R_s and R_g:

$$R_s.x = \begin{cases} S_i.x, & S_i.x = G_i.x \\ \max\{S_i.x, S_j.x\}, & S_i.x < G_i.x \\ \min\{S_i.x, S_j.x\}, & S_i.x > G_i.x \end{cases} \quad (7)$$

$$R_g.x = \begin{cases} G_i.x, & S_i.x = G_i.x \\ \min\{G_i.x, G_j.x\}, & S_i.x < G_i.x \\ \max\{G_i.x, G_j.x\}, & S_i.x > G_i.x. \end{cases} \quad (8)$$

We can calculate $R_s.y$ and $R_g.y$ by replacing all x by y in Equations (7) and (8). Next, for the locations of R_i and R_j, if $(S_i.x - S_j.x)(S_j.x - R_g.x) \geq 0$, then $R_i.x = R_g.x$, $R_i.y = S_i.y$, $R_j.x = S_j.x$ and $R_j.y = R_g.y$; else, $R_i.x = S_i.x$, $R_i.y = R_g.y$, $R_j.x = R_g.x$ and $R_j.y = S_j.y$. Finally, for the timesteps of all corner nodes R_k $(k = i, j, s, g)$, $R_k.t = S_i.t + |S_i.x - R_k.x| + |S_i.y - R_k.y|$.

5.3 Add Barrier Constraints

Since all combinations of the optimal paths of the agents conflict, we resolve the cardinal rectangle conflict by giving one agent priority within the rectangle and forcing the other agent to *leave* it later or take a detour. To integrate this idea into CBS, we introduce the *barrier constraint*, $B(a_k, R_k, R_g)$ $(k = i, j)$, which is a set of vertex constraints that prohibits agent a_k from occupying all locations along the border of the rectangle that is opposite of its start node (i.e., from R_k to R_g) at the timestep when a_k would optimally reach the location. For example, in Figure 3(a), two barrier constraints are $B(a_i, R_i, R_g) = \{\langle a_i, (2+n, 4), 3+n\rangle | n = 0, 1\}$ and $B(a_j, R_j, R_g) = \{\langle a_j, (3, 2+n), 2+n\rangle | n = 0, 1, 2\}$. $B(a_k, R_k, R_g)$ blocks all possible paths for a_k that reach its goal node G_k via the rectangle, and thus forces a_k to wait or take a detour. When resolving a cardinal rectangle conflict, we generate two child CT nodes and add $B(a_i, R_i, R_g)$ to one of them and $B(a_j, R_j, R_g)$ to the other one. We now present two obvious properties of barrier constraints.

Property 1. *For all combinations of paths of agents a_i and a_j with a cardinal rectangle conflict, if one path violates $B(a_i, R_i, R_g)$ and the other path violates $B(a_j, R_j, R_g)$, then the two paths have one or more vertex conflicts within the rectangle.*

Proof. We assume that the vertex conflict between agents a_i and a_j that underlies the cardinal rectangle conflict is $\langle a_i, a_j, (C.x, C.y), C.t\rangle$. We then assume $S_i.x \leq C.x$ and $S_i.y \leq C.y$ without loss of generality (because the problem is invariant under rotations of axes). According to Equations (1) to (4),

$$\max\{S_i.x, S_j.x\} \leq C.x \leq \min\{G_i.x, G_j.x\} \quad (9)$$

$$\max\{S_i.y, S_j.y\} \leq C.y \leq \min\{G_i.y, G_j.y\} \quad (10)$$

$$(C.x - S_i.x) + (C.y - S_i.y) = (C.x - S_j.x) + (C.y - S_j.y). \quad (11)$$

From Equation (11), we know

$$S_i.x + S_i.y = S_j.x + S_j.y. \quad (12)$$

We can assume that $S_i.x \geq S_j.x$ without loss of generality (because the problem is invariant under swaps of the indexes of agents), which implies $S_i.y \leq S_j.y$. From Equations (9) and (10) and the method for calculating rectangle corner nodes in Section 5.2, we have $R_g.x = \min\{G_i.x, G_j.x\} \geq S_i.x$, $R_g.y = \min\{G_i.y, G_j.y\} \geq S_j.y$, $R_i.x = S_i.x$, $R_i.y = R_g.y$, $R_j.x = R_g.x$ and $R_j.y = S_j.y$. Thus,

$$S_j.x \leq S_i.x = R_i.x \leq R_g.x = R_j.x \quad (13)$$

$$S_i.y \leq S_j.y = R_j.y \leq R_g.y = R_i.y. \quad (14)$$

Consequently, the relative locations of the start, goal and rectangle corner nodes are exactly the same as given in Figure 3(a). For every node N_i on the border R_i-R_g (i.e., $R_i.x \leq N_i.x \leq R_g.x, N_i.y = R_g.y, N_i.t = R_i.t + N_i.x - R_i.x$) and every node N_j on the border R_j-R_g (i.e., $N_j.x = R_g.x, R_j.y \leq N_j.y \leq R_g.y, N_j.t = R_j.t + N_j.y - R_j.y$), we need to prove that the path from S_i to N_i and the path from S_j to N_j have at least one node in common within the rectangle. Since $S_j.x \leq S_i.x \leq N_i.x \leq N_j.x$ and $S_i.y \leq S_j.y \leq N_j.y \leq N_i.y$, the S_i-N_i rectangle and the S_j-N_j rectangle consist of a cross shape, which implies that the path from S_i to N_i and the path from S_j to N_j have at least one location in common within the intersection of the S_i-N_i rectangle and the S_j-N_j rectangle, i.e., this location is within the rectangle. By the definition of rectangle conflicts, the two paths traverse this location at the same timestep, i.e., they have at least one node in common within the rectangle. □

Property 2. *If agents a_i and a_j have a cardinal rectangle conflict, then the cost of any path of agent a_k $(k = i, j)$ that satisfies $B(a_k, R_k, R_g)$ is larger than the cost of an optimal path of agent a_k.*

Proof. We use the same assumptions as in the proof for Property 1. Then, Equations (13) and (14) also hold here. According to Equation (5), $S_i.x = R_i.x \leq R_g.x = G_i.x$. Any optimal path that connects locations $(S_i.x, S_i.y)$ and $(G_i.x, G_i.y)$ has at least one of the nodes $\{(R_i.x + n, R_i.y, R_i.t+n) | n = 0, \ldots, R_g.x - R_i.x\}$. But all of these nodes are constrained by $B(a_i, R_i, R_g)$. Therefore, the cost of any path of agent a_i that satisfies $B(a_i, R_i, R_g)$ is larger than the cost of an optimal path of agent a_i. The proof for $k = j$ can be derived analogously using Equation (6) instead of Equation (5). □

Property 1 is important because CBS requires the constraints added to child CT nodes to not block any conflict-free paths, which is why we add constraints that force an agent to leave the rectangle later rather than enter the it later.

5.4 CBSH-CR

We now present our first algorithm, *CBSH with cardinal rectangle reasoning* (CBSH-CR). It is identical to CBSH except for the following four modifications.

Perform splits: When the chosen conflict is a cardinal rectangle conflict, CBSH-CR adds $B(a_i, R_i, R_g)$ to one child CT node and $B(a_j, R_j, R_g)$ to the other child CT node. Then, in both child CT nodes, the rectangle conflict is resolved by one of the agents increasing its cost.

Classify conflicts: CBSH-CR first classifies vertex/edge conflicts into cardinal, semi-cardinal and non-cardinal conflicts. It then finds cardinal rectangle conflicts among all semi- and non-cardinal vertex conflicts.

Prioritize conflicts: It follows from Definition 1 and Properties 1 and 2 that both cardinal rectangle conflicts and cardinal vertex/edge conflicts are cardinal conflicts. Therefore, CBSH-CR chooses cardinal conflicts first, then semi-cardinal conflicts and last non-cardinal conflicts. It breaks ties by preferring the earliest conflict, where we define $R_s.t$ as the timestep of a cardinal rectangle conflict.

Calculate heuristics: It uses all cardinal conflicts (including cardinal rectangle conflicts) to compute the heuristics for the high-level search.

Now we show that CBSH-CR is complete and optimal.

Lemma 1. *For every cost c, there is a finite number of CT nodes with cost c.*

Proof. The number of conflicts within c timesteps is finite, and, once a conflict is chosen at a CT node N, it never appears again in the subtree of N. Therefore, the number of CT nodes is also finite. □

Theorem 2. *CBSH-CR is complete and optimal.*

Proof. The proof is similar to the proof for the optimality and completeness of CBS (Sharon et al. 2015). The low-level search always returns an optimal path, the high-level search always chooses a CT node with minimum f-value to expand, and the expansion does not lose any conflict-free paths (Property 1). Therefore, the first chosen CT node whose paths are conflict-free has a set of conflict-free paths with minimum SIC (i.e., CBSH-CR is optimal). Besides, the f-value of CT nodes are non-decreasing in expansion order. It follows from Lemma 1 that, if there exist solutions, a solution must be found after expanding a finite number of CT nodes whose costs are no more than the optimal cost (i.e., CBSH-CR is complete). □

6 Rectangle Reasoning for Entire Paths

Reasoning about cardinal rectangle conflicts does not eliminate all symmetric conflicts on grids for CBS. For instance, the conflict in Figure 3(b) is not a cardinal rectangle conflict because agent a_j has an optimal bypass outside of the rectangle. However, if location $(2, 5)$ at timestep 4 and location $(3, 5)$ at timestep 5 are occupied by other agents, whenever the low-level search of CBS replans agent a_j's path, it always returns a path that conflicts with agent a_i's path, because the low-level search uses the number of conflicts with other agents as the tie-breaking rule. Therefore, CBS again generates many CT nodes before finally finding conflict-free paths. We refer to such cases as *semi-cardinal rectangle conflicts*. Similarly, we refer to cases with symmetric conflicts where both agents have bypasses as *non-cardinal rectangle conflicts*, like the case in Figure 3(c). Together with cardinal rectangle conflicts, we refer to these three types of conflicts as *rectangle conflicts*.

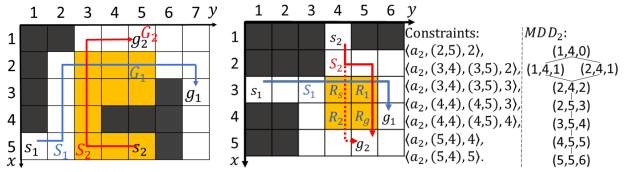
We now show how to identify and classify rectangle conflicts. If agents a_i and a_j have a vertex conflict, then they are involved in a rectangle conflict iff their start and goal nodes satisfy Equations (1) to (4). Moreover, if they also satisfy Equations (5) and (6), it is cardinal; if they also satisfy only one of these equations, it is semi-cardinal; and if they satisfy neither equation, it is non-cardinal. Property 1 holds for all types of rectangle conflicts. It follows from the proof for Property 2 that, when resolving a semi-cardinal rectangle conflict by barrier constraints, at least one of the child CT nodes has to increase its SIC. But for a non-cardinal rectangle conflict, both child CT nodes may not change their SICs.

6.1 CBSH-R

We now introduce the second algorithm, *CBSH with rectangle reasoning* (CBSH-R). It is identical to CBSH-CR except for the following three modifications.

Perform splits: CBSH-R uses barrier constraints to resolve all rectangle conflicts (not only cardinal ones).

Classify conflicts: After classifying all vertex/edge conflicts, CBSH-R checks all semi- and non-cardinal vertex conflicts to identify and classify rectangle conflicts. If a semi-/non-cardinal rectangle conflict has been resolved in one of the ancestors of the current CT node, it ignores this rectangle conflict, otherwise it could always choose to resolve the same rectangle conflict and thus be in a cycle forever. A semi-/non-cardinal rectangle conflict can be found multiple times in a CT branch because its barrier constraint does not disallow all optimal paths that traverse locations inside the rectangle. For example, in Figure 3(c), both agents a_i and a_j have optimal paths that contains node R_s but do not contain nodes that are constrained by $B(a_i, R_i, R_g)$ and $B(a_j, R_j, R_g)$, respectively. So, after adding barrier constraints, a vertex conflict could still happen between two optimal paths within the rectangle and then it is identified as a semi-/non-cardinal rectangle conflict again.



(a) (b)

Figure 4: Rectangle conflicts between path segments. In Figure (b), a_2 follows the red solid arrow but waits at $(1, 4)$ or $(2, 4)$ for one timestep because of constraints.

Prioritize conflicts: CBSH-R uses the same conflict prioritization as CBSH-CR, except that it adds a tie-breaking rule for semi-/non-cardinal rectangle conflicts. Since our reasoning method ignores obstacles and constraints inside the rectangle, it is possible that both child CT nodes increase their costs when CBSH-R resolves a semi-/non-cardinal rectangle conflict using barrier constraints. Therefore, for all semi-cardinal conflicts, it prefers semi-cardinal rectangle conflicts to semi-cardinal vertex/edge conflicts. Similarly, for all non-cardinal conflicts, it prefers non-cardinal rectangle conflicts to non-cardinal vertex/edge conflicts. The secondary tie-breaking rule is still to prefer the earliest conflict, where we define $R_s.t$ as the timestep of a rectangle conflict.

Theorem 3. *CBSH-R is complete and optimal.*

Proof. Since all chosen rectangle conflicts are different in any CT branch, Lemma 1 still holds. Property 1 also holds for semi-/non-cardinal rectangle conflicts. Therefore, we can directly use the proof for Theorem 2 without changes. □

7 Rectangle Reasoning for Path Segments

Our rectangle reasoning methods so far ignore obstacles and constraints, so they can reason only about the rectangle conflicts for entire paths. In some cases, however, rectangle conflicts exist for path segments but not entire paths, such as the cardinal rectangle conflict in Figure 4(a). Since the paths are not Manhattan-optimal, our rectangle reasoning methods so far fail to identify the rectangle conflict. Therefore, in this section, we discuss a rectangle reasoning method for path segments using MDDs.

7.1 Identify Rectangle Conflicts using MDDs

A Multi-Valued Decision Diagram (MDD) (Sharon et al. 2013) MDD_i for agent a_i is a directed acyclic graph that consists of all optimal paths of agent a_i. The nodes at depth t in MDD_i correspond to all possible locations at timestep t in these paths. If MDD_i has only one node (x, y, t) at depth t, we call this node a *singleton*, and all optimal paths of agent a_i traverse location (x, y) at timestep t. CBSH uses singletons in MDDs to classify cardinal, semi-cardinal and non-cardinal vertex/edge conflicts.

MDDs offer information about the impact of obstacles in the grid and constraints imposed on an agent, and thus help us to reason about its path segments. We extend rectangle reasoning to reasoning about rectangle conflicts between two path segments, each of which starts at a singleton (called its start node) and ends at another singleton (called its goal node). If we find a rectangle conflict for a combination of start and goal nodes, we can impose barrier constraints.

Algorithm 1: Identify rectangle conflicts for path segments.

Input: A semi/non-cardinal vertex conflict $\langle a_i, a_j, v, t\rangle$.
// Collect start and goal node candidates.
1 $N_i^S \leftarrow$ singletons in MDD_i no later than timestep t;
2 $N_i^G \leftarrow$ singletons in MDD_i no earlier than timestep t;
3 $N_j^S \leftarrow$ singletons in MDD_j no later than timestep t;
4 $N_j^G \leftarrow$ singletons in MDD_j no earlier than timestep t;
5 $type' \leftarrow$ Not-Rectangle; $area' \leftarrow 0$;
// Try all combinations.
6 **foreach** $S_i \in N_i^S, S_j \in N_j^S, G_i \in N_i^G, G_j \in N_j^G$ **do**
7 **if** *isRectangle*(S_i, S_j, G_i, G_j) **then**
8 $\{R_i, R_j, R_s, R_g\} \leftarrow$ getVertices(S_i, S_j, G_i, G_j);
9 $type \leftarrow$ classifyRect$(R_i, R_j, R_g, S_i, S_j, G_i, G_j)$;
10 $area \leftarrow |R_i.x - R_j.x| \times |R_i.y - R_j.y|$;
11 **if** $type'$ = *Not-Rectangle* or *type is better than* $type'$ or ($type = type'$ and $area > area'$) **then**
12 $type' \leftarrow type$; $area' \leftarrow area$;
13 $\{R_i', R_j', R_s', R_g'\} \leftarrow \{R_i, R_j, R_s, R_g\}$;
14 **if** $type' \neq$ *Not-Rectangle* and *no ancestor CT node has chosen rectangle conflict* $\{R_i', R_j', R_s', R_g'\}$ *before* **then**
15 return $type'$ and $\{R_i', R_j', R_s', R_g'\}$;
16 return Not-Rectangle;

Algorithm 1 shows the pseudo-code. It first treats all singletons as start and goal node candidates (Lines 1-4) and then tries all combinations to find rectangle conflicts. If multiple rectangle conflicts are identified, it prefers one of the highest priority type and breaks ties by preferring a conflict with the largest rectangle area (Line 11). Line 14 prohibits choosing the same rectangle conflict more than once in any CT branch. We discuss details of the three functions on Lines 7, 8 and 9 in Section 7.3.

7.2 Add Modified Barrier Constraints

When reasoning about entire paths, all paths of agent a_i always traverse its start node S_i. However, path segments do not necessarily traverse its start node S_i. In this case, barrier constraints may disallow pairs of conflict-free paths and thus lose the completeness and optimality guarantees.

Figure 4(b) provides a counterexample where a CT node N has the set of constraints listed in the figure. The constraints force agent a_2 to wait for at least one timestep before reaching its goal location. It can either wait before entering the rectangle, which leads to a conflict with agent a_1, or enter the rectangle without waiting and wait later, which might avoid conflicts with agent a_1. However, all optimal paths of agent a_2 in N (whose costs are 6) have to wait for one timestep before entering the rectangle (see MDD_2 shown in the figure). Therefore, node $S_2 = (2, 4, 2)$ is a singleton, and agents a_1 and a_2 have a cardinal rectangle conflict. If this conflict is resolved using barrier constraints, the CT subtree of N disallows the pair of conflict-free paths where

agent a_1 directly follows the blue arrow (which traverses node $(3, 5, 4)$ constrained by $B(a_1, R_1, R_g)$) and agent a_2 follows the dotted red arrow but waits at location $(4, 4)$ for 2 timesteps (which traverses node $(4, 4, 4)$ constrained by $B(a_2, R_2, R_g)$). Barrier constraints fail here because the constrained node $(4, 4, 4)$ is not in MDD_2 and thus agent a_2 could have a path with a larger cost that does not traverse node S_2 but traverses node $(4, 4, 4)$.

Therefore, we add a barrier constraint only for nodes that are in the current MDD of the agent. We call this a *modified barrier constraint* $B'(a_k, R_k, R_g) = \{\langle a_k, (x, y), t\rangle \in B(a_k, R_k, R_g) | (x, y, t) \in MDD_k\}$ $(k = i, j)$, and its properties are discussed in Section 7.3.

7.3 CBSH-RM

Our last algorithm is *CBSH with rectangle reasoning by MDDs* (CBSH-RM), which reasons about rectangle conflicts between path segments. It uses Algorithm 1 to identify and classify rectangle conflicts and uses modified barrier constraints to resolve them.

Previously, all start nodes were at timestep 0, and thus their distances to the rectangle were equal. However, now we allow start nodes to be at different timesteps, e.g., $S_1 = (5, 2, 1)$ and $S_2 = (5, 3, 2)$ in Figure 4(a). We thus need to modify how to identify rectangle conflicts, calculate rectangle corner nodes and classify rectangle conflicts, corresponding to the three functions on Lines 7, 8 and 9 of Algorithm 1, respectively.

Identify rectangle conflicts: The start and goal nodes of a rectangle conflict have to satisfy not only Equations (1) to (4) but also

$$(S_i.x - S_j.x)(S_i.y - S_j.y)(S_i.x - G_i.x)(S_i.y - G_i.y) \leq 0. \tag{15}$$

This guarantees that the start nodes are on different borders of the rectangle since, otherwise, adding modified barrier constraints might lose a pair of paths that allow both agents to reach the constrained border without waiting, such as in the example of Figure 3(f). We also require that $S_i \neq S_j$, otherwise the two agents have a cardinal vertex conflict at node S_i and CBS constraints can resolve it in a single step.

Calculate rectangle corner nodes: The method in Section 5.2 can miscalculate R_i and R_j when $S_i.x = S_j.x$, such as in Figures 3(d) and 3(e). Instead, we calculate R_i and R_j with the following method when $S_i.x = S_j.x$: If $(S_i.y - S_j.y)(S_j.y - R_g) \leq 0$, then $R_i.x = R_g.x$, $R_i.y = S_i.y$, $R_j.x = S_j.x$ and $R_j.y = R_g.y$; otherwise, $R_i.x = S_i.x$, $R_i.y = R_g.y$, $R_j.x = R_g.x$ and $R_j.y = S_j.y$.

Classify rectangle conflicts: Similarly, Equations (5) and (6) misclassify rectangle conflicts when $S_i.x = S_j.x$ or $S_i.y = S_j.y$. Instead, we classify rectangle conflicts using the corner nodes of their rectangles. Since we always add modified barrier constraints along two adjacent borders of the rectangle, we only need to compare the length and width of the rectangle with those of the S_i-G_i and S_j-G_j rectangles. Consider the two equations:

$$R_k.x - R_g.x = S_k.x - G_k.x \tag{16}$$

$$R_k.y - R_g.y = S_k.y - G_k.y. \tag{17}$$

If one holds for $k = i$ and the other one holds for $k = j$, the rectangle conflict is cardinal; if only one of them holds for $k = i$ or $k = j$, it is semi-cardinal; otherwise, it is non-cardinal.

Lemma 4. *If agents a_i and a_j have a rectangle conflict, any path of agent a_k $(k = i, j)$ that traverses a node constrained by $B'(a_k, R_k, R_g)$ also traverses its start node S_k.*

Proof. Let N_k be a node constrained by $B'(a_k, R_k, R_g)$. Thus node N_k is in MDD_k. Then, any node before timestep $N_k.t$ on any path of agent a_k that traverses node N_k is also in MDD_k. Since node S_k is a singleton of MDD_k, any path of agent a_k that traverses node N_k also traverses its start node S_k. □

Property 3. *For all combinations of paths of agents a_i and a_j with a rectangle conflict, if one path violates $B'(a_i, R_i, R_g)$ and the other path violates $B'(a_j, R_j, R_g)$, then the two paths have one or more vertex conflicts within the rectangle.*

Proof. By Lemma 4, we need to prove that any path of agent a_i from its start node S_i to one of the nodes constrained by $B'(a_i, R_i, R_g)$ and any path of agent a_j from its start node S_j to one of the nodes constrained by $B'(a_j, R_j, R_g)$ have at least one node in common within the rectangle. This holds by applying the proof for Property 1 after replacing Equations (11) and (12) by Equation (15) and replacing the method for calculating rectangle corner nodes in Section 5.2 by the method in this section. □

Property 4. *If agents a_i and a_j have a rectangle conflict and one of the Equations* (16) *and* (17) *holds for k $(k = i, j)$, the cost of any path of agent a_k that satisfies $B'(a_k, R_k, R_g)$ is larger than the cost of an optimal path of agent a_k.*

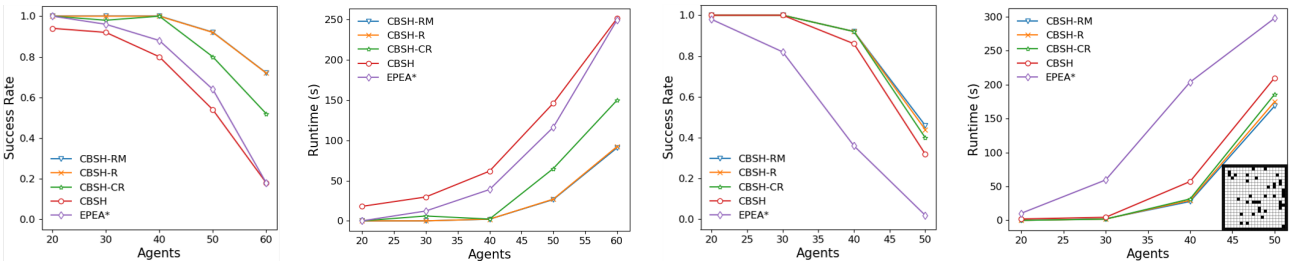
Proof. Since nodes S_k and G_k are singletons, all optimal paths contain these two nodes. If one of Equations (16) and (17) holds, any path from node S_k to node G_k traverses at least one node constrained by $B'(a_k, R_k, R_g)$. So, all optimal paths violate $B'(a_k, R_k, R_g)$. Therefore, the cost of any path of agent a_k that satisfies $B'(a_k, R_k, R_g)$ is larger than the cost of an optimal path of agent a_k. □

Theorem 5. *CBSH-RM is complete and optimal.*

Proof. The proof for Theorem 3 applies after replacing Properties 1 and 2 by Properties 3 and 4, respectively. □

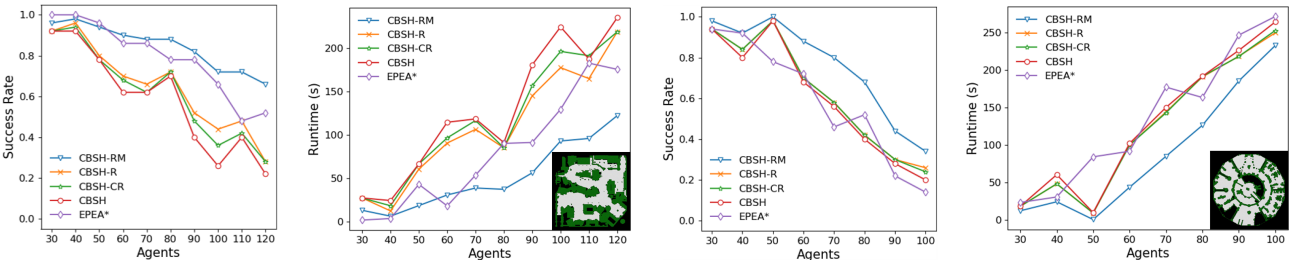
8 Experimental Results

In this section, we compare CBSH-CR, CBSH-R and CBSH-RM with CBSH on grids with randomly blocked cells and benchmark grids. Previous research found that A*-based solvers usually run faster than CBS-based solvers on sparse grids, where many rectangle conflicts exist (Sharon et al. 2015). Therefore, we compare our algorithms also with EPEA* (Goldenberg et al. 2014), a state-of-the-art A*-based solver. Following Boyarski et al. (2015), we enhance EPEA* with Independence Detection (ID) (Standley 2010), which identifies independent groups of agents and runs the solver



(a) Success rate on 0%-blocked (b) Runtime on 0%-blocked (c) Success rate on 10%-blocked (d) Runtime on 10%-blocked

Figure 5: Results on 20×20 grids with 0% and 10% blocked cells. (a) and (c) plot the success rates within 5 minutes. (b) and (d) plot the runtimes, where the runtime limit of 5 minutes is included in the average for unsolved instances. Many parts of the blue lines in (a) and (b) are hidden by the yellow lines.



(a) Success rate on den520d (b) Runtime on den520d (c) Success rate on lak503d (d) Runtime on lak503d

Figure 6: Results on the game grids den520d and lak503d. (a) and (c) plot the success rates within 5 minutes. (b) and (d) plot the runtimes, where the runtime limit of 5 minutes is included in the average for unsolved instances.

Table 2: Results on 20×20 grids. The first "Ins" column shows the number of instances solved by both CBSH and CBSH-RM, and the following columns show results on these instances. Similarly, the second "Ins" column shows the number of instances solved by CBSH-CR, CBSH-R and CBSH-RM, and the following columns show results on these instances.

	m	Ins	Runtime (s)		CT Nodes		Ins	Runtime (s)			CT Nodes		
			CBSH	RM	CBSH	RM		CR	R	RM	CR	R	RM
0%	30	46	6.2	**0.02**	29,506	**87**	49	0.06	0.03	**0.02**	222	89	**82**
	40	40	2.1	**0.02**	10,889	**105**	50	2.2	**2.1**	**2.1**	11,282	11,029	**10,140**
	50	27	14.8	**1.4**	92,627	**5,925**	39	**0.6**	1.6	1.1	**3,454**	6,770	4,327
	60	9	28.4	**2.9**	169,916	**16,194**	26	11.0	9.6	**7.5**	54,300	45,691	**37,210**
10%	20	50	2.1	**0.002**	9,567	**8**	50	**0.002**	**0.002**	**0.002**	10	9	**8**
	30	50	4.5	**2.2**	19,322	**8,702**	50	**1.8**	2.0	2.2	**7,250**	7,962	8,702
	40	43	17.7	**4.4**	96,121	**21,384**	46	8.2	6.0	**4.1**	37,425	28,686	**20,232**
	50	16	19.0	**16.4**	97,553	**79,975**	20	14.4	**11.1**	14.8	70,517	**56,762**	73,624

for each group. We ran experiments on a 2.80 GHz Intel Core i7-7700 laptop with 8 GB RAM with a runtime limit of 5 minutes. For every grid and every number of agents, we average over 50 instances with random start and goal locations.

8.1 Results on Small Grids

Figure 5 presents the success rates and runtimes of all algorithms on a 20×20 empty grid and a 20×20 grid with 10% randomly blocked cells. On both grids, many optimal paths are Manhattan-optimal. As expected, EPEA* runs faster than CBSH on sparse grids (with no blocked cells and few agents). The success rates of EPEA* drop dramatically as grids get denser. The success rates of CBSH, however, has higher success rates on the non-empty grid than the empty grid when the number of agents is at most 40, indicating that rectangle conflicts significantly slow down CBSH on sparse grids. The three new algorithms run significantly faster than CBSH and EPEA* on both grids. In particular, CBSH-R and CBSH-RM perform similarly, and both of them run faster than CBSH-CR, especially on the empty grid with many agents. This observation implies that these instances have many semi- or non-cardinal rectangle conflicts.

Table 2 provides additional details. It first compares CBSH-RM with CBSH by showing their runtimes and numbers of expanded CT nodes on instances solved by both algorithms, i.e., instances that are relatively easy to solve. CBSH-RM wins on both metrics in all cases, by factors of up to three orders of magnitude, and its overhead due to reasoning with rectangle conflicts appears negligible. CBSH-RM improves CBSH by two techniques, using barrier constraints to resolve rectangle conflicts and using cardinal rectangle conflicts to calculate heuristics. In order to see the improvements of the two techniques independently, we also compare CBSH and CBSH-RM with two modified versions of CBSH-RM where one of the two techniques is turned off. The results show that using cardinal rectangle conflicts to calculate heuristics speeds up CBSH, and using barrier constraints to resolve rectangle conflicts speeds up CBSH

more. The combination of the two techniques, i.e., CBSH-RM, runs faster than all of them.

Table 2 also compares CBSH-CR, CBSH-R and CBSH-RM on both metrics on instances solved by all three algorithms. All of them perform similarly. In a few instances, CBSH-CR even expands fewer CT nodes than CBSH-R and CBSH-RM, because our reasoning methods ignore blocked cells and constraints inside the rectangles. So, sometimes rectangle conflicts do not have many symmetries and are faster to solve with CBS constraints than with barrier constraints.

8.2 Results on Large Grids

We also compare the algorithms on two standard benchmark game grids, den520d and lak503d, from (Sturtevant 2012). Figures 6(a) and 6(b) present the success rates and runtimes on map den520d, a 257×256 grid with 28,178 empty cells and 37,614 blocked cells. This grid has a large open space and many large obstacles around the open space. Thus, many optimal paths are not Manhattan-optimal. Therefore, although CBSH-CR and CBSH-R run faster than CBSH, EPEA* runs faster than all of them. However, CBSH-RM, which reasons about rectangle conflicts between path segments, runs faster than CBSH, CBSH-CR and CBSH-R as well as, in most cases, EPEA*.

Figures 6(c) and 6(d) present the success rates and runtimes on map lak503d, a 192×192 grid with 17,953 empty cells and 18,911 blocked cells. This grid also has large open spaces. But it has many narrow corridors as well, which A*-based solvers cannot handle efficiently. EPEA*, CBSH, CBSH-CR and CBSH-R perform similarly, while CBSH-RM runs faster than all of them.

9 Conclusions and Future Work

In this paper, we introduced a new way of reasoning about a special class of symmetric conflicts, called rectangle conflicts, between two agents in grid-based MAPF problems. We demonstrated the poor performance of CBS and CBSH when resolving them. We then proposed three methods, CBSH-CR, CBSH-R and CBSH-RM, for identifying such conflicts and resolving them efficiently. Experimental results showed that all three proposed algorithms improve significantly on CBSH and, among them, CBSH-RM runs the fastest and also runs faster than the A*-based MAPF solver EPEA*.

We suggest the following future research directions: (1) Generalize the symmetry reasoning methods to general graphs; (2) study symmetric conflicts among multiple agents; and (3) apply symmetry reasoning methods to suboptimal MAPF solvers.

References

Banfi, J.; Basilico, N.; and Amigoni, F. 2017. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters* 2(4):1941–1947.

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.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, 290–296.

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, T. K. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 83–87.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research* 50:141–187.

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*.

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.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

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

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, 810–818.

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

Wagner, G., and Choset, H. 2011. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, 3260–3267.

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. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

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

Yu, J. 2016. Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters* 1(1):33–40.