Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

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

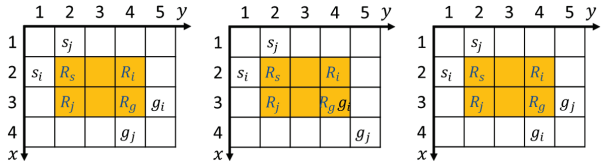
Abstract

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 significantly speed up Conflict-Based Search (CBS), a popular framework for MAPF.

Introduction

A Multi-Agent Path Finding (MAPF) problem is defined by a graph $G = (V, E)$ and a set of agents $\{a_1, \ldots, a_m\}$. Each agent a_i has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. At each discretized 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. 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 path costs, where a conflict is either a *vertex conflict* where two agents occupy the same vertex at the same timestep or an *edge conflict* where two agents traverse the same edge in opposite directions at the same timestep.

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 vertices, 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(a) shows an example. All shortest paths for the two agents conflict somewhere inside the yellow 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 efficient algorithms to detect cardinal rectangle conflicts as well as two other types of conflicts, *semi-cardinal rectangle conflicts* where one of the agents has bypasses that does not traverse the yellow area (e.g., Figure 1(b)) and *non-cardinal rectangle conflicts* where both agents have bypasses (e.g., Figure 1(c)). We also introduce *barrier constraints* to CBS to resolve these rectangle conflicts in a single step while guaranteeing optimality.



(a) Cardinal (b) Semi-cardinal (c) Non-cardinal

Figure 1: Examples of rectangle conflicts. The start and goal vertices are shown in the figures.

Conflict-Based Search (CBS)

CBS (Sharon et al. 2015) is a two-level optimal MAPF algorithm. At the high level, CBS performs a best-first search on a binary *constraint tree* (CT). Each CT node contains a set of spatio-temporal constraints, where a constraint is either a *vertex constraint* $\langle a_i, v, t \rangle$ that prohibits agent a_i from occupying vertex v at timestep t or an *edge constraint* $\langle a_i, u, v, t \rangle$ that prohibits agent a_i from traversing edge (u, v) at timestep t. It also contains a set of shortest paths, one for each agent, that satisfy all constraints. The cost of a CT node is the sum of the path costs. When CBS expands a CT node N, it checks for conflicts among its paths. If there are none, then N is a goal CT node and CBS terminates. Otherwise, CBS chooses a conflict 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 by exploring both ways of resolving each conflict.

CBS is very inefficient when resolving cardinal rectangle

*This paper is a short version of (Li et al. 2019). 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.

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 specify 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

conflicts because it has to try many combinations of shortest paths before realizing that one of the agents has to wait. To illustrate this issue, we ran CBSH (Felner et al. 2018) (an advanced version of CBS) on MAPF instances where two agents are involved in a cardinal rectangle conflict. 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. For the other two types of rectangle conflicts, CBS is also inefficient because it does not always have a good tie-breaking rule for choosing the bypasses to avoid the conflict.

Rectangle Reasoning

In this section, we present approaches to efficiently identify and resolve rectangle conflicts. See Figure 1. We define the *rectangular area* as the intersection of the s_i-g_i rectangle and the s_j-g_j rectangle, where the s_k-g_k rectangle ($k = i, j$) represents the rectangle whose diagonal corners are vertices s_k and g_k. We represent a rectangular area by four spatio-temporal nodes R_s, R_g, R_i and R_j, where the locations of R_s and R_g are the corners of the rectangular area closest to the start and goal vertices, respectively, the locations of R_i and R_j are the other corners on the opposite borders of S_i and S_j, respectively, and the timestep of each node is the timestep when an optimal path of agent a_i or a_j reaches the location of the node.

Identify rectangle conflicts. By observing the examples in Figure 1, the three sufficient conditions for a rectangle conflict are intuitive: (1) the two agents have a vertex conflict; (2) both agents follow their *Manhattan-optimal* paths, i.e., the cost of each path equals the Manhattan distance from its start vertex to its goal vertex, and (3) the distances from each vertex inside the rectangular area to the two start vertices are equal, which can be simplified to the condition that both agents move in the same direction in both dimensions (because the two agents have a vertex conflict inside the rectangular area and thus the distances from the conflicting vertex to their start vertices are equal).

Resolve rectangle conflicts. We resolve a rectangle conflict by giving one agent priority within the rectangular area 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

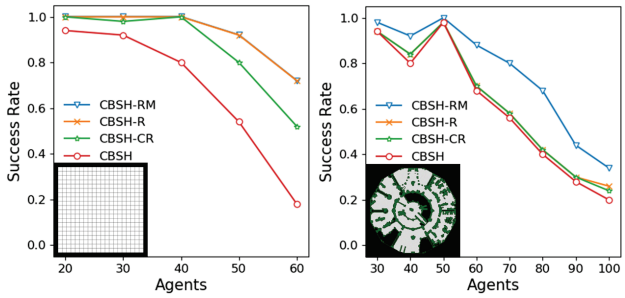


Figure 2: The success rates over 50 instances on a 20×20 grid (left) and a game map (right). The runtime limit is 5 minutes for each instance.

constraints that prohibits agent a_k from occupying any vertex along the border of the rectangular area that is opposite of its start vertex (i.e., from R_k to R_g) at the timestep when agent a_k would optimally reach the vertex. For instance, for every example in Figure 1, the 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 of agent a_k that reach its goal vertex g_k via the exit border of the rectangular area, and thus forces it to wait or take a detour. When resolving a 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. For all combinations of paths of agents a_i and a_j, 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 rectangular area. Therefore, the two barrier constraints added to child CT nodes do not block any conflict-free paths, which guarantees optimality.

Experimental Results

We test our rectangle reasoning techniques on CBSH where (1) CBSH-CR only considers cardinal rectangle conflicts, (2) CBSH-R considers all three types of rectangle conflicts, and (3) CBSH-RM also considers rectangle conflicts between path segments (details are provided in (Li et al. 2019)). Figure 2 presents the success rates on a 20×20 grid and a game map named lak503d from (Sturtevant 2012). All proposed algorithms beat CBSH. CBSH-RM outperforms CBSH-R, which in turn outperforms CBSH-CR.

References

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.

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

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

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