16891 Project Final Report: Using Non-Blocking Cells to Improve the Scalability of Conflict Based Search

Alvin Zou
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA
azou@andrew.cmu.edu

***Abstract*—Within the spectrum of Multi-Agent Path Finding (MAPF) algorithms, there is often a trade-off between the speed/scalability of the algorithm and the solution optimality/theoretical properties it provides. This work looks at a way of breaking this trade-off through exploiting the properties of the given MAPF instance. The proposed method Decoupled Conflict Based Search (DCBS) identifies a set of non-blocking cells within the graph, assigns priorities to agents based on whether their start/goal states are blocking, and plans agents in groups in descending priority. This effectively decouples the problem into a series of smaller subproblems, which in turn makes the algorithm more scalable while maintaining completeness. Experimental results show that DCBS has faster runtime and better scalability than Priority Based Search and has almost identical solution costs.**

I. Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding feasible paths for multiple agents from their respective starts to goals that do not collide with each other while minimizing some object (e.g makespan). Many algorithms have been developed that aim to solve this problem, with varying optimality and theoretical guarantees. One popular class of solvers is conflict based solvers, first introduced as conflict based search (CBS) [1]. CBS has a two level structure, where the high level search constructs a constraint tree (CT), and at each node there is a list of paths for all agents, the collision between the paths, the sum of cost, and the existing constraints. If there are no collisions, then a valid solution is found. Otherwise, a collision (conflict) between two agents is selected and the node branches, where in each node one constraint is added to one agent. Then the low level planner plans the paths for the two agents respecting the new constraints. CBS is a powerful planner, as it is optimal, and it will find a solution if it exists (but it cannot report the non-existence of solution). However, CBS suffers from the issue of repeatedly replanning agents due to new collisions generated by resolving other conflicts, making it hard to produce solutions in complex environments or scale to many agents.

There has been multiple extensions to CBS. Bounded CBS (BCBS) [2] is a bounded suboptimal variant of CBS that trades off optimality in favor of efficiency. It uses a focal search for the low level and high level planners, speeding up the search with a $w_L w_H$-suboptimal solution. Enhanced CBS (ECBS) [2] improves upon BCBS, producing solutions bounded by a single factor w. Apart from modifications to CBS trading suboptimality for speed, there are also a wide range of techniques that speed up CBS. These include prioritizing conflicts based on the cardinality of the conflict [3], bypassing conflicts to avoid unnecessary branching [4], adding admissible heuristics to the high level search [5] [6], symmetry reasoning techniques to avoid introducing unnecessary collisions [7] [8] [9] [10], disjoint splitting for stronger constraints when resolving a conflict [11], and clustering agents to identify more bypasses [12].

Even with all the existing extensions to CBS, it is still limited in its ability to scale to more complex environments and more agents compared to suboptimal or incomplete methods such as Prioritized Planning (PP), Priority Based Search (PBS), and Lazy Constraints Addition Search for MAPF (LaCAM). However, there is still a trade-off for these algorithms, between the scalability and ability to return solutions quickly versus the theoretical properties it provides and guarantees on solution optimality. I am interested in the question of whether we can develop algorithms that are scalable and have good theoretical properties/return good solutions. To that end, the proposed method attempts to break the trade-off and offers a potential direction of future research.

The proposed method, called Decoupled Conflict Based Search (DCBS), decouples the original MAPF problem into smaller subproblems by exploiting some properties of the MAPF instance. It uses the notion of non-blocking cells, where non-blocking is defined as cells that do not affect the feasibility of the MAPF solution if it is turned into an obstacle. Using this notion, agents can be split into groups based on whether their start and goal states are non-blocking, and can then be planned in groups. This effectively reduces the original problem into multiple subproblems with fewer agents, improving the speed and scalability of the search by sacrificing some solution quality. The experimental results indicate that

DCBS can dramatically improve the scalability of CBS while maintaining similar solution quality.

II. Related Work

Exploiting the properties of the MAPF instance to enhance existing algorithms is not a new idea. [14] introduced the idea of well-formed infrastructure and endpoints, in the context of lifelong MAPF. Endpoints are categorized by task endpoints and non-task endpoints, where task endpoints are all possible start and goal locations for agents and non-task endpoints are locations chosen by the user that satisfy some properties. An instance is considered well-formed if the number of non-task endpoints are greater than the number of agents, for any two endpoints there exists a path between them that traverses no other endpoints, and the tasks are finite. If a lifelong MAPF instance is well formed, then it is solvable and many decentralized approaches are guaranteed to give valid solutions. [16] improves this idea by using standby nodes, a set of nodes determined in real time using articulation point finding algorithms, as temporary endpoints for agents to avoid deadlocks.

Decoupling the original MAPF problem into smaller problems is also not new. [15] proposes an algorithm that finds an execution sequence that minimizes the dimension of the highest-dimensional subproblem over all possible execution sequences, where an execution sequence is defined as an ordered partition of robots into a sequence of composite robots (a set of robots treated as one agent). The composite robots will then execute their plans in sequence. This decomposition reduces the dimensionality of the original problem, and is useful for problems with many robots. [17] groups agents into three priority groups based on whether their start and goal states are non-essential vertices, and solves for each agent group separately, decoupling the problem into three subproblems. The definition of non-essential vertices is similar to how non-blocking cells are defined, but differs from the proposed method via how the problem is decoupled and how priorities are assigned to agents.

III. Terminology and Preliminaries

I consider the classic MAPF problem, where a MAPF instance consists of an undirected graph $W = (V, E)$ abstracted from a 4 connected grid space, and a set of agents $A = \{a_1, ..., a_n\}$ with respective starts $S = \{s_1, ..., s_n\}$ and goals $G = \{g_1, ..., g_n\}$ at different vertex locations. Time is discretized into time steps, and at each time step an agent can either wait at its current location or move to an adjacent vertex denoted by the edges in E. Let $\pi_i(t) \in V$ denote the vertex occupied by agent i at time t. Then the trajectory of agent i can be denoted as $\pi_i = \{\pi_i(0), ..., \pi_i(T)\}$, where $\pi_i(0) = s_i$ and $\pi_i(T) = g_i$. The task is to find a set of valid trajectories $\pi = \{\pi_i, ..., \pi_n\}$ such that any two trajectories from two different agents are collision free, that minimizes the sum of costs of the corresponding joint plan $C = \Sigma_{i=0}^{N} \pi_i$.

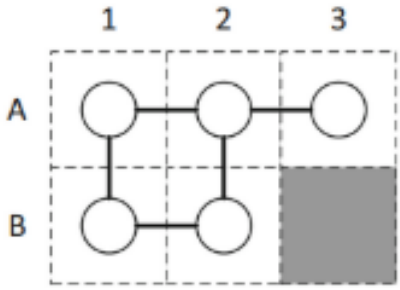

Fig. 1: A simple example to illustrate a bifurcated vertex (A2)

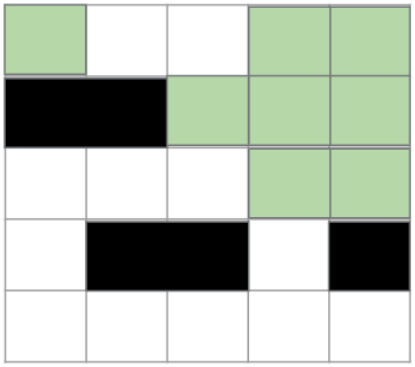

Fig. 2: The non-blocking cells are marked in green.

Definition 1: Connected graph: A connected graph is one where there exists a path connecting any two vertices u and v.

Definition 2: Bifurcated vertex: A bifurcated vertex is a vertex with at least three adjacent vertices.

As shown in Figure 1 (borrowed from [17]), vertex A2 is connected to A1, A3, and B2, and hence is a bifurcated vertex.

Definition 3: Articulation point: a vertex whose removal increases the number of connected components in the graph.

Given a MAPF instance, non-blocking vertices (cells) are defined as follows.

Definition 4: Non blocking vertex: a vertex that is 1) not an articulation point and 2) is not cardinally adjacent to a bifurcated vertex or is cardinally adjacent to two cells that are adjacent to the same bifurcated vertex.

This definition is a sufficient condition for a cell to be non blocking. Condition 1 ensures that turning the cell into an obstacle does not affect the connectivity of the graph (which means the reachability of every agent's goal does not change), and condition 2 ensures that turning the cell into an obstacle does not affect the ability of two agents to swap positions using that cell. An example is illustrated in Figure 2, where the non-blocking cells are marked in green. Note that this definition of non-blocking cells can be further refined. For example, if turning a cell into an obstacle disconnects the graph into two connected components, but all the start/goal pairs of the agents are in the same component, then that cell can still be considered as non-blocking.

Algorithm 1 DECOUPLED CONFLICT BASED SEARCH

Input: MAPF Instance
Output: solution path π

1: **procedure** DCBS
2: $\pi \leftarrow \emptyset$
3: **while** $\pi.size() != num\_of\_agents$ **do**
4: $cells \leftarrow get\_nonblocking\_cells(G)$
5: $agents \leftarrow get\_agent\_group(cells, G)$
6: $\pi \leftarrow \pi \cup get\_paths(agents)$
7: **end while**
8: return π
9: **end procedure**

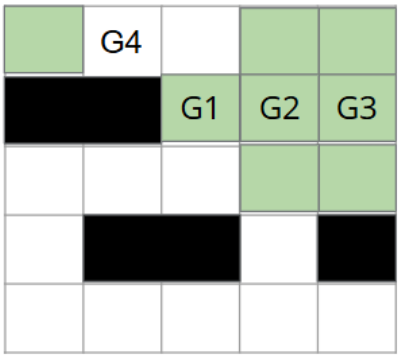

Fig. 3: If agents 1, 2, and 3 are planning to their goals at the same time they will collectively disconnect the graph and potentially cause agent 4 to never be able to reach its goal.

IV. METHOD

DCBS relies on a key observation: given a MAPF instance, there are many cells, called non-blocking cells that can be turned into obstacles without affecting the feasibility of the MAPF instance (i.e a solution still exists). Thus, it is possible to split agents into groups based on if their start and goal cells are non-blocking, and plan for these agents in groups. This will reduce the dimensionality of the original problem, which in turn increases the scalability of the solver.

DCBS has the following structure outlined in Alg. 1: First, it identifies the non-blocking cells in the given map based on its current configuration (line 4, Alg. 1). Next, it assigns agents to different priorities based on their start and goal cells (line 5, Alg. 1). Next, it plans for the agent group with the highest priority and updates the map correspondingly (line 6, Alg. 1). This process is repeated until paths for all agents are found.

A. Identifying Non-Blocking Cells

The procedure to identify non-blocking cells in a given map is outlined in $get\_nonblocking\_cells()$ in Alg. 2. To implement this procedure efficiently, an articulation point finding algorithm like Tarjan's algorithm can be used to check whether the cell disconnects the graph. To quickly check for the other criteria, the 2d grid can be convolved with different kernels.

Algorithm 2 ROUTINES

1: **procedure** GET_NONBLOCKING_CELLS
2: Scan through entire map, marking a cell as non-blocking if it satisfies one of the following criteria:
3: 1. The cell does not disconnect the graph if converted to an obstacle and does not belong to a bifurcated vertex.
4: 2. The cell does not disconnect the graph if converted to an obstacle and it is adjacent to two neighbors that are adjacent to the same bifurcated vertex.
5: return the non-blocking cells
6: **end procedure**

7: **procedure** GET_AGENT_GROUP
8: Assign agent to different priorities based on the following:
9: 1. Priority 0 p_0 to agents with start in blocking cell and goal in blocking cell
10: 2. Priority 1 p_1 to agents with start in blocking cell and goal in non-blocking cell
11: 3. Priority 2 p_2 to agents with start in non blocking cell and goal in non-blocking cell
12: 4. Priority 3 p_3 to agents with start in non blocking cell and goal in blocking cell
13: **if** Agents exist in p_0 **then**
14: For agents in p_0 assign available non-blocking cells as temporary goals and plan paths for these agents.
15: **if** Not enough available non-blocking cells **then**
16: return all agents
17: **end if**
18: **else**
19: return highest priority group
20: **end if**
21: **end procedure**

B. Assigning Agent Priorities

Once the set of non-blocking cells are identified, the agents are assigned priorities based on whether their starts and goals are non-blocking. This process is outlined in $get\_agent\_group()$ in Alg. 2. Agents are assigned higher priority if their goal state is non-blocking, and agents are assigned lower priority if their start state is non-blocking. This is because if an agent can stay at their goal forever (be treated as an obstacle) and not effect the feasibility of the solution for the remaining agents, then it should be planned first. Similarly, if an agent can stay at their start and not effect the feasibility of the solution for the remaining agents, then it should be planned last.

Once priorities are assigned to all agents, the procedure will behave differently based on what priorities exist. There are four cases.

Case 1: There are agents in p_0. In this case, the procedure can not simply return the agents in p_0. This is because the goal states of these agents are blocking, which means if they are planned to their goals and remain there it might

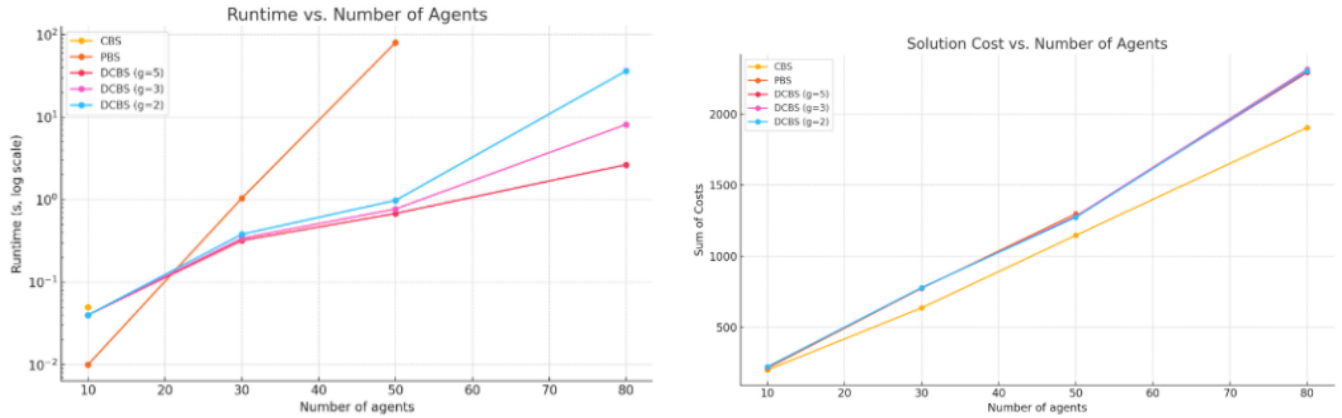

Fig. 4: Runtime and solution cost results for warehouse-20-40-10-2-2

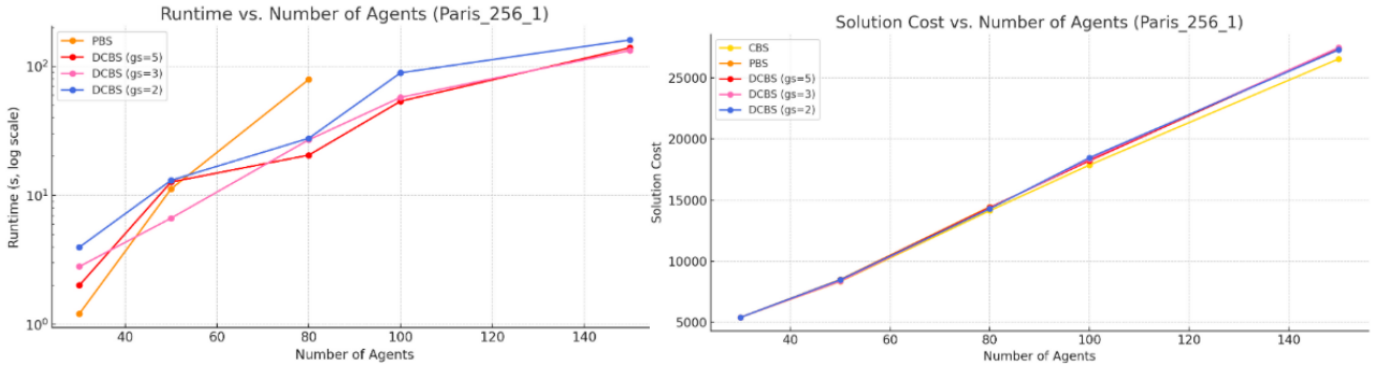

Fig. 5: Runtime and solution cost results for Paris_1_256

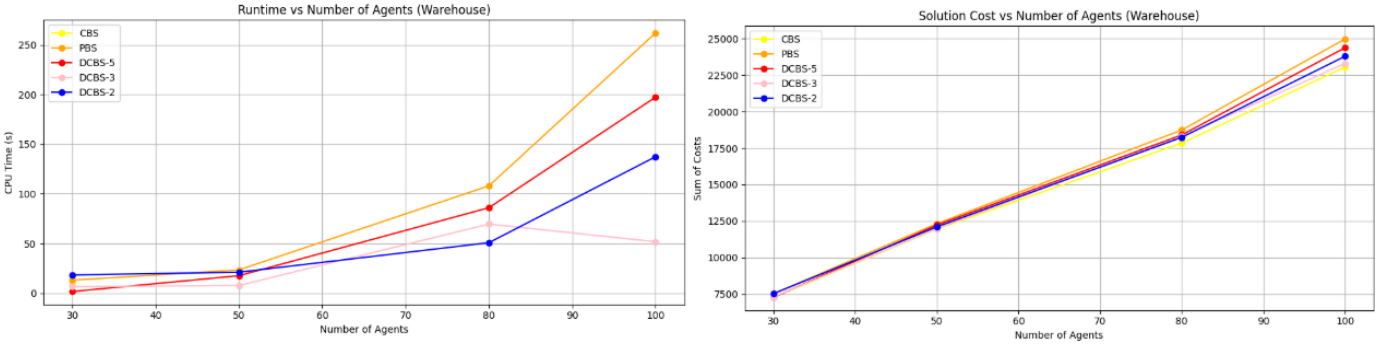

Fig. 6: Runtime and solution cost results for random-32-32-20

make the remaining MAPF problem unsolvable, affecting the completeness of the solver. However, they can not be planned last either, since their start states are also blocking. To resolve this issue, temporary goals are assigned to these agents, where the temporary goals are selected from the set of available non-blocking cells. After the temporary goals are assigned, the paths are planned for these agents, and their starting positions are updated. This effectively moves these agents from p_0 to p_3. However, if there are not enough available non-blocking cells, all the agents are returned to ensure completeness of the algorithm.

Case 2: The highest agent group is p_1. In this case all the agents are returned. This is to ensure the completeness of the algorithm, since agents in p_1 have blocking starts. If only a subset of the agents is returned and planned, a feasible solution might not exist since an agent in p_1 not included in the subset can block the other agents.

Case 3: The highest agent group is p_2. In this case, there is flexibility on what agents to return. It is possible to return all the agents or a certain subset of agents in that priority group. Returning a smaller subset of agents has the advantage of further decoupling the problem and makes it easier to solve

(especially for CBS), but it may result in extra computation for running more iterations for the main loop. Regardless, when deciding which agents to return, there needs to exist some check to ensure that collectively the set of agents staying at their non-blocking goals will not create a blockage and disconnect the graph. Figure 3 illustrates this. A naive way to perform this check is to check if the goals are not adjacent to any other goal and ensure that only agents that satisfy this are returned (in the worst case only one agent is returned).

Case 4: The highest agent group is p_3. In this case all the agents are returned. This is to ensure the completeness of the algorithm, since agents in p_3 have blocking goals, so the all the agents have to be planned at once.

C. Planning Agents

Once the group of agents have been identified, paths are planned for that agent group. Agents whose paths are already planned are treated as dynamic obstacles, while the remaining agents that are not in the current group remain at their start states and are treated as obstacles. However, this introduces some complications, since treating these agents as obstacles affects the layout of the map and in turn might affect the set of non-blocking cells. To ensure consistency of which cells are non-blocking, one method is to define a set of non-blocking cells for each agent treating the other agents as obstacles, but this may be too conservative. Other potential approaches include using some sort of backtracking mechanism or assigning temporary standby nodes (similar to the related work) to agents that are causing the deadlock.

V. Experiments & Results

To test the performance of DCBS, it is compared against CBS/CBSH2-RTC and PBS as baselines. As mentioned previously, there is flexibility on how many agents to return in case 3. Thus, I experimented with returning 2, 3, and 5 agents, denoted by DCBS-2, DCBS-3, and DCBS-5. For the experiments, three maps (warehouse-20-40-10-2-2, Paris_1_256, random-32-32-20) were selected from the Moving AI lab MAPF benchmark [18]. Each map was tested with increasing number of agents, and a timeout of 5 minutes was applied to each planner.

A. warehouse-20-40-10-2-2

The results are averaged over 5 runs and are shown in Figure 4. The map was tested with 10, 30, 50, and 80 agents. CBS failed with 30 agents and CBSH2-RTC failed with 80 agents, and so the solution cost for 80 agents is the cost at timeout, not the optimal solution cost. The results show that DCBS has significantly better runtime then PBS while having nearly identical solution cost, which is also similar to the optimal cost provided by CBS. Comparing DCBS with different group sizes, DCBS-5 performed the best with the fastest runtime. This is likely due to the fact that the reduction in loop iterations out weighed the additional time complexity of solving a subproblem with more agents.

B. Paris_1_256

The results are shown in Figure 5. The map was tested with 30, 50, 80, 100, and 150 agents. Similar trends are observed, where DCBS was successful in all experiments, PBS failed with 100 agents, and CBS failed with 30 agents.

C. random-32-32-20

The results are shown in Figure 6. The map was tested with 30, 50, 80, and 100 agents. DCBS and PBS were successful on all experiments, and CBS failed with 50 agents. However, in this map DCBS-5 performed worse than the other two group sizes. This is likely due to the large map size making the calculations of non-blocking cells more time consuming.

VI. Conclusion

In this paper I introduced a framework named Decoupled Conflict Based Search that exploits the properties of the MAPF instance to increase the scalability of CBS. It iteratively identifies non-blocking cells and groups agents based on whether their start and goal states are non-blocking, then plans for one group of agents at a time. This effectively decouples the original problem into smaller subproblems that are easier to solve, which in turn makes the algorithm more scalable. The experiment results validate that DCBS indeed makes CBS much more scalable, while maintaining similar solution costs.

There are multiple directions which this work can be further explored. Notably, establishing more rigorous definitions for non-blocking cells and deriving proofs for theoretical properties. Other directions include experimenting using the framework with different solvers (not just CBS), generalizing the framework to non-2d grid environments, and trying to further decouple the problem by assigning temporary goals that are non-blocking close to their goals (that are blocking) for agents in p_3.

References

[1] Sharon, Guni, et al. "Conflict-based search for optimal multi-agent pathfinding." Artificial intelligence 219 (2015): 40-66.

[2] 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 International Symposium on Combinatorial Search, 19–27.

[3] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 740–746, 2015.

[4] Eli Boyarski, Ariel Felner, Guni Sharon, and Roni Stern. Don't Split, Try to Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 47-51, 2015.

[5] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 83-87, 2018.

[6] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 442-449, 2019.

[7] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pages 6087-6095, 2019.

[8] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. CoRR, abs/2103.07116, 2021.

[9] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 193-201, 2020.

[10] Han Zhang, Jiaoyang Li, Pavel Surynek, Sven Koenig, and T. K. Satish Kumar. Multi-Agent Path Finding with Mutex Propagation. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 323-332, 2020.

[11] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 279-283, 2019.

[12] Shen, Bojie, et al. "Beyond pairwise reasoning in multi-agent path finding." Proceedings of the International Conference on Automated Planning and Scheduling. Vol. 33. 2023.

[13] Okumura, Keisuke. "Lacam: Search-based algorithm for quick multi-agent pathfinding." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 37. No. 10. 2023.

[14] Čáp, Michal, Jiří Vokřínek, and Alexander Kleiner. "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures." Proceedings of the international conference on automated planning and scheduling. Vol. 25. 2015.

[15] van Den Berg, Jur, et al. "Centralized path planning for multiple robots: Optimal decoupling into sequential plans." Robotics: Science and systems. Vol. 2. No. 2.5. 2009.

[16] Yamauchi, Tomoki, Yuki Miyashita, and Toshiharu Sugawara. "Standby-based deadlock avoidance method for multi-agent pickup and delivery tasks." arXiv preprint arXiv:2201.06014 (2022).

[17] Liao, Bin, et al. "A decoupling method for solving the multi-agent path finding problem." Complex & Intelligent Systems 9.6 (2023): 6767-6780.

[18] Stern, Roni, et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks." Proceedings of the International Symposium on Combinatorial Search. Vol. 10. No. 1. 2019.