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

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

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 collisions. 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.

Multi-Agent Path Finding (MAPF)

MAPF is the planning problem of finding a set of paths for a team of agents on a given graph. Each agent is required to move from a start location to a goal location, while avoiding collisions with others.

Symmetries in MAPF

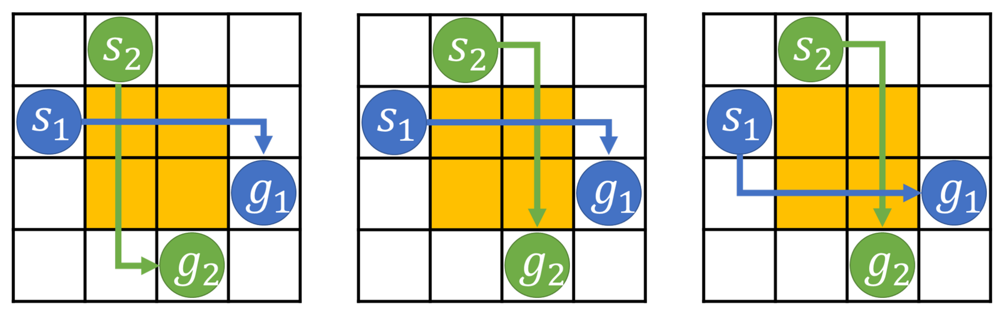


Figure 1: An example of symmetries in MAPF. All shortest paths of the two agents collide somewhere inside the yellow rectangular area. The optimal strategy here is for one agent to wait for the other. We refer to such cases as **cardinal rectangle conflicts**.

CBS is extremely inefficient when resolving cardinal rectangle conflicts.

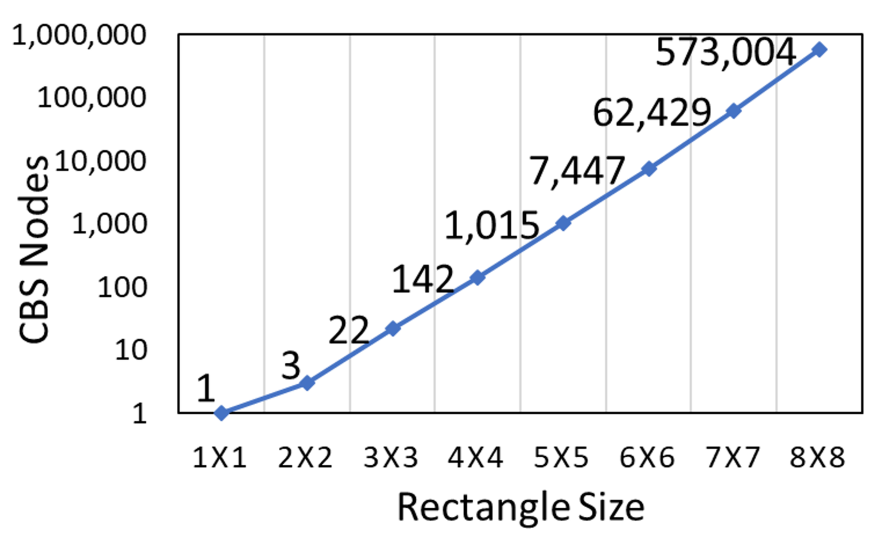


Figure 2: Number of expanded CBS nodes on MAPF instances where 2 agents are involved in one cardinal rectangle conflict.

Conflict-Based Search (CBS) [2]

1. Find a path for every agent independently.
2. Check for conflicts among paths.
3. If there is a conflict where both agent A and agent B stay in location v at timestep t:
   - Option 1: prohibit A from staying in v at t by adding a vertex constraint <A,v,t>.
   - Option 2: prohibit B from staying in v at t by adding a vertex constraint <B,v,t>.
4. Repeat until finding conflict-free paths.

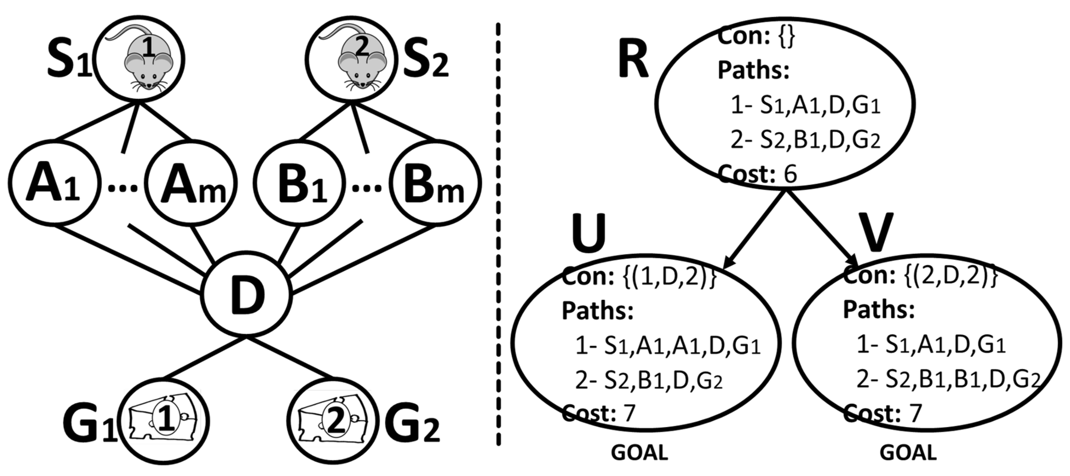


Figure 3: A CBS example reproduced from [1].

* If there are multiple conflicts, we first choose to resolve cardinal conflicts. A conflict is *cardinal* iff replanning for any agent involved in the conflict increases the cost.

Rectangle Conflicts and Barrier Constraints

1. Identify **rectangle conflicts** for entire paths:
   - Shortest paths of both agents are Manhattan-optimal.
   - The two start locations have the same Manhattan distances to any location inside the rectangle.

1*. Identify rectangle conflicts for path segments:
   (1) Find critical locations (i.e., locations that all shortest paths traverse) for both agents.
   (2) Regard these critical locations as the start locations and the goal locations.
   (3) Reason about these locations using a similar method as Step 1.

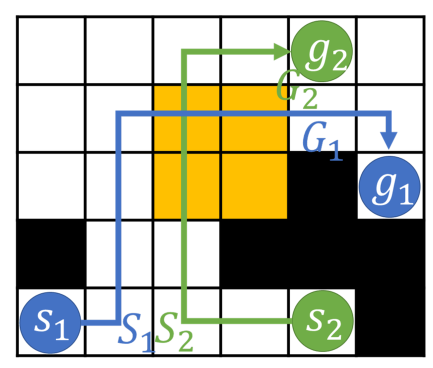
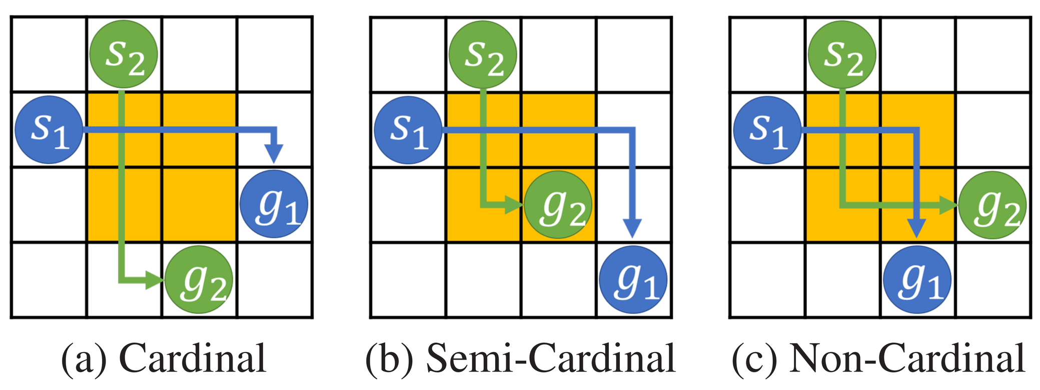


Figure 4: A rectangle conflict for path segments.

2. Classify rectangle conflicts:
   - Cardinal: unavoidable conflict. One of the two agents has to wait.
   - Semi-Cardinal: just one agent can be replanned to avoid the conflict.
   - Non-Cardinal: either agent can be replanned to avoid the conflict.



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

Figure 5: Different types of rectangle conflicts.

*Classification of rectangle conflicts allows us to choose to resolve cardinal conflicts first.

3. Resolve rectangle conflicts:
   - We give one agent priority within the rectangle and force the other agent to *leave* the rectangle later or take a detour by adding barrier constraints.
   - A **barrier constraint** is a set of vertex constraints that prohibits one agent from occupying all locations along the border of the rectangle that is opposite of its start location at the timestep when it would optimally reach it.
   - Adding barrier constraints guarantees the optimality since any pair of conflict-free paths satisfies at least one of the barrier constraint.

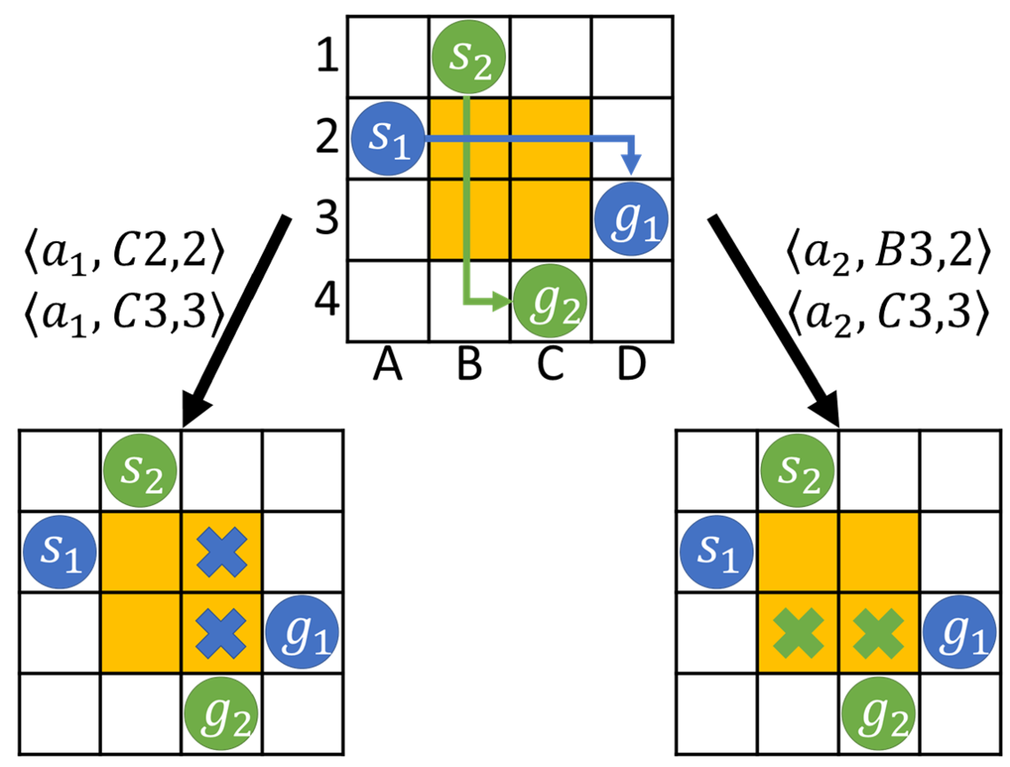


Figure 6: An example of adding barrier constraints. In the left child node, a_2 has priority and a_1 has to wait for one timestep. In the right child node, a_1 has priority and a_2 has to wait for one timestep. Both child nodes have conflict-free paths.

Experimental Results

EPEA*: a state-of-the-art A*-based MAPF algorithm. CBSH: a state-of-the-art CBS variant.
CBSH-RM: CBSH with our rectangle reasoning method.

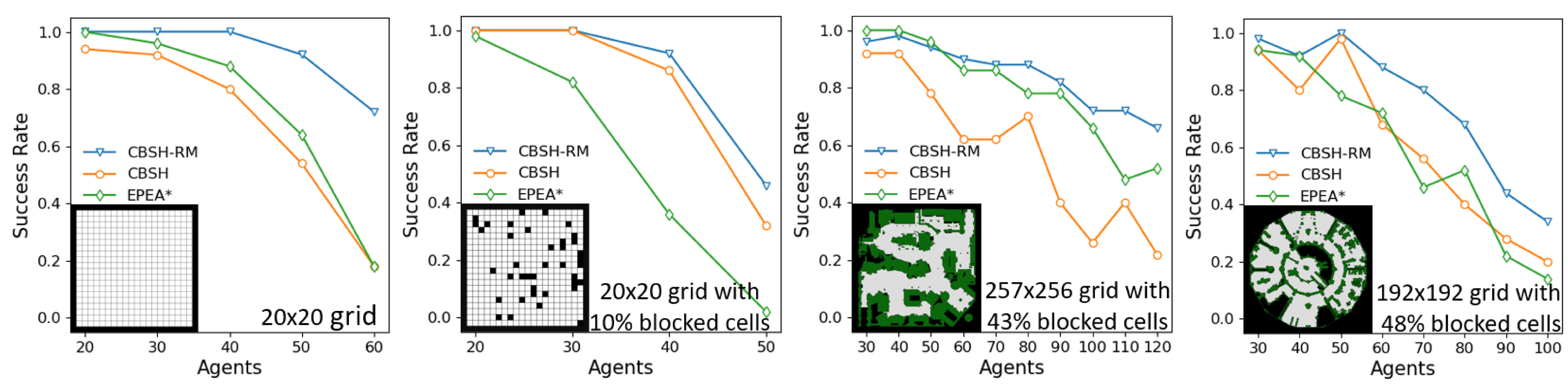


Figure 7: Success rate (=solved instances within 5 minutes) on different grids.

Table 1: Runtime and Number of expanded CBS nodes on 20 × 20 grids with and without blocked cells. Numbers shown in the table are averages over instances solved by both algorithms.

Blocked cells	Agents	Solved instances	Runtime (s)		#CBS Nodes	
			CBSH	CBSH-RM	CBSH	CBSH-RM
0%	30	46	6.2	**0.02**	29,506	**87**
	40	40	2.1	**0.02**	10,889	**105**
	50	27	14.8	**1.4**	92,627	**5,925**
	60	9	28.4	**2.9**	169,916	**16,194**
10%	20	50	2.1	**0.002**	9,567	**8**
	30	50	4.5	**2.2**	19,322	**8,702**
	40	43	17.7	**4.4**	96,121	**21,384**
	50	16	19.0	**16.4**	97,553	**79,975**

[1] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, S. Kumar, and S. Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, pages 83–87, 2018.
[2] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

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.