Multi-Agent Path Finding with Mutex Propagation

Han Zhang[1], Jiaoyang Li[1], Pavel Surynek[2], Sven Koenig[1], T. K. Satish Kumar[1]
[1]University of Southern California, [2]Czech Technical University
{zhan645, jiaoyanl}@usc.edu, pavel.surynek@fit.cvut.cz, skoenig@usc.edu, tkskwork@gmail.com

Abstract

Mutex propagation is a form of efficient constraint propagation popularly used in AI planning to tightly approximate the reachable states from a given state. We utilize this idea in the context of Multi-Agent Path Finding (MAPF). When adapted to MAPF, mutex propagation provides stronger constraints for conflict resolution in Conflict-Based Search (CBS), a popular optimal MAPF algorithm, and provides it with the ability to identify and reason with symmetries in MAPF. While existing work identifies a limited form of symmetries using rectangle reasoning and requires the manual design of symmetry-breaking constraints, mutex propagation is more general and allows for the automated design of symmetry-breaking constraints. Our experimental results show that CBS with mutex propagation is capable of outperforming CBSH with rectangle reasoning, a state-of-the-art variant of CBS, with respect to runtime and success rate.

1 Introduction

The Multi-Agent Path Finding (MAPF) problem is a generalization of the single-agent path finding problem to multiple agents. Each agent is required to move from a given start vertex to a given goal vertex on a given graph while avoiding conflicts with other agents. A conflict happens when two agents stay at the same vertex or traverse the same edge in opposite directions at the same time. Common objectives for the MAPF problem include minimizing the sum of the path costs and the makespan. Under both objectives, the MAPF problem arises in many real-world application domains, including automated warehouse robots (Wurman, D'Andrea, and Mountz 2008) and aircraft-towing vehicles (Morris et al. 2016).

Conflict-Based Search (CBS) (Sharon et al. 2015) is a popular algorithm for solving the MAPF problem optimally for both objectives, which is known to be NP-hard (Yu and LaValle 2013; Ma et al. 2016). CBS is a two-level MAPF algorithm that starts with an individual minimum-cost path for each agent. On the high level, CBS maintains a Constraint Tree (CT) and lazily resolves conflicts between pairs of agents by adding spatio-temporal constraints to prohibit one of the agents from occupying the conflicting vertex or traversing the conflicting edge at the conflicting timestep. On the low level, CBS finds individual minimum-cost paths for each agent that satisfy the spatio-temporal constraints specified by the high-level node. CBS expands high-level nodes in a best-first order and returns a set of paths as solution when they are conflict-free. Many improvements to CBS have been made, such as adding conflict-selection strategies (Boyarski et al. 2015) and heuristic guidance (Li et al. 2019a).

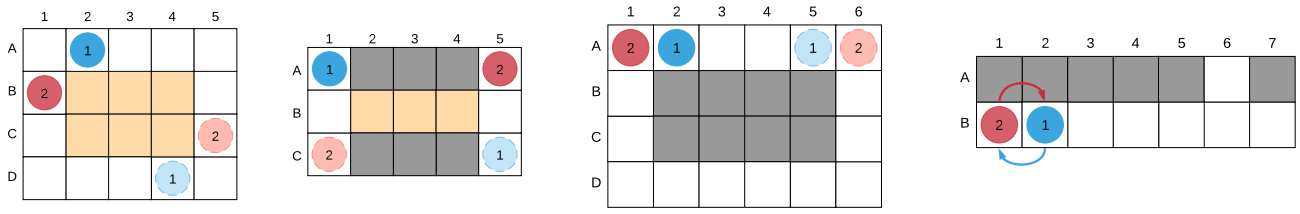
In this paper, we utilize a well-known technique, called *mutex propagation*, from AI planning.[1] It is a form of constraint propagation that corresponds to directed 3-consistency, which in turn is a truncated form of path consistency (Weld 1999). Like all constraint propagation techniques, it makes implicit constraints explicit, and it does so efficiently. In AI planning, mutex propagation is applied to the planning graph (Blum and Furst 1997) to tightly approximate the set of all reachable states from a given state in polynomial time (Weld 1999). It has successfully been used to design reachability heuristics for state-space planners (Nguyen and Kambhampati 2000), design heuristics for plan-space planners that make them competitive with state-space planners (Nguyen and Kambhampati 2001) and improve SAT-based planners (Kautz and Selman 1996).[2]

Elements of the planning graph idea have reappeared in MAPF research in the form of Multi-valued Decision Diagrams (MDDs) (Boyarski et al. 2015). MDDs are constructed for each agent individually and essentially capture reachability information for them. However, they do not capture reachability information for groups of agents. On the other hand, building joint MDDs for groups of agents is computationally prohibitive because the joint space grows exponentially with the number of agents. Knowing that mutex propagation alleviates this dilemma in AI planning, we seek to transfer this technique to MAPF, particularly in the CBS framework.

We show that mutex propagation is beneficial in the CBS framework for its ability to identify and reason about symmetries. While existing work identifies a limited form of

[1]Mutex is short for mutual exclusion.

[2]SAT is short for the Boolean satisfiability problem.



(a) (b) (c) (d)

Figure 1: Shows some examples of cardinal conflicts in the root CT node. The white and yellow cells indicate free cells while the gray cells indicate obstacles. The start vertices of the agents are marked with solid-line circles, and their goal vertices are marked with dashed-line circles. (a) shows a cardinal rectangle conflict where one of the two agents needs to wait for one timestep in every optimal solution (Li et al. 2019c). (b) shows a cardinal corridor conflict where one of the two agents needs to wait until the other agent exits the corridor (Li et al. 2019b; Lam et al. 2019; Li et al. 2020). (c) shows a goal vertex conflict where agent a_2 needs to take the long path in the optimal solution. (d) shows a switching agents conflict where both agents need to move to the rightmost side of the corridor in order to switch their vertices (Sharon et al. 2013).

symmetries and requires the manual design of symmetry-breaking constraints (Li et al. 2019c; 2020), mutex propagation is more general and allows for the automated design of symmetry-breaking constraints, resulting in stronger conflict resolution for CBS that is capable of outperforming CBS with rectangle reasoning, a state-of-the-art variant of CBS, with respect to runtime and success rate. We present experimental results to support our claims.

2 Preliminaries

In this section, we provide background material related to Multi-Agent Path Finding (MAPF), Conflict-Based Search (CBS) and Multi-valued Decision Diagrams (MDDs).

2.1 Multi-Agent Path Finding (MAPF)

The Multi-Agent Path Finding (MAPF) problem has many variants (Stern et al. 2019) and, in this paper, we focus on the variant defined in (Stern et al. 2019) that (1) considers vertex and swapping conflicts, (2) uses the "stay at target" assumption and (3) optimizes the sum of costs. Formally, we define the MAPF problem by an undirected graph $G = (V, E)$ and a set of m agents $\{a_1 \dots a_m\}$. Each agent has a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. In each timestep, an agent either moves to a neighboring vertex or waits at its current vertex. When an agent is at its goal vertex, it can *terminally wait* there, which means the agent waits at its goal vertex forever. Both move and wait actions have unit cost, while terminally waiting at the goal vertex has zero cost. A *path* for an agent is a sequence of move and wait actions that leads it from its start vertex to terminally waiting at its goal vertex. A *sub-path* for an agent is a sequence of actions that leads it from one vertex at a specific timestep to another vertex at a specific timestep. The *cost* of a path is the accumulated cost of all actions in this path. A *vertex conflict* happens when two agents stay at the same vertex simultaneously, and an *edge conflict* happens when two agents traverse the same edge simultaneously in opposite directions. A *solution* is a set of conflict-free paths for all agents. In this paper, we limit our discussion to finding conflict-free paths for all agents while minimizing the *Sum of path Costs* (SoC) of all agents. An *optimal solution* is a solution with minimum SoC.

2.2 Conflict-Based Search (CBS)

Conflict-Based Search (CBS) is a two-level MAPF algorithm. On the high level, CBS maintains a *Constraint Tree* (CT). Each CT node contains a set of constraints and a set of paths, one for each agent, that satisfy all these constraints. The root CT node contains no constraints. The cost of a CT node is the SoC of its paths. On the low level, for each CT node, CBS finds an *individual minimum-cost path* for each agent, that is, a path that has the smallest cost among all paths that satisfy all constraints of the CT node (but might conflict with the other paths). The *individual minimum cost* $l_i^\star$ of agent a_i in a CT node is the cost of its path in the CT node. When expanding a CT node, CBS returns a solution if its paths are conflict-free. Otherwise, CBS picks a conflict and splits the CT node into two child CT nodes, which inherit all constraints of their parent CT node. CBS adds a constraint to each child CT node to prohibit either one or the other of the two agents involved in the conflict to use the conflicting vertex or edge at the conflicting timestep. On the high level, CBS expands nodes in a best-first order. Therefore, the paths of the first expanded CT node with conflict-free paths form an optimal solution.

Constraints: A constraint is a spatio-temporal restriction introduced by CBS to resolve conflicts. A vertex constraint $\langle a_i, t, v\rangle$ prohibits agent a_i from occupying vertex v at timestep t, and an edge constraint $\langle a_i, t, v, v'\rangle$ prohibits agent a_i from moving from vertex v to vertex v', that is, traversing edge $\langle v, v'\rangle$, between timesteps t and $t+1$.

Cardinal conflicts: A conflict in a CT node is cardinal iff any pair of individual minimum-cost paths of both conflicting agents that satisfy all constraints of the CT node has at least one vertex or edge conflict. In other words, there does not exist a pair of conflict-free paths for these two agents with their individual minimum costs, respectively, that satisfy all constraints of the CT node. CBS is inefficient in resolving some cases of cardinal conflicts (e.g., the four cases

in Figure 1) as it needs to check all combinations of paths whose SoC is less than the SoC of an optimal solution, which can lead to a large number of node expansions.

We extend the definition of cardinal conflicts as follows: agents a_i and a_j have a *cardinal conflict within costs* (l_i, l_j) in a CT node iff there does not exist a pair of conflict-free paths for these two agents with costs l_i and l_j, respectively, that satisfy all constraints of the CT node. The definition of cardinal conflicts is identical to the definition of cardinal conflicts within costs $(l_i^\star, l_j^\star)$.

Cardinal rectangle conflicts and barrier constraints: In four-neighbor grid maps, two agents have a cardinal rectangle conflict in a CT node iff (1) all individual minimum-cost paths of both agents cross the same rectangular area; (2) the earliest possible timesteps of both agents reaching each vertex inside the rectangular area are equal; and (3) the directions of both agents moving through the rectangular area are same in both dimensions. Then, any pair of individual minimum-cost paths of both agents has at least one conflict. A barrier constraint is a set of vertex constraints that prohibits one or the other of the two agents from leaving the rectangular area on an individual minimum-cost path. Li et al. (2019c) proved that using barrier constraints to resolve cardinal rectangle conflicts guarantees the optimality and completeness of CBS.

Example 1. *Consider the example in Figure 1a. Agents a_1 and a_2 have a cardinal rectangle conflict, and any pair of their individual minimum-cost paths has at least one vertex conflict in the yellow rectangular area. In any optimal solution, one or the other of the two agents needs to wait for one timestep, and the optimal SoC is 11. The barrier constraint for agent a_1 is $\{\langle a_1, 2, C2\rangle, \langle a_1, 3, C3\rangle, \langle a_1, 4, C4\rangle\}$, and the barrier constraint for agent a_2 is $\{\langle a_2, 3, B4\rangle, \langle a_2, 4, C4\rangle\}$.*

2.3 Multi-Valued Decision Diagrams (MDDs)

A Multi-valued Decision Diagram (MDD) (Sharon et al. 2015; 2013) MDD_i^l for agent a_i in a CT node is a $(l+1)$-level directed acyclic graph that consists of all paths of cost l for agent a_i that satisfy all constraints of the CT node. We assume that l is not smaller than the individual minimum cost of agent a_i. The nodes of MDD_i^l at level t correspond to all possible vertices of agent a_i at timestep t in these paths. At level 0, MDD_i^l has a single *source node* corresponding to agent a_i occupying its start vertex s_i at timestep 0. At level l, MDD_i^l has a single *sink node* corresponding to agent a_i occupying its goal vertex g_i at timestep l. For an MDD node n of MDD_i^l, we use $n.level$ to denote the timestep of n and $n.loc$ to denote the vertex of n. For a directed MDD edge $e = \langle n, n'\rangle$ of MDD_i^l, we define $e.level = n.level$, $e.from = n$ and $e.to = n'$. We use MDD_i when the number of levels of the MDD is not important to the discussion.

3 Mutexes and Mutex Propagation

In this section, we explain the original idea of mutex propagation on planning graphs and generalize it to mutex propagation on MDDs for MAPF.

3.1 Mutex Propagation on Planning Graphs

Planning graphs (Weld 1999) contain two types of nodes, *proposition nodes* and *action nodes*, arranged into levels. Even-numbered levels contain only proposition nodes, while odd-numbered levels contain only action nodes. The zeroth level represents the start state. An edge connects a proposition node to an action node at the next level iff the proposition is a precondition of that action. An edge also connects an action node to a proposition node at the next level iff the proposition is made true by that action. The planning graph represents the effects of parallel actions, but it does so very loosely. To better approximate the set of reachable states, mutex propagation is done on the planning graph using the following rules:

- Two action nodes at level i are mutex iff (1) the effect of one action is the negation of the effect of the other action; (2) one action deletes the precondition of the other action; or (3) there exists a precondition of one action and a precondition of the other action that are mutex at level $i-1$.
- Two proposition nodes at level i are mutex iff (1) one proposition is the negation of the other proposition or (2) all actions at level $i-1$ that achieve one proposition are pairwise mutex with all actions at level $i-1$ that achieve the other proposition.

In the context of MAPF, MDDs are directed and leveled data structures that resemble planning graphs. However, each action has a single precondition because each agent at each timestep can either wait at its current vertex u or traverse some edge $\langle u, v\rangle$, both with the single precondition of the agent being at vertex u at that timestep. Therefore, there is no necessity to represent the action layers explicitly, and a collection of MDDs built individually for each agent can be seen as a special case of a planning graph. Similarly, the mutex propagation rules can also be simplified in the case of MDDs, as explained in Section 3.2, resulting in the following semantics. If two MDD nodes n_i of MDD_i and n_j of MDD_j at level t are mutex, then there do not exist conflict-free sub-paths that move agents a_i and a_j from their start vertices at timestep 0 to the vertices $n_i.loc$ and $n_j.loc$ at timestep t. Since mutex propagation can be done in polynomial time, the set of reachable vertices can be efficiently and tightly approximated, from which useful information can be derived for symmetry breaking and guiding the high-level search of CBS.

3.2 Mutex Propagation on MDDs

We define two types of *initial mutexes* that correspond to vertex and edge conflicts in MAPF, respectively:

- Two MDD nodes n_i and n_j are initial mutex iff n_i and n_j are from MDDs for different agents, $n_i.level = n_j.level$ and $n_i.loc = n_j.loc$.
- Two MDD edges $e_i = \langle n_i, n_i'\rangle$ and $e_j = \langle n_j, n_j'\rangle$ are initial mutex iff e_i and e_j are from MDDs for different agents, $e_i.level = e_j.level$, $n_i.loc = n_j'.loc$ and $n_j.loc = n_i'.loc$.

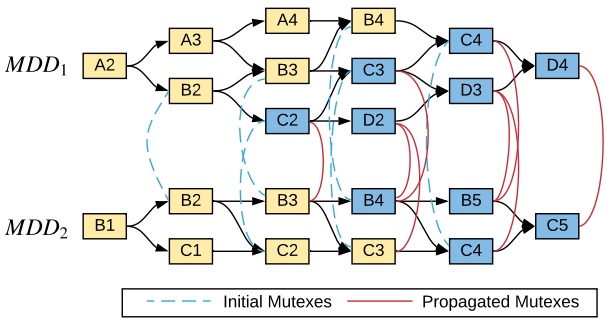


Figure 2: Shows the MDDs for agents a_1 and a_2 on the rectangle conflict of Figure 1a along with the mutexes between their nodes. Initial mutexes are represented with blue dashed arcs, and propagated mutexes are represented with red solid arcs.

Example 2. *Figure 2 shows the MDDs for agents a_1 and a_2 on the rectangle conflict of Figure 1a. The label of each MDD node is its vertex. Initial mutexes are represented with blue dashed arcs.*

At level 1, MDD nodes B2 of MDD_1 and B2 of MDD_2 are initial mutex and thus connected by a blue dashed arc because both agents staying at vertex B2 at timestep 1 causes a vertex conflict.

We define two types of *propagated mutexes* that express our mutex propagation rules:

1. **Forward propagation for MDD nodes:** Two MDD nodes n_i and n_j are propagated mutex iff n_i and n_j are from MDDs for different agents, $n_i.level = n_j.level$ and all pairs of MDD edges e_i and e_j with $e_i.to = n_i$ and $e_j.to = n_j$ are either initial mutex or propagated mutex.
2. **Forward propagation for MDD edges:** Two MDD edges e_i and e_j are propagated mutex iff e_i and e_j are from MDDs for different agents, $e_i.level = e_j.level$ and MDD nodes $e_i.from$ and $e_j.from$ are either initial mutex or propagated mutex.

Example 3. *Propagated mutexes between MDD nodes in Figure 2 are represented with red solid arcs. As MDD nodes B2 of MDD_1 and B2 of MDD_2 are initial mutex, the MDD edges from B2 to C2 of MDD_1 and from B2 to B3 of MDD_2 are propagated mutex. At level 2, MDD nodes C2 of MDD_1 and B3 of MDD_2 have only one incoming MDD edge each, namely, MDD edges from B2 to C2 of MDD_1 and from B2 to B3 of MDD_2. Thus, MDD nodes C2 of MDD_1 and B3 of MDD_2 at level 2 are propagated mutex and connected by a red solid arc.*

We define two MDD nodes or two MDD edges to be *mutex* iff they are initial mutex or propagated mutex. We use Algorithm 1 to find all mutexes between two MDDs. The algorithm is similar to AC-3 (Mackworth 1977). The pseudocode is only for illustrating the general idea and is not intended to be efficient. We add all initial mutexes to a queue and check all mutexes in the order of their levels for whether the mutex can be propagated. The propagated mutexes are then added to the queue. At the same level, we first check mutexes between MDD nodes and then mutexes between MDD edges.

Algorithm 1: GENERATE-MUTEX: Determine all mutexes between two MDDs.

Input : Two MDDs MDD_i and MDD_j.
Output: A set of mutexes M.

```
queue ← all initial mutexes between MDD_i and MDD_j;
M ← ∅;
while queue is not empty do
    m ← pop a mutex from queue with the smallest level,
      breaking ties in favor of node mutexes;
    Add m to M;
    if m is a node mutex then
        ⟨n_i, n_j⟩ ← m;
        foreach e_i such that e_i.from = n_i do
            foreach e_j such that e_j.from = n_j do
                Add ⟨e_i, e_j⟩ to queue;
            end
        end
    else // m is an edge mutex
        ⟨e_i, e_j⟩ ← m;
        n_i ← e_i.to;
        n_j ← e_j.to;
        is_propagated_mutex ← True;
        foreach e'_i such that e'_i.to = n_i do
            foreach e'_j such that e'_j.to = n_j do
                if ⟨e'_i, e'_j⟩ is not in M then
                    is_propagated_mutex ← False;
            end
        end
        if is_propagated_mutex then
            Add ⟨n_i, n_j⟩ to queue;
    end
end
return M;
```

Property 1. *If two MDD nodes n_i of MDD_i and n_j of MDD_j with $n_i.level = n_j.level = l$ are mutex, then there does not exist a pair of conflict-free sub-paths p_i and p_j for agents a_i and a_j, respectively, such that p_i begins at s_i at timestep zero and reaches $n_i.loc$ at timestep l and p_j begins at s_j at timestep zero and reaches $n_j.loc$ at timestep l.*

Proof. The property is trivially true if n_i and n_j are initial mutex. For a proof of the property by contradiction if n_i and n_j are propagated mutex, assume that there exist two such sub-paths p_i and p_j that are conflict-free. Define $n_{i,t}$ as the MDD node that corresponds to the vertex of agent a_i at timestep t when it follows p_i. Similarly, define $n_{j,t}$ as the MDD node that corresponds to the vertex of agent a_j at timestep t when it follows p_j. By definition, $n_{i,0}.loc = s_i$, $n_{j,0}.loc = s_j$, $n_{i,l} = n_i$ and $n_{j,l} = n_j$. Using induction, we now prove the contradiction that n_i and n_j are not propagated mutex. In the base case, $n_{i,0}$ and $n_{j,0}$ are not mutex because $s_i \neq s_j$, given that p_i and p_j are conflict-free. Assume that $n_{i,t}$ and $n_{j,t}$ are not propagated mutex for timestep $t < l$. $n_{i,t}$ and $n_{j,t}$ are not initial mutex because p_i and p_j are conflict-free. We define MDD edge e_i as $\langle n_{i,t}, n_{i,t+1} \rangle$ and MDD edge e_j as $\langle n_{j,t}, n_{j,t+1} \rangle$. e_i and e_j are not initial

Algorithm 2: CLASSIFY-CONFLICT: Determine whether agents a_i and a_j have a cardinal conflict within costs (l_i, l_j).

Input : Two MDDs $MDD_i^{l_i}$ and $MDD_j^{l_j}$ with $l_i \leq l_j$.
Output: The conflict type between agents a_i and a_j, which is PC, AC or NC.

1 $N_j^{'} \leftarrow$ MDD nodes in level l_i of $MDD_j^{l_j}$ that are not mutex with the sink node of $MDD_i^{l_i}$;
2 **if** $N_j^{'} = \emptyset$ **then**
3 | **return** *PC*;
4 **foreach** $n_j \in N_j'$ **do**
5 | **if** *there exists a sub-path in* $MDD_j^{l_j}$ *from* n_j *to its sink node without traversing any MDD node with vertex* g_i **then**
6 | | **return** *NC*;
7 **end**
8 **return** *AC*;

mutex because p_i and p_j are conflict-free. e_i and e_j are not propagated mutex either because their source MDD nodes $n_{i,t}$ and $n_{j,t}$ are not mutex. This implies that $n_{i,t+1}$ and $n_{j,t+1}$ are not propagated mutex. By induction, n_i and n_j are not propagated mutex, which contradicts the assumption. □

Property 2. *If two MDD nodes n_i of MDD_i and n_j of MDD_j with $n_i.level = n_j.level = l$ are not mutex, then there exists a pair of conflict-free sub-paths p_i and p_j for agents a_i and a_j, respectively, such that p_i begins at s_i at timestep zero and reaches $n_i.loc$ at timestep l and p_j begins at s_j at timestep zero and reaches $n_j.loc$ at timestep l.*

Proof. Because n_i and n_j are not mutex, there exists a pair of their incoming MDD edges that are not mutex. Therefore, the source MDD nodes of these two edges are not mutex either. Continuing this backward induction, we can construct the desired conflict-free sub-paths. □

Theorem 1 combines Properties 1 and 2.

Theorem 1. *Iff two MDD nodes n_i of MDD_i and n_j of MDD_j with $n_i.level = n_j.level = l$ are not mutex, then there exists a pair of conflict-free sub-paths p_i and p_j for agents a_i and a_j, respectively, such that p_i begins at s_i at timestep zero and reaches $n_i.loc$ at timestep l and p_j begins at s_j at timestep zero and reaches $n_j.loc$ at timestep l.*

Example 4. *In Figure 2, MDD nodes $D4$ of MDD_1 and $C5$ of MDD_2 at level 5 are propagated mutex. From Theorem 1, there does not exist a pair of conflict-free sub-paths for agents a_1 and a_2 such that both agents arrive at their respective goal vertices at timestep 5, which means that there does not exist a pair of conflict-free paths of cost 5 for a_1 and a_2. Therefore, by definition, the conflict between a_1 and a_2 is cardinal.*

4 Identifying Cardinal Conflicts with Mutex Propagation

In this section, we present an algorithm that uses mutex propagation to identify cardinal conflicts in a CT node.

Two agents with different individual minimum costs can have vertex conflicts after one agent terminally waits at its goal vertex, and these conflicts are not captured by mutexes. To handle them differently, we define two classes of cardinal conflicts:

Pre-goal Cardinal conflict (PC) within costs (l_i, l_j): There does not exist a pair of conflict-free paths with the given costs for both agents from their start vertices to their goal vertices even if we do not consider conflicts that happen after one agent terminally waits at its goal vertex.

After-goal Cardinal conflict (AC) within costs (l_i, l_j): There exists a pair of conflict-free paths with the given costs for both agents from their start vertices to their goal vertices if we do not consider conflicts that happen after one agent terminally waits at its goal vertex. However, for every such pair of paths, one agent traverses the goal vertex of the other agent after the other agent terminally waits at its goal vertex.

Given two agents a_i and a_j and their corresponding MDDs $MDD_i^{l_i}$ and $MDD_j^{l_j}$, we use Algorithm 2 to determine whether these agents have a cardinal conflict within costs (l_i, l_j). Algorithm 2 returns PC, AC or NC ("Not a Cardinal conflict"). Without loss of generality, we assume that $l_i \leq l_j$ throughout this paper. Algorithm 2 first checks whether all MDD nodes of $MDD_j^{l_j}$ at level l_i are mutex with the sink node of $MDD_i^{l_i}$. If so, then it classifies the conflict as a PC. Otherwise, it checks whether there exist an MDD node n_j of $MDD_j^{l_j}$ at level l_i that is not mutex with the sink node of $MDD_i^{l_i}$ and a sub-path in $MDD_j^{l_j}$ from n_j to its sink node that does not traverse any MDD node with vertex g_i. If so, it classifies the conflict as an NC. If such an MDD node and a sub-path do not exist, then it classifies the conflict as an AC.

Theorem 2. *There exists a pair of conflict-free paths p_i and p_j for agents a_i and a_j with costs l_i and l_j, respectively, iff Algorithm 2 returns* NC *given $MDD_i^{l_i}$ and $MDD_j^{l_j}$.*

Proof. First, assume that there exist such conflict-free paths p_i and p_j. From Theorem 1, the MDD node n_j of $MDD_j^{l_j}$ that corresponds to the vertex of agent a_j at timestep l_i is not mutex with the sink node of $MDD_i^{l_i}$. Therefore, N_j' is not empty. Since p_i and p_j are conflict-free, agent a_j following p_j does not traverse vertex g_i at or after timestep l_i. Thus, there exists a sub-path from n_j to its sink node that does not traverse any MDD node with vertex g_i, and Algorithm 2 returns NC.

Now assume that Algorithm 2 returns NC. From Line 5 of Algorithm 2, there exists a sub-path p in $MDD_j^{l_j}$ from an MDD node $n_j \in N_j'$ to its sink node without traversing any MDD node with vertex g_i. From Line 1 of Algorithm 2, n_j is not mutex with the sink node of $MDD_i^{l_i}$. From Theorem 1, there exists a pair of conflict-free sub-paths p_i and p_j for agents a_i and a_j, respectively, such that p_i begins at s_i at

Algorithm 3: GENERATE-CONSTRAINTS-PC: Generate constraints for PCs.

Input : Two MDDs $MDD_i^{l_i}$ and $MDD_j^{l_j}$.
Output: Constraint set C_i for agent a_i and constraint set C_j for agent a_j.

1 $C_i \leftarrow$ constraints on every MDD node of $MDD_i^{l_i}$ that is mutex with all MDD nodes of $MDD_j^{l_j}$ at the same level;
2 $C_j \leftarrow$ constraints on every MDD node of $MDD_j^{l_j}$ that is mutex with all MDD nodes of $MDD_i^{l_i}$ at the same level;
3 **return** $\langle C_i, C_j \rangle$;

Algorithm 4: GENERATE-CONSTRAINTS-AC: Generate constraints for ACs.

Input : Two MDDs $MDD_i^{l_i}$ and $MDD_j^{l_j}$ with $l_i \leq l_j$.
Output: Constraint set C_i for agent a_i and constraint set C_j for agent a_j.

1 $C_i \leftarrow \{$cost constraint $\langle a_i, l_i \rangle\}$;
2 $N_j \leftarrow$ MDD nodes of $MDD_j^{l_j}$ at level l_i that are mutex with the sink node of $MDD_i^{l_i}$;
3 $N_{AC} \leftarrow$ MDD nodes n of $MDD_j^{l_j}$ with $n.loc = g_i$ and $n.level > l_i$;
4 $C_j \leftarrow$ constraints on all MDD nodes in $N_j \cup N_{AC}$;
5 **return** $\langle C_i, C_j \rangle$;

timestep zero and reaches g_i at timestep l_i and p_j begins at s_j at timestep zero and reaches $n_j.loc$ at timestep l_i. If agent a_i follows p_i until timestep l_i and then terminally waits at g_i and agent a_j follows p_j until timestep l_i then follows p to g_j and terminally waits at g_j, then these two paths are conflict-free and of costs l_i and l_j, respectively. □

To apply Theorem 2, l_i and l_j do not have to be the individual minimum costs of agents a_i and a_j, respectively.

Corollary 1. *Agents a_i and a_j have a cardinal conflict iff Algorithm 2 returns* PC *or* AC *given $MDD_i^{l_i^\star}$ and $MDD_j^{l_j^\star}$, where $l_i^\star$ and $l_j^\star$ are the individual minimum costs of agents a_i and a_j, respectively.*

5 Resolving Cardinal Conflicts with Mutex Propagation

In this section, we first present two algorithms that generate constraint sets to resolve PCs and ACs within costs (l_i, l_j), respectively. We then describe how we find suitable l_i and l_j so that the generated constraints are effective for CBS branching.

5.1 Generating the Constraint Sets

Given $MDD_i^{l_i}$ and $MDD_j^{l_j}$ for which CLASSIFY-CONFLICT returns PC, we use Algorithm 3 to generate the constraint sets C_i and C_j for agents a_i and a_j, respectively. We define the constraint on MDD node n of MDD_i as the vertex constraint $\langle a_i, n.level, n.loc \rangle$. Constraint set C_i contains the constraints on every MDD node of $MDD_i^{l_i}$ that is mutex with all MDD nodes of $MDD_j^{l_j}$ at the same level. Similarly, constraint set C_j contains the constraints on every MDD node of $MDD_j^{l_j}$ that is mutex with all MDD nodes of $MDD_i^{l_i}$ at the same level.

Property 3. *For all pairs of paths of agents a_i and a_j with a PC, if a_i's path p_i violates a constraint in C_i and a_j's path p_j violates a constraint in C_j, then paths p_i and p_j are not conflict-free.*

Proof. Let $\langle a_i, t_i, v_i \rangle$ and $\langle a_j, t_j, v_j \rangle$ denote the two constraints violated by paths p_i and p_j, respectively. If $t_i \leq t_j$, use n_j to denote the MDD node in $MDD_j^{l_j}$ corresponding to the vertex of p_j at timestep t_i. From Line 1 of Algorithm 3, the MDD node n_i of $MDD_i^{l_i}$ with $n_i.loc = v_i$ and $n_i.level = t_i$ is mutex with n_j. From Theorem 1, p_i and p_j are not conflict-free. A similar proof works for $t_i \geq t_j$. □

This property holds for paths p_i and p_j of any costs. Similarly, if CLASSIFY-CONFLICT returns AC, we use Algorithm 4 to generate constraint sets. We introduce a new type of constraint.

Cost Constraint: Cost constraint $\langle a_i, l \rangle$ forces the path cost of agent a_i to be larger than l.

Such a cost constraint can be implemented easily by changing the termination condition of the low-level search of CBS.

Given $MDD_i^{l_i}$ and $MDD_j^{l_j}$ for which CLASSIFY-CONFLICT returns AC, we use Algorithm 4 to generate the constraint sets C_i and C_j for agents a_i and a_j, respectively. Constraint set C_i contains only the cost constraint $\langle a_i, l_i \rangle$. Constraint set C_j contains the constraints on all MDD nodes of $MDD_j^{l_j}$ at level l_i that are mutex with the sink node of $MDD_i^{l_i}$ and all MDD nodes n of $MDD_j^{l_j}$ with $n.loc = g_i$ and $n.level > l_i$.

Property 4. *For all pairs of paths of agents a_i and a_j with an AC, if a_i's path cost is not larger than l_i and a_j's path violates a vertex constraint from C_j, then the two paths are not conflict-free.*

Proof. From the calculation of C_j on Line 4 of Algorithm 4, C_j contains the vertex constraints on all MDD nodes in $N_j \cup N_{AC}$. N_{AC} contains all those MDD nodes of $MDD_j^{l_j}$ at levels larger than l_i whose vertices are g_i. If agent a_j violates the constraint on an MDD node n in N_{AC}, it must have a conflict with agent a_i because a_i terminally waits at g_i at timestep $n.level$. N_j contains all those MDD nodes of $MDD_j^{l_j}$ that are mutex with the sink node of $MDD_i^{l_i}$. From Theorem 1, since a_i occupies g_i at timestep l_i and a_j violates the constraint on an MDD node in N_j, the two paths are not conflict-free. □

Properties 3 and 4 ensure that the constraint sets generated by Algorithms 3 and 4 do not rule out any pairs of conflict-free paths for the agents to which the constraint sets are added.

Property 5. *Algorithms 3 and 4 generate constraint sets that increase the individual minimum costs of agents to which these constraint sets are added.*

Proof. If Algorithm 2 outputs PC, then $C_k, k \in \{i, j\}$, contains constraints on all MDD nodes of $MDD_k^{l_k}$ at level l_i because the sink node of $MDD_i^{l_i}$ is mutex with all MDD nodes of $MDD_j^{l_j}$ at level l_i. Therefore, any path of agent a_k satisfying C_k must have cost of at least $l_k + 1$.

If Algorithm 2 outputs AC, then C_i contains only one cost constraint $\langle a_i, l_i \rangle$. Therefore, any path of agent a_i satisfying C_i must have cost of at least $l_i + 1$. For agent a_j, C_j contains constraints on all MDD nodes in N_j and N_{AC}. We prove by contradiction that there does not exist a path for agent a_j of cost less than or equal to l_j. Assume that such a path p exists. We use N'_j to denote the set of nodes of $MDD_j^{l_j}$ at level l_i that are not mutex with the sink node of $MDD_i^{l_i}$. From Theorem 1, p must traverse a vertex that corresponds to an MDD node in N'_j at timestep l_i. Because Algorithm 2 outputs AC, from Line 5, there does not exist a sub-path in $MDD_j^{l_j}$ from any MDD node in N'_j to the sink node of $MDD_j^{l_j}$ that does not traverse an MDD node in N_{AC}. Therefore, such a path p does not exist. □

Property 5 shows that, in every child CT node generated with newly-added constraints from Algorithms 3 or 4, the individual minimum cost of at least one agent increases. Therefore, the SoC of that child CT node is larger than the SoC of its parent CT node.

Properties 3-5 show that using Algorithms 3 and 4 to generate constraints for cardinal conflicts guarantees the optimality and completeness of CBS, since the proof of Theorem 2 in (Li et al. 2019c) applies. Moreover, Properties 3-5 do not rely on l_i and l_j being the individual minimum costs of a_i and a_j, respectively. Therefore, we can pick any l_i and l_j as long as a_i and a_j have a cardinal conflict within (l_i, l_j).

In practice, to keep the sizes of the constraint sets C_i and C_j small (which could reduce the runtime of the low-level search of CBS), we remove the constraints on all such MDD nodes n from the constraint set C_k, $k \in \{i, j\}$, if the constraints on all of n's predecessors in the MDD are also in C_k. Such constraints are redundant because the agent cannot reach $n.loc$ at timestep $n.level$.

Example 5. *In Figure 2, the constraints generated by Algorithm 3 are those of the MDD nodes which are filled with solid blue. After removing redundancies, the constraint set for agent a_1 contains constraints $\langle a_1, 2, C2 \rangle$, $\langle a_1, 3, C3 \rangle$ and $\langle a_1, 4, C4 \rangle$, while the constraint set for agent a_2 contains constraints $\langle a_2, 3, B4 \rangle$ and $\langle a_2, 4, C4 \rangle$. These two constraint sets are exactly the barrier constraints for this cardinal rectangle conflict.*

5.2 Determining the Numbers of Levels of MDDs

For some cardinal conflicts, the minimum SoC of conflict-free paths for the two conflicting agents is much larger than the sum of their individual minimum costs. If we generate constraints using MDDs for the conflicting agents whose numbers of levels are the respective individual minimum costs, CBS still needs to expand multiple CT nodes to resolve all conflicts between the conflicting agents. An example is a corridor conflict where, in any optimal solution, one agent needs to wait for a certain number k of timesteps to allow the other agent to traverse the corridor. Since Algorithms 3 and 4 use MDDs whose numbers of levels are the respective individual minimum costs, the constraints generated by them can increase the cost of either agent by only one. Therefore, CBS needs to increase the CT to a depth of at least k to find an optimal solution. Without heuristic guidance, CBS thus needs to expand $\omega(2^k)$ CT nodes.

To resolve cardinal conflicts efficiently when agents need to increase their sum of individual minimum costs by more than one, we aggressively increase the numbers of levels of MDDs before using them to generate the constraint sets. As long as Algorithm 2 classifies two MDDs as AC or PC, we can apply Algorithms 3 and 4. Although there are several ways of determining suitable MDDs for generating the constraint sets, we adopt Algorithm 5 as our overall constraint generation algorithm. We begin with $d_i = d_j = 0$ and increase d_i and d_j simultaneously until CLASSIFY-CONFLICT($MDD_i^{l_i^\star+d_i+1}$, $MDD_j^{l_j^\star+d_j+1}$) returns NC. We then increase only d_i until CLASSIFY-CONFLICT($MDD_i^{l_i^\star+d_i+1}$, $MDD_j^{l_j^\star+d_j}$) returns NC. Finally, MDDs $MDD_i^{l_i^\star+d_i}$ and $MDD_j^{l_j^\star+d_j}$ are used to generate the constraint sets on Lines 10-14. We leave it to future work to

Algorithm 5: GENERATE-CONSTRAINTS-C: Generate constraints for cardinal conflicts.

Input : Two agents a_i and a_j with $l_i^\star \leq l_j^\star$ and CLASSIFY-CONFLICT($MDD_i^{l_i^\star}$, $MDD_j^{l_j^\star}$) $\neq$ NC.
Output: Constraint set C_i for agent a_i and constraint set C_j for agent a_j.

```
1  d_i ← 0;
2  d_j ← 0;
3  while CLASSIFY-CONFLICT(MDD_i^{l_i^*+d_i+1}, MDD_j^{l_j^*+d_j+1}) ≠ NC do
4  |   d_i ← d_i + 1;
5  |   d_j ← d_j + 1;
6  end
7  while CLASSIFY-CONFLICT(MDD_i^{l_i^*+d_i+1}, MDD_j^{l_j^*+d_j}) ≠ NC do
8  |   d_i ← d_i + 1;
9  end
10 if CLASSIFY-CONFLICT(MDD_i^{l_i^⋆+d_i}, MDD_j^{l_j^⋆+d_j}) = PC then
11 |   return GENERATE-CONSTRAINTS-PC(MDD_i^{l_i^⋆+d_i}, MDD_j^{l_j^⋆+d_j});
12 else // CLASSIFY-CONFLICT returns AC
13 |   return GENERATE-CONSTRAINTS-AC(MDD_i^{l_i^⋆+d_i}, MDD_j^{l_j^⋆+d_j});
14 end
```

study how different ways of choosing the numbers of MDD levels affect the efficiency of CBS.

Example 6. *In any optimal solution of the corridor conflict example in Figure 1b, one of the two agents has a cost of 11. In Algorithm 5,* CLASSIFY-CONFLICT *returns* PC *until* $d_i = d_j = 4$*. Therefore, Algorithm 5 uses* MDD_1^{10} *and* MDD_2^{10} *to generate the constraint sets. After removing redundancies, the constraint set for agent* a_1 *is*

$$C_1 = \{\langle a_1, 5, B5\rangle, \langle a_1, 6, B4\rangle, \langle a_1, 6, B5\rangle, \langle a_1, 7, B3\rangle, \langle a_1, 7, B4\rangle\},$$

which prevents agent a_1 *from arriving at vertex* $C5$ *before timestep 11. Similarly, the constraint set for agent* a_2 *is*

$$C_2 = \{\langle a_2, 5, B1\rangle, \langle a_2, 6, B2\rangle, \langle a_2, 6, B1\rangle, \langle a_2, 7, B3\rangle, \langle a_2, 7, B2\rangle\},$$

which prevents agent a_2 *from arriving at vertex* $C1$ *before timestep 11.*

6 CBSH with Mutex Propagation

Based on our proposed mutex propagation techniques, we develop a new MAPF solver, called CBSH with Mutex Propagation (CBSH-MP), that extends CBSH-RM (Li et al. 2019c). CBSH-RM is equipped with heuristic guidance and rectangle reasoning that are retained in CBSH-MP. When resolving semi-cardinal and non-cardinal rectangle conflicts, CBSH-MP uses rectangle reasoning to generate barrier constraints. In addition, CBSH-MP uses Algorithm 2 to identify cardinal conflicts. When resolving cardinal conflicts, it uses Algorithm 5 to generate the constraint sets. CBSH-MP also caches the MDDs and the constraint sets for cardinal conflicts in two hash tables to reduce the runtime overhead of mutex propagation. From Section 5.1, CBSH-MP is complete and optimal.

7 Experimental Results

In this section, we report experimental results for CBSH-MP. We first compare CBSH-MP with CBSH-RM on the cardinal-conflict instances shown in Figure 1. Then, we compare CBSH-MP with CBSH-RM and CBSH (Felner et al. 2018) on various types of four-neighbor grid map instances from the MAPF benchmark set (Stern et al. 2019). These three MAPF solvers share the same codebase, except for conflict classification and constraint generation. We run all experiments on t2.large AWS EC2 instances with 8GB of memory.

7.1 Cardinal-Conflict Instances

We use the cardinal-conflict instances shown in Figure 1. Table 1 shows the number of CT node expansions of CBSH-MP and CBSH-RM. The Prefix "$>$" in an entry means that the MAPF solver does not solve the instance within five minutes, and the number after "$>$" is the number of CT node expansions when the runtime limit is reached. CBSH-MP solves all instances within one second.

CBSH-RM does not efficiently solve cardinal conflicts other than cardinal rectangle conflicts since it lacks rules to deal with such conflicts. However, it solves cardinal rectangle conflicts faster than CBSH-MP because rectangle reasoning has a smaller overhead than mutex propagation.

Table 1: Shows the number of CT node expansions on different cardinal-conflict instances. The "Size" and "Length" of rectangle and corridor conflicts are the size and length of the yellow areas in Figures 1a and 1b, respectively. The "Size" of goal vertex conflicts is the map size. The "Width" of switching agents conflicts is the map width.

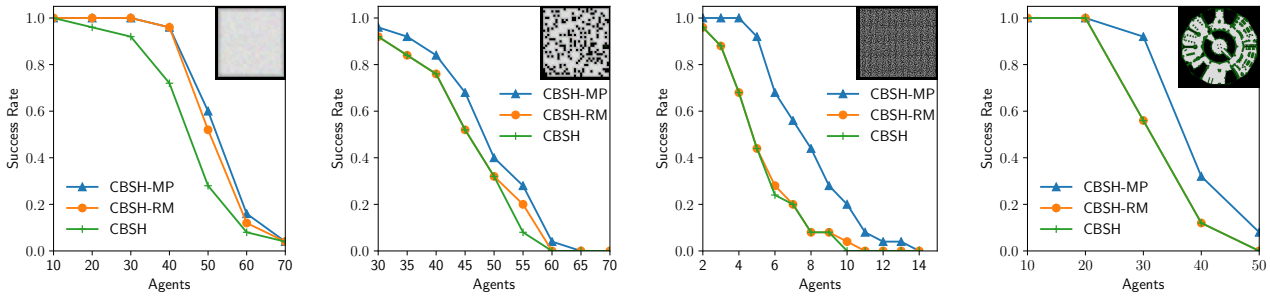
Cardinal Rectangle Conflict				
Size	5×5	6×6	7×7	8×8
CBSH-RM	1	1	1	1
CBSH-MP	1	1	1	1
Cardinal Corridor Conflict				
Length	12	14	16	18
CBSH-RM	$9,302$	$39,596$	$> 102,238$	$> 95,105$
CBSH-MP	1	1	1	1
Goal Vertex Conflict				
Size	6×6	6×6	7×7	8×8
CBSH-RM	$182,065$	$> 1,277,243$	$> 1,107,885$	$> 1,051,743$
CBSH-MP	1	1	1	1
Switching Agents Conflict				
Width	7	8	9	10
CBSH-RM	$> 685,234$	$> 695,391$	$> 690,585$	$> 690,394$
CBSH-MP	19	32	130	32

For rectangle conflicts, corridor conflicts and goal vertex conflicts, CBSH-MP expands only one CT node before finding an optimal solution. For switching agents conflicts, CBSH-MP expands only three CT nodes with cardinal conflicts at the bottom of the CT, and it expands only CT nodes with semi-cardinal and non-cardinal conflicts in the rest of the CT.

7.2 Benchmark Map Instances

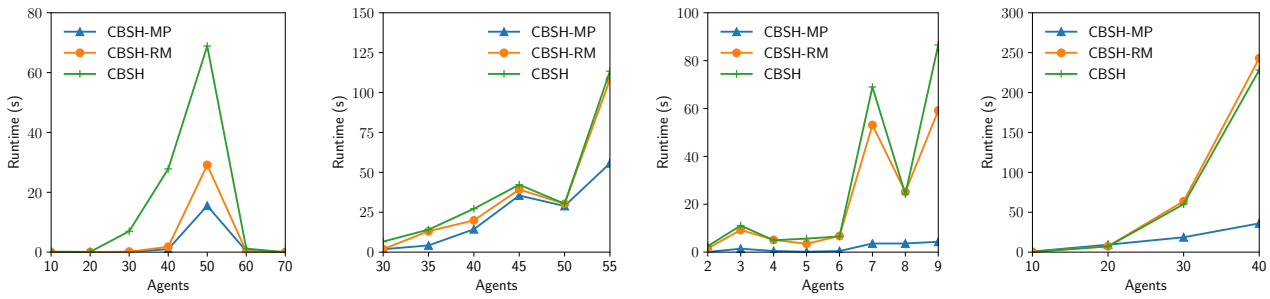
We use four benchmark maps from Stern et al. (2019): two small maps, which are the 16×16 empty map and the 32×32 map with 20% randomly blocked cells, and two large maps, which are the 194×194 game map lak303d and the 128×128 maze map with corridor width 1. We vary the number of agents and, for each number of agents, average over 25 "even scenarios" from the benchmark set.

Figure 3 shows the success rates of CBSH-MP, CBSH-RM and CBSH, which respectively specify how many instances are solved by them within the time limit of five minutes, and Figure 4 shows the runtime of each MAPF solver averaged over the instances solved by all MAPF solvers. In the 16×16 empty map, both CBSH-RM and CBSH-MP outperform CBSH since there are many rectangle conflicts in the problem instances. However, CBSH-MP only has a small advantage over CBSH-RM. In the other three maps with narrower environments, CBSH-MP outperforms CBSH-RM and CBSH in both runtime and success rate because there are more cardinal conflicts that CBSH-RM and CBSH do not have rules to deal with.



(a) empty-16-16 (b) random-32-32-20 (c) maze-128-128-1 (d) lak303d

Figure 3: Shows the success rates of various MAPF solvers on each map with different numbers of agents.



(a) empty-16-16 (b) random-32-32-20 (c) maze-128-128-1 (d) lak303d

Figure 4: Shows the runtimes of various MAPF solvers averaged over the instances solved by all of them on each map with different numbers of agents.

8 Related Work

Mutex propagation is related to merging MDDs in (Sharon et al. 2013) since merging MDDs can also be used to determine the existence of conflict-free paths. Different from mutex propagation, merging MDDs finds all reachable states instead of all mutually exclusive states. It is also applicable to more than two agents. However, it becomes very time-consuming as the number of agents increases.

Our work is also related to CBS with improved heuristics (CBSH2) (Li et al. 2019a) since CBSH2 also uses pairwise reasoning between agents to aggressively reason about the minimum increment of SoC for pairs of agents. In particular, the heuristic used by CBSH-MP is equivalent to the DG heuristic of CBSH2 because *a cardinal conflict* defined in this paper is synonymous with *a pair of dependent agents* defined in (Li et al. 2019a). In addition, the numbers of levels determined in Section 5.2 can also be used to improve the WDG heuristic of CBSH2, which we leave for future work.

Finally, our work is related to mutex reasoning built into a SAT-based MAPF solver (Surynek 2013). Different from CBSH-MP, that MAPF solver only marginally benefits from mutex reasoning.

9 Conclusions and Future Work

In this paper, we studied mutex propagation in the context of MAPF to efficiently reason about the interaction of two agents and infer applicable constraints from the resulting mutexes. We also proposed a novel algorithmic framework for automatically identifying cardinal conflicts and generating strong constraint sets for branching while preserving the optimality guarantee of CBS. Our experimental results report significant improvements over the state-of-the-art MAPF solvers with respect to runtime and success rate.

There are several interesting directions for future work, namely to (1) study how to use mutexes to resolve semi-cardinal conflicts and non-cardinal conflicts; (2) apply mutex propagation to variants of MAPF problems where conflicts are implicitly represented, such as the robust MAPF problem (Atzmon et al. 2018) and the MAPF problem with generalized conflicts (Hönig et al. 2018); and (3) study mutex propagation for incomplete Boolean models of MAPF where the formula is built lazily (Surynek 2019).

10 Acknowledgments

We thank Peter J. Stuckey, Daniel Harabor and Graeme Gange for helpful discussions and providing a first example of goal vertex conflicts. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189 and 1837779 as well as a gift from Amazon. The research at the Czech Technical University was supported by GAČR - the Czech Science Foundation under grant num-

ber 19-17966S. 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.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N. F. 2018. Robust multi-agent path finding. In *SoCS*, 2–9.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

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

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.

Hönig, W.; Preiss, J. A.; Kumar, T. K. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics* 34(4):856–869.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI*, 1194–1201.

Lam, E.; Bodic, P. L.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-cut-and-price for multi-agent pathfinding. In *IJCAI*, 1289–1296.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, 442–449.

Li, J.; Harabor, D.; Stuckey, P. J.; Felner, A.; Ma, H.; and Koenig, S. 2019b. Disjoint splitting for multi-agent path finding with conflict-based search. In *ICAPS*, 279–283.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019c. Symmetry-breaking constraints for grid-based multi-agent path finding. In *AAAI*, 6087–6095.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New techniques for pairwise symmetry breaking in multi-agent path finding. In *ICAPS*.

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

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

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

Nguyen, X., and Kambhampati, S. 2000. Extracting effective and admissible state space heuristics from the planning graph. In *AAAI*, 798–805.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *IJCAI*, 459–464.

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.

Stern, R.; Sturtevant, N. R.; Atzmon, D.; Walker, T.; Li, J.; Cohen, L.; Ma, H.; Kumar, T. K. S.; Felner, A.; and Koenig, S. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SoCS*, 151–158.

Surynek, P. 2013. Mutex reasoning in cooperative path finding modeled as propositional satisfiability. In *IROS*, 4326–4331.

Surynek, P. 2019. Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In *IJCAI*, 1177–1183.

Weld, D. S. 1999. Recent advances in AI planning. *AI Magazine* 20(2):93–123.

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

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 1443–1449.