Multi-Agent Path Finding for Large Agents *

Jiaoyang Li
CS Department
Univ. of Southern California
jiaoyanl@usc.edu

Pavel Surynek
Faculty of Information Technology
Czech Technical University
pavel.surynek@fit.cvut.cz

Ariel Felner
SISE Department
Ben-Gurion University
felner@bgu.ac.il

Hang Ma
T. K. Satish Kumar
Sven Koenig
CS Department
Univ. of Southern California

Abstract

Multi-Agent Path Finding (MAPF) has been widely studied in the AI community. For example, Conflict-Based Search (CBS) is a state-of-the-art MAPF algorithm based on a two-level tree-search. However, previous MAPF algorithms assume that an agent occupies only a single location at any given time, e.g., a single cell in a grid. This limits their applicability in many real-world domains that have geometric agents in lieu of point agents. Geometric agents are referred to as "large" agents because they can occupy multiple points at the same time. In this paper, we formalize and study LA-MAPF, i.e., MAPF for large agents. We first show how CBS can be adapted to solve LA-MAPF. We then present a generalized version of CBS, called Multi-Constraint CBS (MC-CBS), that adds multiple constraints (instead of one constraint) for an agent when it generates a high-level search node. We introduce three different approaches to choose such constraints as well as an approach to compute admissible heuristics for the high-level search. Experimental results show that all MC-CBS variants outperform CBS by up to three orders of magnitude in terms of runtime. The best variant also outperforms EPEA* (a state-of-the-art A*-based MAPF solver) in all cases and MDD-SAT (a state-of-the-art reduction-based MAPF solver) in some cases.

1 Introduction

In robotics and computer games, one has to find collision-free paths for multiple agents operating in a common environment. This has led to the study of Multi-Agent Path Finding (MAPF) in the AI community, where we are required to find a path for each agent from its given start vertex to its given goal vertex on a given graph such that no two agents collide with each other at any given time. MAPF can be applied to warehouse robots (Wurman, D'Andrea, and Mountz 2008), office robots (Veloso et al. 2015), aircraft-towing vehicles (Morris et al. 2016), video game characters (Ma et al. 2017) and many other domains.

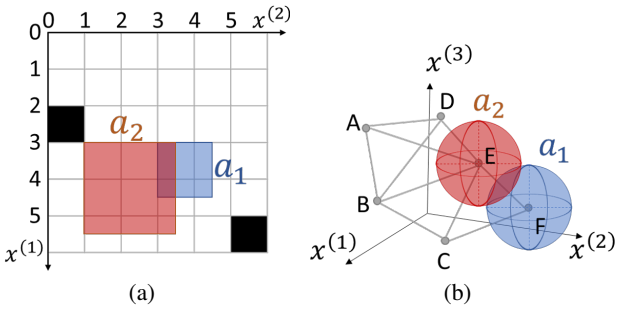
Numerous optimal MAPF algorithms have been developed in recent years, including reduction-based algorithms (Yu and LaValle 2013a; Erdem et al. 2013; Surynek et al. 2016), A*-based algorithms (Standley 2010; Wagner and Choset 2011; Goldenberg et al. 2014) and dedicated search-based algorithms (Sharon et al. 2013; 2015; Boyarski et al. 2015; Felner et al. 2018). See (Ma et al. 2016a; Felner et al. 2017) for complete surveys.

Although previous MAPF algorithms have found some real-world applications (Hönig et al. 2016), they are based on one simplistic assumption that limits their applicability. This assumption is to ignore the shape of agents and consider them as point agents, which occupy exactly one point at any time. In reality, agents are geometric in nature with definite shapes. Therefore, they typically occupy a set of points at any time. We refer to such agents as *large agents*. MAPF algorithms can be applied to large agents by lowering the resolution of discretization of the environment. However, this degrades their performance and thus their practical applicability.

To address these concerns, we formally study *Multi-Agent Path Finding for Large Agents* (LA-MAPF) which takes into consideration the shapes of agents. This consideration also addresses the case where agents have to maintain a safety distance from each other since a virtual boundary can now be defined for each agent. Intuitively, LA-MAPF is harder to solve than MAPF since it is a generalization of MAPF and agents are more likely to collide with each other.

The remainder of the paper is organized as follows. In Section 2, we formalize the LA-MAPF problem. In Section 3, we solve LA-MAPF by a straightforward adaption of a state-of-the-art search-based MAPF solver, Conflict-Based Search (CBS) (Sharon et al. 2015), and analyze its poor performance. In Section 4, we present a new algorithm, called Multi-Constraint CBS (MC-CBS), that improves CBS by adding multiple constraints in a high-level node expansion. We propose three different approaches to choose such constraints as well as a special constraint representation for regular shaped agents. In Section 5, we explain how to compute heuristics for the high-level search of MC-CBS. In Section 6, we compare the performances of adapted CBS, vari-

*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 was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2017692. 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.



(a) (b)

Figure 1: Shows two examples of two agents colliding with each other. (a) shows a 4-neighbor 2D grid with square-shaped agents, where lines represent edges, intersection points represent vertices, and black cells represent obstacles. (b) shows a 3D roadmap with sphere-shaped agents.

ants of MC-CBS, adapted EPEA* and adapted MDD-SAT.[1] EPEA* (Goldenberg et al. 2014) is a state-of-the-art A*-based MAPF solver, and MDD-SAT (Surynek et al. 2016) is a start-of-the-art reduction-based MAPF solver. All MC-CBS variants outperform CBS by up to three orders of magnitude in terms of runtime. The best variant also outperforms EPEA* in all cases and MDD-SAT in some cases.

2 Problem Definition and Related Concepts

We formalize LA-MAPF as follows: We are given an undirected graph $G = (V, E)$ embedded in a d-dimensional Euclidean space (usually $d = 2, 3$). Each vertex $v \in V$ is a point in the Euclidean space that is specified by its coordinates $(v^{(1)}, \ldots, v^{(d)})$. We are also given a set of k agents $\{a_1 \ldots a_k\}$ with unique start and goal vertices, and each agent has a fixed geometric shape around a reference point that cannot undergo transformations like rotations. We say that an agent is at a vertex v when its reference point is at vertex v, and we say that an agent traverses an edge (u, v) when its reference point moves along edge (u, v). If a_i is at vertex v, then the set of points occupied by a_i is denoted by $S_i(v)$. Figure 1(a) shows an example of two square-shaped agents on a 2D grid. Their reference points are the top-left corners. a_1 (blue) is at vertex $v = (3, 3)$ with an 1.5×1.5 square shape $S_1(v) = \{(x^{(1)}, x^{(2)}) \mid 0 \leq x^{(j)} - v^{(j)} \leq 1.5, j = 1, 2\}$, and a_2 (red) is at vertex $u = (3, 1)$ with a 2.5×2.5 square shape $S_2(u) = \{(x^{(1)}, x^{(2)}) \mid 0 \leq x^{(j)} - u^{(j)} \leq 2.5, j = 1, 2\}$. Figure 1(b) shows another example on a 3D roadmap where two sphere-shaped agents a_1 (blue) and a_2 (red) are at vertices F and E, respectively. Their reference points are the sphere centers.

On the same graph G, different agents have different traversable subgraphs because of their different shapes (e.g., in Figure 1(a), a_1 can be at $(0, 0)$ while a_2 cannot). This subgraph $G_i = (V_i, E_i)$ is called the configuration space of a_i. We assume that the start and goal vertices of each agent are in the same connected component of its configuration space. At each discrete timestep t, a_i can either *wait* at its current vertex $v \in V_i$, or *move* to an adjacent vertex u, where $(v, u) \in E_i$. Both wait and move actions have unit costs unless a_i terminally waits at its goal vertex.

In MAPF, a conflict between two agents is either a vertex conflict, where two agents are at 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 LA-MAPF, however, agents could collide when they are in close proximity with each other. Therefore, we generalize the definitions of conflicts. We define a *vertex conflict* as a five-element tuple $\langle a_i, a_j, u, v, t \rangle$, where the shapes of a_i and a_j overlap if a_i is at vertex u and a_j is at vertex v at the same timestep t. Similarly, we define an *edge conflict* as a seven-element tuple $\langle a_i, a_j, u_1, u_2, v_1, v_2, t \rangle$, where the shapes of a_i and a_j overlap if a_i moves from vertex u_1 to u_2 and a_j moves from vertex v_1 to v_2 at the same timestep t. We assume that there exists a conflict detection function that checks entire paths and returns all vertex conflicts and edge conflicts. Our task is to find a set of conflict-free paths that move all agents from their start vertices to their goal vertices with minimum sum of costs of all paths.

Previous research has also used generalized conflicts. Hönig et al. (2018) considered the shape of large agents like quadrotors and defined three new types of generalized conflicts: vertex-vertex conflicts, vertex-edge conflicts and edge-edge conflicts, where the first one is a special case of our vertex conflicts and the two other ones are special cases of our edge conflicts. They solved the problem by adapting ECBS (Barer et al. 2014), a suboptimal version of CBS, to their problem in a way similar to the adapted CBS that we introduce in Section 3.2. We show in Section 3.2 that this kind of direct adaptation is inefficient in many cases. Using a different perspective, Atzmon et al. (2018) generalized the definition of a conflict from a timestep to a time range in order to obtain robust plans.

3 Conflict-Based Search (CBS)

LA-MAPF is NP-hard, since it generalizes MAPF, which is NP-hard (Yu and LaValle 2013b; Ma et al. 2016b). In this section, we review Conflict-Based Search for MAPF and introduce its direct adaption to LA-MAPF.

3.1 CBS for MAPF

Conflict-Based Search (CBS) (Sharon et al. 2015) is a two-level search-based algorithm that is complete and optimal for MAPF. Its high level performs a best-first search on a binary *constraint tree* (CT). Each CT node contains a set of constraints, where a constraint is either a *vertex constraint* $\langle a_i, u, t \rangle$ that prohibits agent a_i from being at vertex u at timestep t or an *edge constraint* $\langle a_i, u_1, u_2, t \rangle$ that prohibits a_i from moving from u_1 to u_2 at timestep t, and a set of paths for all agents that satisfy all constraints. The *cost* of the CT node is the sum of costs of all paths. The CT root node contains an empty set of constraints and a set of individual shortest paths.

When the high level expands a CT node N, it first runs a conflict detection function to find conflicts among all of its

[1] We refer to the adapted versions of CBS, EPEA* and MDD-SAT also as CBS, EPEA* and MDD-SAT, respectively.

paths. If they are conflict-free, then N is a goal node and CBS returns its paths. Otherwise, the high level arbitrarily chooses and resolves a vertex conflict $\langle a_i, a_j, u, v, t\rangle$ ($u = v$ in MAPF) or an edge conflict $\langle a_i, a_j, u_1, u_2, v_1, v_2, t\rangle$ ($u_1 = v_2$ and $u_2 = v_1$ in MAPF) by *splitting* N into two child CT nodes, N_1 and N_2, that inherit all constraints and paths from N. The high level also adds a new vertex constraint $\langle a_i, u, t\rangle$ or edge constraint $\langle a_i, u_1, u_2, t\rangle$ to N_1 and a new vertex constraint $\langle a_j, v, t\rangle$ or edge constraint $\langle a_j, v_1, v_2, t\rangle$ to N_2 and then runs a low-level search for both child nodes to find shortest paths for all agents that satisfy the new set of constraints in each of them. Since each child node has one additional constraint imposed on only one agent, the low-level search needs to re-plan the path for only that one agent.

Improved CBS (ICBS) (Boyarski et al. 2015) improves CBS by classifying conflicts and choosing to resolve cardinal conflicts first. A conflict is *cardinal* if, when CBS uses this conflict to split a CT node, the cost of each one of the two resulting child nodes is larger than the cost of the CT node. CBSH (Felner et al. 2018) improves it further by aggregating cardinal conflicts and computing admissible heuristics to guide the high-level search.

3.2 CBS for LA-MAPF

Similar to techniques used in (Hönig et al. 2018), CBS can be directly adapted to LA-MAPF by considering generalized conflicts and changing the conflict detection function to the one discussed in Section 2. However, the resulting version of CBS is very inefficient for large agents. For example, in Figure 1(a), a_1 tries to move from $(3, 0)$ to $(3, 4)$, and a_2 tries to move from $(0, 1)$ to $(3, 1)$. In the CT root node, both agents follow their individual shortest paths and have vertex conflicts at timesteps 1, 2 and 3. The drawing is a snapshot that shows their vertex conflict at timestep 3. If CBS chooses to resolve this vertex conflict, then the right child node has a new constraint $\langle a_2, (3, 1), 3\rangle$. a_2 is forced to wait for one timestep and thus stays at vertex $(2, 1)$ at timestep 3. However, a_2 then still conflicts with a_1 at timestep 3 in this child node. The conflict between a_1 and a_2 at timestep 3 is therefore not resolved by this split. Although it will be resolved eventually, many intermediate CT nodes are generated.

4 Multi-Constraint CBS (MC-CBS)

The above observations motivate the idea of adding multiple constraints in a single CT node expansion. We present a new algorithm, MC-CBS, which adds multiple constraints for the same timestep to child nodes in order to resolve multiple related conflicts in a single CT node expansion, which resembles lookahead reasoning at the high level and can result in smaller CTs, thus making the search more efficient.

We say that two vertex or edge constraints for a_i and a_j respectively are *mutually disjunctive* iff any pair of conflict-free paths for a_i and a_j satisfies at least one of the two constraints, i.e., there do not exist two conflict-free paths such that both constraints are violated. In particular, the constraints that CBS adds to two child nodes are always mutually disjunctive. We say that two sets of constraints are *mutually disjunctive* iff each constraint in one set is mutually disjunctive with each constraint in the other set. Intuitively, working with mutually disjunctive constraint sets gives us the power of constraint propagation.

When MC-CBS resolves a conflict $\langle a_i, a_j, u, v, t\rangle$ (or $\langle a_i, a_j, u_1, u_2, v_1, v_2, t\rangle$) in a CT node N, it generates two child nodes with N's current constraint set and additional constraint sets, C_1 and C_2, respectively:

1. C_1 and C_2 include the *core constraints* that CBS uses to resolve the conflict, i.e., $\langle a_i, u, t\rangle \in C_1$ and $\langle a_j, v, t\rangle \in C_2$ (or $\langle a_i, u_1, u_2, t\rangle \in C_1$ and $\langle a_j, v_1, v_2, t\rangle \in C_2$).
2. C_1 and C_2 are enhanced with other constraints that ensure that C_1 and C_2 remain mutually disjunctive.

Different strategies for choosing C_1 and C_2 are discussed in Sections 4.1 and 4.2. Here, we show that MC-CBS is complete and optimal.

Lemma 1 (Optimality)**.** *If MC-CBS terminates, it returns a set of conflict-free paths with the minimum sum of costs.*

Proof. The proof has two parts. We first show that, for a given CT node N with constraint set C, any set of conflict-free paths that satisfy C also satisfy at least one of the constraint sets $C \bigcup C_1$ and $C \bigcup C_2$. This is true because, otherwise, there would exist two conflict-free paths such that both of them are consistent with C but one path violates a constraint $c_1 \in C_1$ and the other one violates a constraint $c_2 \in C_2$. Then, c_1 and c_2 are not mutually disjunctive, contradicting the assumption. Therefore, the expansion of a CT node does not lose any set of conflict-free paths.

The second part is the same as the proof of optimality for CBS (Sharon et al. 2015). The cost of a CT node is a lower bound on the sum of costs of conflict-free paths that satisfy its constraint set. The high-level search chooses to expand a CT node with minimum cost and the expansion does not lose any conflict-free paths. So the cost of the chosen CT node is always a lower bound on the sum of costs of all conflict-free paths. Therefore, the set of paths of the first CT node chosen for expansion whose paths are conflict-free is indeed a set of conflict-free paths with the minimum sum of costs. □

Lemma 2 (Completeness)**.** *MC-CBS terminates if there exists a set of conflict-free paths.*

Proof. The costs of CT nodes are non-decreasing in expansion order because MC-CBS runs a best-first search and the costs of child nodes are at least as large as the costs of their parents. In addition, there are a finite number of CT nodes for a given cost c, because there are only a finite number of possible conflicts within c timesteps and, once a conflict is resolved at CT node N, it does not reappear in the subtree of N. Therefore, a set of conflict-free paths must be found after expanding a finite number of CT nodes. □

There are numerous ways to generalize the core constraints to two mutually disjunctive constraint sets for MC-CBS. For example, in Figure 1(a), the two sets first include the core constraints, i.e., $C_1 = \{\langle a_1, (3, 3), 3\rangle\}$ and $C_2 = \{\langle a_2, (3, 1), 3\rangle\}$. We can add constraints for other vertices to C_1 and C_2, such as $\langle a_2, (2, 1), 3\rangle$ to C_2 (because a_1 being at $(3, 3)$ conflicts with a_2 being at $(2, 1)$). We can also

add constraints for other timesteps, such as $\langle a_2, (3,2), 4\rangle$ to C_2 (because, no matter where a_1 moves from $(3,3)$ at timestep 3, it conflicts with a_2 being at $(3,2)$ at timestep 4).

In this paper, we only add constraints for the same timestep to keep MC-CBS simple. In general, it is hard to determine whether two constraints for different timesteps are mutually disjunctive. Moreover, we can often find equivalent (e.g., block the same set of paths) or better (e.g., block a larger set of paths) constraint sets to add for the same timestep. In Figure 1(a), the pair of constraint sets $\{\langle a_1, (3,3), 3\rangle\}$ and $\{\langle a_2, (3,1), 3\rangle, \langle a_2, (3,2), 4\rangle\}$ is no better than the pair of constraint sets $\{\langle a_1, (3,3), 3\rangle\}$ and $\{\langle a_2, (x,y), 3\rangle \mid 2 \leq x \leq 4, 1 \leq y \leq 3\}$ because a_2 can only be at vertex $(x,y), 2 \leq x \leq 4, 1 \leq y \leq 3$, at timestep 3 to reach vertex $(3,2)$ at timestep 4.

In Sections 4.1 and 4.2, we introduce three approaches for choosing constraint sets for MC-CBS. In Section 4.3, we present a new representation of constraint sets to speed up MC-CBS for regular shaped agents. We only discuss the constraint sets for vertex conflicts since the constraint sets for edge conflicts can be derived analogously.

4.1 Asymmetric and Symmetric Approaches

Consider the resolution of a vertex conflict $\langle a_i, a_j, u, v, t\rangle$. Two mutually disjunctive constraint sets can be built asymmetrically or symmetrically.[2]

1. The *asymmetric approach* (ASYM) adds a constraint set of size one $\{\langle a_i, u, t\rangle\}$ to the left child node that prohibits agent a_i from being at its current vertex v at timestep t and a large constraint set $\{\langle a_j, v', t\rangle \mid \langle a_i, a_j, u, v', t\rangle$ is a vertex conflict$\}$ to the right child node that prohibits agent a_j from being at any vertex at timestep t where it could collide with a_i.
2. The *symmetric approach* (SYM) chooses a point p in the Euclidean space that is inside the overlap area $S_i(u) \cap S_j(v)$, and then adds one constraint set to each child node. The constraint set blocks all vertices that the agent could be at while including p in its shape, i.e., $C_1 = \{\langle a_i, v', t\rangle \mid p \in S_i(v'), v' \in V\}$ and $C_2 = \{\langle a_j, v', t\rangle \mid p \in S_j(v'), v' \in V\}$. Unlike ASYM, that adds a constraint set of size one to one child node and a large constraint set to the other child node, SYM adds constraint sets of similar cardinalities to both child nodes.

For example, in Figure 1(a), ASYM adds $\{\langle a_1, (3,3), 3\rangle\}$ to one child node and $\{\langle a_2, (x,y), 3\rangle \mid 1 \leq x, y \leq 4\}$ to the other child node. But SYM adds $\{\langle a_1, (x,y), 3\rangle \mid 2 \leq x, y \leq 3\}$ to one child node and $\{\langle a_2, (x,y), 3\rangle \mid 1 \leq x, y \leq 3\}$ to the other child node when the chosen point p is $(3.25, 3.25)$. Thus, ASYM adds 17 constraints in total while SYM adds only 13 constraints in total. However, we empirically show in Section 6.1 that SYM usually outperforms ASYM, perhaps because SYM usually has more pairs of mutually disjunctive constraints, e.g., 36 pairs in this example. Since each such pair may block some conflicting paths, SYM usually results in smaller search spaces.

Another important observation is that not all constraints generated by SYM or ASYM are relevant. Some constraints are irrelevant because they block vertices that are not reachable by the agents at the given timestep, such as $\langle a_2, (4,1), 3\rangle$ given that the start vertex of a_i is $(0,1)$, and some constraints are less relevant because they block the vertices that can be reached only when paths are very long, such as $\langle a_2, (1,4), 3\rangle$ given that the start vertex and the goal vertex of a_2 are $(0,1)$ and $(3,1)$, respectively. Therefore, we present a more intelligent approach that aims to add more relevant constraints in the next section.

4.2 Maximizing the Costs of Child Nodes

When splitting a CT node, large costs of both child nodes mean more progress for the high-level search. Thus, the costs of the child nodes can be used to measure the relevance of a constraint set. ICBS (Boyarski et al. 2015) utilizes this insight by choosing cardinal conflicts whenever available, where both child nodes have higher costs than their parent node. Here, we try to find a pair of constraint sets where the child nodes have the highest possible costs.

Calculating the weights of constraint sets. A Multi-Valued Decision Diagram (MDD) (Sharon et al. 2013) MDD_i^μ is a directed acyclic graph that consists of all paths of cost μ for agent a_i from its start vertex to its goal vertex. All nodes at depth t of MDD_i^μ correspond to all vertices where a_i can be at timestep t en route such a path. Therefore, if a constraint set for timestep t blocks all vertices of MDD_i^μ at depth t, the cost of the agent's new path is at least $\mu + 1$ after adding it. Conversely, if a constraint set for timestep t does not block all vertices of MDD_i^μ at depth t, the cost of the agent's new path is at most μ after adding it.

ICBS uses MDD_i^μ to predict whether the cost of a_i's path is μ or higher after adding a constraint. Here, we generalize this idea and use $MDD_i^{\mu+d}$ to predict whether the cost of a_i's path is $\mu, \ldots, \mu + d$ or at least $\mu + d + 1$ after adding a constraint set. $d \in \mathbb{N}$ is called the *lookahead depth*, and the predicted cost increment in $\{0, 1, \ldots, d+1\}$ is called the *weight* of the constraint set.

Given a CT node N and an agent a_i whose path is of cost μ, we build $MDD_i^{\mu+d}$. During this construction, we also assign a weight w to each MDD node (v,t) such that $\mu + w$ equals the cost of a_i's shortest path moving from its start vertex to its goal vertex via vertex v at timestep t, i.e., $(v,t) \in MDD_i^{\mu+w}$ and $(v,t) \notin MDD_i^{\mu+w-1}$. The weight of a constraint set C' for a_i at timestep t is calculated as follows. Consider all MDD nodes at depth t: (1) If C' does not block all MDD nodes with weight 0, then the cost of a_i's path after adding C' does not change, and thus the weight of C' is 0. (2) Otherwise, if C' blocks all MDD nodes with weights no more than w ($0 \leq w < d$) but does not block all MDD nodes with weight $w+1$, then the cost of a_i's path after adding C' increases by exactly $w+1$, and thus the weight of C' is $w+1$. (3) Otherwise, C' blocks all MDD nodes, the cost of a_i's path after adding C' increases by at least $d+1$, and thus the weight of C' is $d+1$.

[2] Atzmon et al. (2018) introduced similar asymmetric and symmetric approaches to add multiple constraints, although they studied a different problem and added constraints for the same vertex at different timesteps.

Algorithm 1: A template for MaxWeight-d.

Input: Vertex conflict $\langle a_i, a_j, u, v, t\rangle$, and depth d.
Output: Constraint sets, C_1 and C_2, and weights, w_1 and w_2.

1 $C_1 \leftarrow \{\langle a_i, u, t\rangle\}$;
2 $C_2 \leftarrow \{\langle a_j, v', t\rangle \mid \langle a_i, a_j, u, v', t\rangle$ is a vertex conflict$\}$;
3 $w_1 \leftarrow$ getWeight(C_1); $w_2 \leftarrow$ getWeight(C_2);
4 **for** $w_1' = w_1 + 1 : d + 1$ **do**
5 $\quad C_1' \leftarrow \{\langle a_i, u', t\rangle \mid (u', t) \in MDD_i^{\mu+d}$ and its weight $< w_1'\}$// *μ is the cost of a_i's shortest path.*
6 $\quad C_2' \leftarrow \{\langle a_j, v', t\rangle \mid \langle a_j, v', t\rangle$ is mutually disjunctive with every constraint in $C_1'\}$;
7 $\quad$ **if** $\langle a_j, v, t\rangle \notin C_2'$ **then**
8 $\quad\quad$ break;
9 $\quad w_2' \leftarrow$ getWeight(C_2');
10 $\quad$ **if** $\{w_1', w_2'\}$ *is better than* $\{w_1, w_2\}$ **then**
11 $\quad\quad C_1, C_2, w_1, w_2 \leftarrow C_1', C_2', w_1', w_2'$;
12 return C_1, C_2, w_1 and w_2;

Maximizing the weights of constraint sets. We can now design a new approach for choosing constraint sets for MC-CBS based on maximizing their weights. We refer to this approach as *MaxWeight-d* (MAX-d or MAX for short). It builds MDDs for both agents with lookahead depth d and chooses the *best* pair of constraint sets, i.e, the pair that maximizes the smaller weight of the two. It breaks ties by preferring the pair with the maximum sum of weights.

Algorithm 1 shows the pseudo-code for MAX. It enumerates all possible weights of C_1 and, for each possible weight, computes the maximum weight of C_2 such that C_2 includes the core constraint and is mutually disjunctive with C_1. Initially, it regards the pair of constraint sets generated by ASYM as the best pair seen so far (Lines 1-3). It then iteratively increases the weight w_1' of the first constraint set (Line 4). In each iteration, it first chooses C_1' to be of minimum size for the given weight w_1' (Line 5), i.e., C_1' only blocks those MDD nodes whose weights are smaller than w_1'. It then builds C_2' of maximum size to be mutually disjunctive with C_1' (Line 6). If the core constraint $\langle a_j, v, t\rangle \notin C_2'$, indicating that C_1' cannot have a weight equal to or larger than w_1', it just returns the best constraint sets and weights found so far (Lines 7-8). Otherwise, it computes the weight of C_2' (Line 9) and updates the best constraint sets and weights if necessary (Lines 10-11).

Algorithm 1 is shown only for vertex conflicts for ease of explanation. When expanding a CT node in MC-CBS, we run Algorithm 1 for all vertex conflicts and edge conflicts and choose to resolve the conflict with the best constraint sets. Particularly, the smaller weight of two constraint sets for cardinal conflicts is always at least one, and the smaller weight of two constraint sets for other conflicts is always zero. Therefore, cardinal conflicts still have higher priority than other conflicts, which is consistent with the conflicts prioritization of ICBS.

4.3 Constraints for Regular Shaped Agents

Agents often have regular shapes, such as rectangles and circles in a 2D space or cuboids and spheres in a 3D space. In such cases, two mutually disjunctive constraint sets have geometric representations that can be leveraged for computational benefits.

For example, when all agents are rectangular, an agent a_i of size $\vec{s}_i = (s_i^{(1)}, s_i^{(2)})$ whose reference point is the point with minimum coordinates has $S_i(v) = \{\vec{p} \mid \vec{v} \leq \vec{p} \leq \vec{v} + \vec{s}_i\}$. Here, $\vec{v}$ is the vector representation of vertex v and the vector inequalities indicate that the inequalities hold for both dimensions. The tuple $\langle a_i, a_j, u, v, t\rangle$ is a vertex conflict iff $S_i(u) \cap S_j(v) \neq \emptyset$ or, equivalently,

$$-\vec{s}_j \leq \vec{v} - \vec{u} \leq \vec{s}_i. \tag{1}$$

A *rectangle constraint* of the form $\langle a_i, \vec{u}_{min}, \vec{u}_{max}, t\rangle$ can be used to represent a constraint set that prohibits agent a_i from being in the rectangular area $\{v \mid \vec{u}_{min} \leq \vec{v} \leq \vec{u}_{max}\}$ at timestep t. We say that two rectangle constraints $\langle a_i, \vec{u}_{min}, \vec{u}_{max}, t\rangle$ and $\langle a_j, \vec{v}_{min}, \vec{v}_{max}, t\rangle$ are *mutually disjunctive* iff their corresponding constraint sets are mutually disjunctive or, equivalently, $-\vec{s}_j \leq \vec{v}_{min} - \vec{u}_{max}$ and $\vec{v}_{max} - \vec{u}_{min} \leq \vec{s}_i$.

We can replace the constraint sets C_1 and C_2 for MC-CBS by two rectangle constraints. Instead of checking for each pair of constraints in C_1 and C_2 whether they are mutually disjunctive we need to only check two diagonal vertices, and all approaches in Sections 4.1 and 4.2 can thus be simplified.

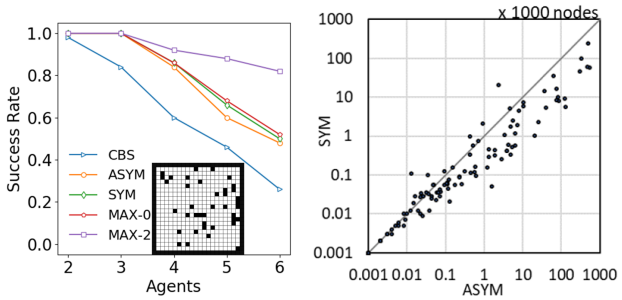
For some other regular shaped agents, such as circle, cuboid or sphere shaped agents, we can use similar representations and thus simplify MC-CBS as well.

5 Adding Heuristics to MC-CBS

Felner et al. (2018) improve CBS by adding heuristics to the high-level search. A CT node's g-value is its cost, and its admissible heuristic is the cost of the minimum vertex cover of an unweighted conflict graph. The conflict graph has vertices representing agents and edges representing cardinal conflicts between agents. Here, we generalize this model to a weighted conflict graph $G_{CF} = (V_{CF}, E_{CF})$, where each vertex $v_i \in V_{CF}$ represents an agent a_i, each edge $(v_i, v_j) \in E_{CF}$ represents cardinal conflicts between agents a_i and a_j, and each pair of weights w_{ij} and w_{ji} for edge (v_i, v_j) represents that either a_i has to increase its cost by at least w_{ij} or a_j has to increase its cost by at least w_{ji} in order to resolve this conflict. In MC-CBS, we use the weights of C_1 and C_2 for w_{ij} and w_{ji}, respectively. Then, the optimal solution of the following Integer Linear Program (ILP) is an admissible heuristic:

$$\begin{aligned} minimize \quad & \sum_{i=1}^{k} c_i & & (2)\\ s.t. \quad & c_i \geq w_{ij} x_{ij}, & 1 \leq i, j \leq k, i \neq j \\ & x_{ij} + x_{ji} \geq 1, & 1 \leq i < j \leq k \\ & x_{ij} \in \{0, 1\}, & 1 \leq i, j \leq k, i \neq j, \end{aligned}$$

where c_i is a variable that represents the minimum cost that a_i's path has to increase, and x_{ij} is a Boolean variable that represents whether the conflict between a_i and a_j is resolved by increasing the cost of a_i's path. The minimum vertex



(a) Success rate. (b) Expanded CT nodes.

Figure 2: Experimental results for CBS and variants of MC-CBS on a 20×20 grid with 10% blocked cells.

cover problem in (Felner et al. 2018) is a special case where all $w_{ij} = 1$. Similar conflict graphs and ILP models are used in the context of cost-optimal planning (Pommerening et al. 2015).

6 Experimental Results

We implemented and experimented with CBS, all MC-CBS variants (i.e., ASYM, SYM and MAX), EPEA* and MDD-SAT on grids with square agents. We also tested CBS and some MC-CBS variants on a 3D roadmap with ellipsoid agents. All CBS-based algorithms used the conflict prioritization of ICBS. All algorithms were written in C++ and ran on a 2.80 GHz Intel Core i7-7700 laptop with 8 GB RAM and a runtime limit of 5 minutes. We tried both brute force search and an off-the-shelf ILP solver to solve the ILP in Equation (2) for calculating heuristics. Brute force search ran faster on small graphs while the ILP solver ran faster on large graphs. In both cases, the overhead of computing the heuristics was very small since the conflict graphs were sparse. Therefore, we simply use brute force search in our experiments.

6.1 Square Agents on a 2D Grid

We first compare the performances of CBS and all MC-CBS variants on a 4-neighbor 20×20 2D grid with 10% random blocked cells. Each agent is a 2.5×2.5 square whose reference point is its top-left corner. All algorithms use Equation (1) to detect conflicts. All MC-CBS variants use the rectangle constraints discussed in Section 4.3. For MAX, we tested lookahead depths d from 0 to 4. We used 50 instances with randomly generated start vertices and goal vertices for each number of agents.

Overall performances. Figure 2(a) shows the *success rate*, i.e., number of instances solved within 5 minutes, of each algorithm. The success rates of MAX with $d = 1, 2, 3, 4$ are quite close to each other, so we only plotted MAX-2. Overall, CBS performs the worst since it repeatedly resolves related conflicts between the same agents. SYM, ASYM and MAX-0 perform better since they resolve a set of related conflicts in a single expansion. MAX-2 performs the best since it looks ahead several steps and takes into consideration the costs of child nodes. Table 1 shows the runtime and number of expanded CT nodes of CBS, SYM and MAX-2 on instances solved by SYM. SYM outperforms CBS by up to three orders of magnitude on both metrics, indicating that the overhead per expanded CT node is negligible for SYM. MAX-2 outperforms SYM by factors of up to 60 and 20 with respect to the number of expanded CT nodes and the runtime, respectively, indicating that it has more overhead per expanded CT node but is still beneficial overall.

Table 1: Average runtime and number of expanded CT nodes for CBS, SYM and MAX-2 on instances solved by SYM. Cutoff time of 5 minutes is included in the average for unsolved instances. k is the number of agents.

		Runtime (ms)			Expanded CT nodes		
k	Ins.	CBS	SYM	MAX-2	CBS	SYM	MAX-2
2	50	8,907	7	**2**	25,924	20	**3**
3	50	52,876	903	**24**	218,136	2,214	**36**
4	43	>98,056	22,057	**2,100**	>376,921	43,904	**2,958**
5	33	>138,875	72,703	**3,056**	>320,928	125,593	**2,683**
6	25	>199,285	116,613	**6,373**	>802,317	244,991	**8,324**

Table 2: Average runtime (ms) of MAX with different lookahead depths d. Cutoff time of 5 minutes is included in the average for unsolved instances. k is the number of agents.

k	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$
2	6.6	1.8	1.9	**1.7**	2.1
3	551.1	24.0	**23.6**	31.6	49.3
4	43,747.1	31,612.9	29,544.7	**27,391.2**	28,539.8
5	104,597.3	56,336.4	51,225.3	**49,273.0**	55,912.9
6	152,360.1	70,675.5	**61,283.5**	66,329.4	68,116.7

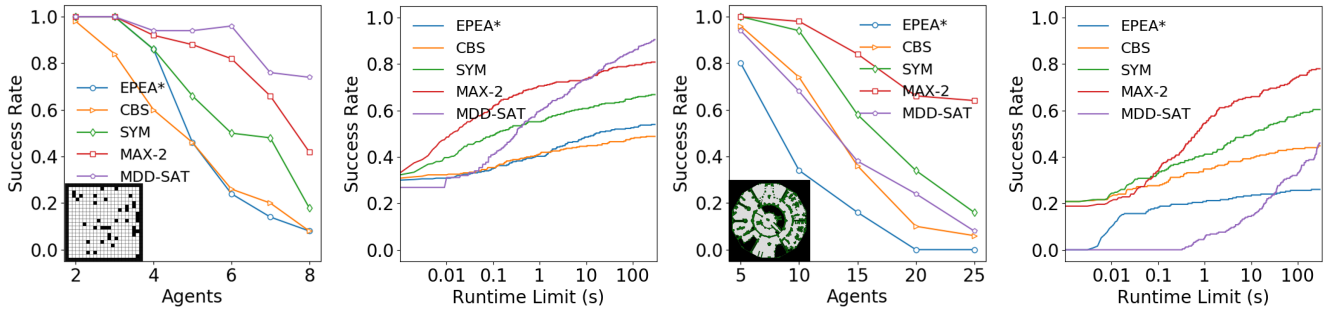
Comparing ASYM and SYM. Figure 2(b) shows the number of expanded CT nodes for ASYM and SYM on instances solved by both algorithms. With a few exceptions, SYM outperforms ASYM not only on the success rate but also on the number of expanded CT nodes (and thus the runtime).

Comparing MAX with different lookahead depths. Table 2 compares the runtime of MAX with d ranging from 0 to 4. We observe that MAX with $d = 0$ (no lookahead) is significantly slower than the others. As d increases, the runtime first decreases and then increases because a larger lookahead depth has more pruning power but also incurs more overhead. $d = 2$ or 3 works best in our experiments.

MAX with heuristics. Adding heuristics to MAX decreases both the number of expanded CT nodes and the runtime, although it does not improve its success rate. For instance, adding heuristics to MAX-4 decreases the number of expanded CT nodes by 30.1% and the runtime by 21.5% on 8-agent instances. Similar observations have been made in (Felner et al. 2018) on sparse grids, but in the next subsection we discuss cases where adding heuristics provides a much larger speedup.

6.2 Ellipsoid Agents on a 3D Roadmap

We also tested our algorithms on the "Swap50" instance from (Hönig et al. 2018), that is used to plan paths for



(a) Success rate on small map (b) Runtime on small map (c) Success rate on large map (d) Runtime on large map

Figure 3: Experimental results on 2D grids.

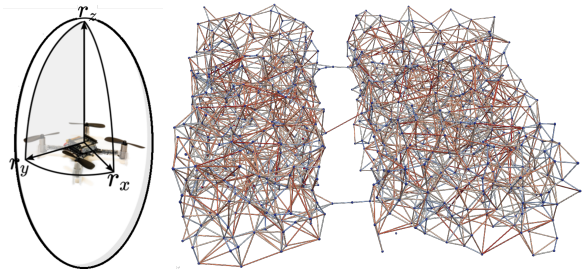


Figure 4: Shape of agents (left), reproduced from (Hönig et al. 2018), and the structure of the roadmap (right).

Table 3: The runtime and number of expanded CT nodes on "Swap50" with a 3D roadmap. k is the number of agents. MAX+h is MAX-1 with heuristics.

	Runtime (ms)				Expanded CT nodes			
k	CBS	ASYM	MAX-1	MAX+h	CBS	ASYM	MAX-1	MAX+h
5	38	61	23	**13**	18	26	9	**7**
6	124	122	126	**40**	28	25	20	**11**
7	1,175	549	735	**290**	188	93	100	**52**
8	253,550	15,829	10,570	**5,221**	30,411	2,408	1,289	**747**

quadrotor swarms in a dense 3D space. The environment is a 7.5 m $\times$ 6.5 m $\times$ 2.5 m space with ellipsoid agents. The ellipsoids model quadrotors and their downwash effects. The ellipsoids have radii (0.15 m, 0.15 m, 0.3 m) with central reference points, as in Figure 4(left). Agents are required to fly through windows in a wall in opposite directions. The roadmap, generated by the SPARS algorithms (Dobson and Bekris 2014), has 869 vertices and 3,371 edges, as in Figure 4(right). In a preprocessing step, we identify all possible vertex and edge conflicts using the swept collision model (Hönig et al. 2018). On average, there exist 1.75 different vertex conflicts per vertex, and 24.08 different edge conflicts per edge. The original instance has 50 agents and was solved by suboptimal solvers in (Hönig et al. 2018). We randomly chose a small subset of agents and solved the resulting instance with our optimal algorithms CBS, ASYM, MAX-1, and MAX-1 with heuristics.

Table 3 shows the runtime and number of expanded CT nodes. ASYM, MAX-1 and MAX-1 with heuristics decrease the runtime of CBS by factors of up to 16, 24 and 49, respectively. Unlike the results in Section 6.1, adding heuristics here significantly reduces the number of expanded CT nodes and runtime because agents traverse narrow corridors in opposite directions, which results in more cardinal conflicts.

6.3 Comparing with Other Solvers on 2D Grids

We now compare CBS and two MC-CBS variants, namely SYM and MAX-2, with an A*-based solver, EPEA*, and a reduction-based solver, MDD-SAT, on the same instances as in Section 6.1. We also test them on larger instances with square agents of different sizes randomly chosen from $\{2.5, 3.5, 4.5\}$. We used a large 4-neighbor 194$\times$194 2D grid with 51.3% blocked cells, namely the benchmark game map lak503d from (Sturtevant 2012). We used 50 instances with randomly generated start vertices and goal vertices for each number of agents.

EPEA*. A*-based MAPF solvers conduct search in the joint multi-agent state space. EPEA* (Goldenberg et al. 2014) is a lazy version which delays the generation of nodes whose costs are larger than the costs of their parents and thus avoid generating unnecessary nodes. It can be adapted to LA-MAPF by using Equation (1) to identify conflict-free actions as legal operators. This adaptation is promising since the branching factor could be smaller than the one for MAPF instances. CBS-based solvers, on the other hand, have more conflicts to resolve for LA-MAPF than for MAPF.

MDD-SAT. MDD-SAT (Surynek et al. 2016) is a SAT-based MAPF solver. It relies on the construction of a propositional formula that is satisfiable iff the given MAPF instance has a set of conflict-free paths of a certain cost. It introduces a propositional variable $\mathcal{X}_v^t(a_i)$ for each agent a_i and node (v, t) in a_i's MDD that is 'true' iff agent a_i is at vertex v at timestep t. MDD-SAT can be modified for LA-MAPF by introducing additional variables $\mathcal{Y}_u^t(a_i)$ and constraints involving them to reflect the shapes of agents. Here, $\mathcal{Y}_u^t(a_i)$ is 'true' iff there exists vertex v such that $\mathcal{X}_v^t(a_i)$ is 'true' and vertex $u \in S_i(v)$. In other words, implications $\mathcal{X}_v^t(a_i) \rightarrow \mathcal{Y}_u^t(a_i)$ added for each vertex $u \in S_i(v)$ ensure that the entire shape of the agent moves in tandem with the reference point. In addition to MDD-SAT's constraints that

encode movement rules for reference points, "at-most-one" constraints $\sum_{i=1}^{k} \mathcal{Y}_v^t(a_i) \leq 1$ are used to disallow collisions between agents.

Results. Figures 3(a) and 3(b) present the success rates and runtimes on the small map. MAD-SAT dominates all other algorithms in terms of success rates within 5 minutes. When we vary the runtime limit, MAX-2 has the highest success rate when the runtime limit is less than 10 s (where instances are relatively easy), while MDD-SAT has the highest success rate when the runtime limit is more than 10 s (where instances are more difficult). Figures 3(c) and 3(d) present the success rates and runtimes on the large game map. MAX-2 significantly outperforms all other algorithms in all cases except when the runtime limit is less than 0.1 s.

Although it is hard to predict the performances of the algorithms in all domains, we can give some guidance based on these observations and analysis. MDD-SAT is strong for difficult problems in small domains. MAX is strong for easy problems or in large domains, despite the fact that its lookahead depth needs to be set appropriately.

7 Conclusions and Future Work

In this paper, we generalized MAPF to a practically viable version, called LA-MAPF, that takes into consideration the shapes of agents. We presented MC-CBS, a new algorithm that improves CBS by adding multiple constraints during the expansion of a CT node. Unlike CBS, the MC-CBS allows us to resolve multiple related conflicts in one shot, while also generalizing the use of the minimum vertex cover for heuristic guidance of the high-level search. We proposed three approaches for choosing the constraints as well as an approach for computing the heuristics. Empirically, we showed that all MC-CBS variants outperform CBS by up to three orders of magnitude in terms of runtime, and the best variant also outperforms EPEA* in all cases and MDD-SAT in some cases.

There are many directions for future work. For example, we could study the problem that allows the rotation of shapes and develop sub-optimal LA-MAPF solvers, with the expectation that they would be more scalable than optimal ones.

References

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

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 *SoCS*, 19–27.

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

Dobson, A., and Bekris, K. E. 2014. Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research* 33(1):18–47.

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

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SoCS*, 29–37.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 83–87.

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

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *ICAPS*, 477–485.

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.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016a. Overview: Generalizations of multi-agent path finding to real-world scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

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

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, 270–272.

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

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2015. Heuristics for cost-optimal classical planning based on linear programming. In *Proceedings of the IJCAI*, 4303–4309.

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.

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

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

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

Veloso, M. M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 4423–4429.

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

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

Yu, J., and LaValle, S. M. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

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