On Satisfisfiability Modulo Theories in Continuous Multi-Agent Path Finding
compilation-based and search-based approaches compared

Pavel Surynek
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czech Republic
pavel.surynek@fit.cvut.cz

Keywords: multi-agent path finding (MAPF), satisfiability modulo theory (SMT), continuous time, continuous space, makespan optimal solutions, geometric agents

Abstract: Multi-agent path finding (MAPF) in continuous space and time with geometric agents, i.e. agents of various geometric shapes moving smoothly between predefined positions, is addressed in this paper. We analyze a new solving approach based on satisfiability modulo theories (SMT) that is designed to obtain makespan optimal solutions. The standard MAPF is a task of navigating agents in an undirected graph from given starting vertices to given goal vertices so that agents do not collide with each other in vertices or edges of the graph. In the continuous version (MAPF$^{\mathcal{R}}$), agents move in an n-dimensional Euclidean space along straight lines that interconnect predefined positions. Agents themselves are geometric objects of various shapes occupying certain volume of the space - circles, polygons, etc. For simplicity, we work with circular omni-directional agents having constant velocities in the 2D plane. As agents can have different shapes/sizes and are moving smoothly along lines, a movement along certain lines done with small agents can be non-colliding while the same movement may result in a collision if performed with larger agents. Such a distinction rooted in the geometric reasoning is not present in the standard MAPF. The SMT-based approach for MAPF$^{\mathcal{R}}$ called SMT-CBS$^{\mathcal{R}}$ reformulates the well established Conflict-based Search (CBS) algorithm in terms of SMT. Lazy generation of decision variables and constraints is the key idea behing SMT-CBS. Each time a new conflict is discovered, the underlying encoding is extended with new variables and constraints to eliminate the conflict. We compared SMT-CBS$^{\mathcal{R}}$ and adaptations of CBS for the continuous variant of MAPF experimentally.

1 Introduction and Background

In *multi-agent path finding* (MAPF) [Kornhauser et al., 1984, Ryan, 2008, Sharon et al., 2015, Sharon et al., 2013a, Silver, 2005, Surynek, 2009, Wang and Botea, 2011] the task is to navigate agents from given starting positions to given individual goals. The problem takes place in undirected graph $G = (V, E)$ where agents from set $A = \{a_1, a_2, ..., a_k\}$ are placed in vertices with at most one agent per vertex. The initial configuration of agents in vertices of the graph can be written as $\alpha_0 : A \to V$ and similarly the goal configuration as $\alpha_+ : A \to V$. The task of navigating agents can be expressed as a task of transforming the initial configuration of agents $\alpha_0 : A \to V$ into the goal configuration $\alpha_+ : A \to V$.

In the standard MAPF, movements are instantaneous and are possible into vacant neighbors assuming no other agent is entering the same target vertex [1]. We usually denote the configuration of agents at discrete time step t as $\alpha_t : A \to V$. Non-conflicting movements transform configuration α_t *instantaneously* into next configuration α_{t+1} so we do not consider what happens between t and $t+1$.

In order to reflect various aspects of real-life applications variants of MAPF have been introduced such as those considering *kinematic constraints* [Hönig et al., 2017], *large agents* [Li et al., 2019], or *deadlines* [Ma et al., 2018] - see [Ma et al., 2017] for more variants. Particularly in this work we are dealing with an extension of MAPF introduced only recently [Andreychuk et al., 2019, Walker et al., 2018] that consid-

[1] Different versions of MAPF permit entering of a vertex being simultaneously vacated by another agent excluding the trivial case when agents swap their position across an edge.

ers continuous time and space and continuous movements of agents between predefined positions placed arbitrarily in the n-dimensional Euclidean space. The continuous version will be denoted as MAPF$^{\mathcal{R}}$. It is natural in MAPF$^{\mathcal{R}}$ to assume geometric agents of various shapes that occupy certain volume in the space - circles in the 2D space, polygons, spheres in the 3D space etc. In contrast to MAPF, where the collision is defined as the simultaneous occupation of a vertex by two agents, collisions are defined as any spatial overlap of agents' bodies in MAPF$^{\mathcal{R}}$ or a an occurrence that is too close to each other. Agents move along straight lines connecting predefined positions. Different shapes of agents' bodies play a role. Hence for example a movement along two distinct lines that is collision free when done with small agents may turn into a collision if performed with large agents.

The motivation behind introducing MAPF$^{\mathcal{R}}$ is the need to construct more realistic paths in many applications such as controlling fleets of robots or aerial drones [Janovsky et al., 2014, Cáp et al., 2013] where continuous reasoning is closer to the reality than the standard MAPF.

The contribution of this paper consists in showing how to apply satisfiability modulo theory (SMT) reasoning [Bofill et al., 2012, Nieuwenhuis, 2010] in MAPF$^{\mathcal{R}}$ solving. The SMT paradigm constructs decision procedures for various complex logic theories by decomposing the decision problem into the propositional part having arbitrary Boolean structure and the theory part that is restricted on the conjunctive fragment.

1.1 Related Work and Organization

Our SMT-based approach focuses on *makespan optimal* MAPF solving and builds on top of the Conflict-based Search (CBS) algorithm [Sharon et al., 2012, Sharon et al., 2015]. Makespan optimal solutions minimize the overall time needed to relocate all agents into their goals.

CBS tries to solve MAPF lazily by adding conflict elimination constraints on demand. It starts with the empty set of constraints. The set of constraints is iteratively refined with new conflict elimination constraints after conflicts are found in solutions for the incomplete set of constraints. Since conflict elimination constraints are disjunctive (they forbid occurrence of one or the other agent in a vertex at a time) the refinement in CBS is carried out by branching in the search process.

CBS can be adapted for MAPF$^{\mathcal{R}}$ by implementing conflict detection in continuous time and space while the high-level framework of the CBS algorithm remains the same as shown in [Andreychuk et al., 2019]. In the SMT-based approach we are trying to build an *incomplete* propositional model so that if a given MAPF$^{\mathcal{R}}$ $\Sigma^{\mathcal{R}}$ has a solution of a specified makespan then the model is solvable (but the opposite implication generally does not hold). This is similar to the previous SAT-based [Biere et al., 2009] MAPF solving [Surynek, 2012, Surynek et al., 2016] where a *complete* propositional model has been constructed (that is, the given MAPF has a solution of a specified makespan if and only is the model is solvable).

The propositional model in the SMT-based approach in constructed *lazily* through conflict elimination refinements as done in CBS. The incompleteness of the model is inherited from CBS that adds constraints lazily. This is in contrast to SAT-based methods like MDD-SAT [Surynek et al., 2016] where all constraints are added *eagerly* resulting in a complete model. We call our new algorithm SMT-CBS$^{\mathcal{R}}$. The major difference of SMT-CBS$^{\mathcal{R}}$ from CBS is that instead of branching the search we only add a disjunctive constraint to eliminate the conflict in SMT-CBS$^{\mathcal{R}}$. Hence, SMT-CBS$^{\mathcal{R}}$ does not branch the search at all at the high-level (the model is incrementally refined at the high-level instead).

Similarly as in the SAT-based MAPF solving we use decision propositional variables indexed by *agent* a, *vertex* v, and *time* t with the meaning that if the variable is *TRUE* agent a appears in v at time t. However the major technical difficulty with the continuous version of MAPF is that we do not know all necessary decision variables in advance due to continuous time. After a conflict is discovered we may need new decision variables to avoid that conflict. For this reason we introduce a special decision variable generation algorithm.

The paper is organized as follows: we first introduce MAPF$^{\mathcal{R}}$ formally. Then we recall a variant of CBS for MAPF$^{\mathcal{R}}$. Details of the novel SMT-based solving algorithm SMT-CBS$^{\mathcal{R}}$ follow. Finally a preliminary experimental evaluation of SMT-CBS$^{\mathcal{R}}$ against the continuous version of CBS is shown. We also show a brief comparison with the standard MAPF.

1.2 MAPF with Continuous Time

We follow the definition of MAPF with continuous time denoted MAPF$^{\mathcal{R}}$ from [Andreychuk et al., 2019] and [Walker et al., 2018]. MAPF$^{\mathcal{R}}$ shares several components with the standard MAPF: the underlying undirected graph $G = (V, E)$, set of agents $A = \{a_1, a_2, ..., a_k\}$, and the initial and goal configuration of agents: $\alpha_0 : A \to V$ and $\alpha_+ : A \to V$.

Definition 1. *(**MAPF**$^{\mathcal{R}}$) Multi-agent path finding with continuous time (MAPF$^{\mathcal{R}}$) is a 5-tuple* $\Sigma^{\mathcal{R}} = (G=(V,E),A,\alpha_0,\alpha_+,\rho)$ *where G, A,* α_0, α_+ *are from the standard MAPF and* ρ *determines continuous extensions as follows:*

- $\rho.x(v), \rho.y(v)$ *for* $v \in V$ *represent the position of vertex v in the 2D plane; to simplify notation we will use* x_v *for* $\rho.x(v)$ *and* y_v *for* $\rho.x(v)$
- $\rho.velocity(a)$ *for* $a \in A$ *determines constant velocity of agent a; simple notation* $v_a = \rho.velocity(a)$
- $\rho.radius(a)$ *for* $a \in A$ *determines the radius of agent a; we assume that agents are circular discs with omni-directional ability of movements; simple notation* $r_a = \rho.radius(a)$

Naturally we can define the distance between a pair of vertices u, v with $\{u,v\} \in E$ as $dist(u,v) = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2}$. Next we assume that agents have constant speed, that is, they instantly accelerate to v_a from an idle state. The major difference from the standard MAPF where agents move instantly between vertices is that in MAPF$^{\mathcal{R}}$ continuous movement of an agent between a pair of vertices (positions) along the straight line interconnecting them takes place. Hence we need to be aware of the presence of agents at some point in the 2D plane on the lines interconnecting vertices at any time.

Collisions may occur between agents due to their size which is another difference from the standard MAPF. In contrast to the standard MAPF, collisions in MAPF$^{\mathcal{R}}$ may occur not only in a single vertex or edge but also on pairs of edges (on pairs of lines interconnecting vertices). If for example two edges are too close to each other and simultaneously traversed by large agents then such a condition may result in a collision. Agents collide whenever their bodies overlap [2].

We can further extend the set of continuous properties by introducing the direction of agents and the need to rotate agents towards the target vertex before they start to move towards the target (agents are no more omni-directional). The speed of rotation in such a case starts to play a role. Also agents can be of various shapes not only circular discs [Li et al., 2019]. For simplicity we elaborate our solving concepts for the above basic continuous extension of MAPF with circular agents only. We however note that all developed concepts can be adapted for MAPF with more continuous extensions like directional agents which only adds another dimension to indices of propositional variables.

[2] In our current implementation we followed a more cautious definition of the collision - it occurs even if agents appear too close to each other.

A solution to given MAPF$^{\mathcal{R}}$ $\Sigma^{\mathcal{R}}$ is a collection of temporal plans for individual agents $\pi = [\pi(a_1), \pi(a_2), ..., \pi(a_k)]$ that are mutually collision-free. A temporal plan for agent $a \in A$ is a sequence $\pi(a) = [((\alpha_0(a), \alpha_1(a)), [t_0(a), t_1(a)))$; $((\alpha_1(a), \alpha_2(a)), [t_1(a), t_2(a)))$; $...;((\alpha_{m(a)-1}, \alpha_{m(a)}(a)), (t_{m(a)-1}, t_{m(a)}))]$ where $m(a)$ is the length of individual temporal plan and each pair $(\alpha_i(a), \alpha_{i+1}(a)), [t_i(a), t_{i+1}(a)))$ in the sequence corresponds to traversal event between a pair of vertices $\alpha_i(a)$ and $\alpha_{i+1}(a)$ starting at time $t_i(a)$ and finished at $t_{i+1}(a)$ (excluding).

It holds that $t_i(a) < t_{i+1}(a)$ for $i = 0, 1, ..., m(a) - 1$. Moreover consecutive vertices must correspond to edge traversals or waiting actions, that is: $\{\alpha_i(a), \alpha_{i+1}(a)\} \in E$ or $\alpha_i(a) = \alpha_{i+1}(a)$; and times must reflect the speed of agents for non-wait actions, that is:

$$\alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow t_{i+1}(a) - t_i(a) = \frac{dist(\alpha_i(a), \alpha_{i+1}(a))}{v_a}.$$

In addition to this, agents must not collide with each other. One possible formal definition of a geometric collision is as follows:

Definition 2. *(**collision**) A* collision *between individual temporal plans* $\pi(a) = [((\alpha_i(a), \alpha_{i+1}(a)), [t_i(a), t_{i+1}(a)))]_{i=0}^{m(a)}$ *and* $\pi(b) = [((\alpha_i(b), \alpha_{i+1}(a)), [t_i(b), t_{i+1}(b)))]_{i=0}^{m(b)}$ *occurs if the following condition holds:*

- $\exists i \in \{0, 1, ..., m(a)\}$ *and* $\exists j \in \{0, 1, ..., m(b)\}$ *such that:*
  - $dist([x_{\alpha_i(a)}, y_{\alpha_i(a)}; x_{\alpha_{i+1}(a)}, y_{\alpha_{i+1}(a)}]; [x_{\alpha_j(b)}, y_{\alpha_j(b)}; x_{\alpha_{j+1}(b)}, y_{\alpha_{j+1}(b)}]) < r_a + r_b$
  - $[t_i(a), t_{i+1}(a)) \cap [t_j(b), t_{j+1}(b)) \neq \emptyset$

  (a vertex or an edge collision - two agents simultaneously occupy the same vertex or the same edge or traverse edges that are too close to each other)

The distance between two lines P and Q given by their endpoint coordinates $P = [x_1, y_1; x_2, y_2]$ and $Q = [x'_1, y'_1; x'_2, y'_2]$ denoted $dist([x_1, y_1; x_2, y_2]; [x'_1, y'_1; x'_2, y'_2])$ is defined as the minimum distance between any pair of points $p \in P$ and $q \in Q$: $min\{dist(p,q) \mid p \in P \wedge q \in Q\}$. The definition covers degenerate cases where a line collapses into a single point. In such a case the definition of *dist* normally works as the distance between points and between a point and a line.

The definition among other types of collisions covers also a case when an agent waits in vertex v and another agent passes through a line that is too close to v. We note that situations classified as collisions according to the above definition may not always result in actual collisions where agents' bodies overlap;

the definition is overcautious in this sense. Alternatively we can use more precise definition of collisions that reports collisions if and only if an actual overlap of agents' bodies occurs. This however requires more complex equations or simulations and cannot be written as simple as above. The presented algorithmic framework is however applicable for any kind of complex definition of collision as the definition enters the process as an external parameter.

The duration of individual temporal plan $\pi(a)$ is called an individual makespan; denoted $\mu(\pi(a)) = t_{m(a)}$. The overall makespan of MAPF$^{\mathcal{R}}$ solution $\pi = [\pi(a_1), \pi(a_2), ..., \pi(a_k)]$ is defined as $\max_{i=1}^{k}(\mu(\pi(a_i)))$. In this work we focus on finding makespan optimal solutions. An example of MAPF$^{\mathcal{R}}$ and makespan optimal solution is shown in Figure 1. We note that the standard makespan optimal solution yields makespan suboptimal solution when interpreted as MAPF$^{\mathcal{R}}$.

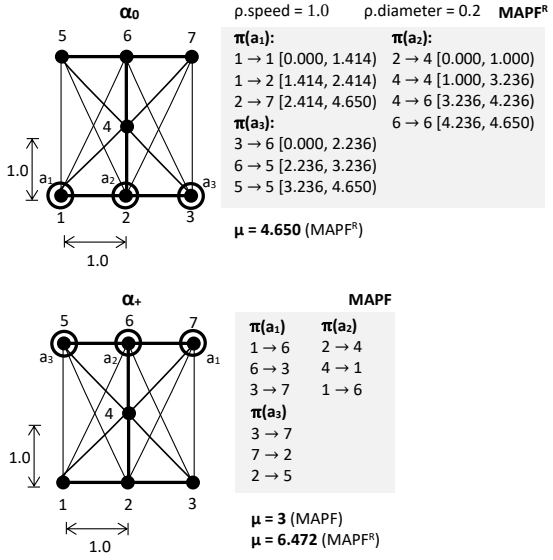

Figure 1: An example of MAPF$^{\mathcal{R}}$ instance on a $[3,1,3]-$graph with three agents and its makespan optimal solution (an optimal solution of the corresponding standard MAPF is shown too).

Through the straightforward reduction of MAPF to MAPF$^{\mathcal{R}}$ it can be observed that finding a makespan optimal solution with continuous time is an NP-hard problem [Ratner and Warmuth, 1990, Surynek, 2010, Yu and LaValle, 2015].

2 Solving MAPF with Continuous Time

We will describe here how to find optimal solution of MAPF$^{\mathcal{R}}$ using the *conflict-based search* (CBS) [Sharon et al., 2015]. CBS uses the idea of resolving conflicts lazily; that is, a solution of MAPF instance is not searched against the complete set of movement constraints. Instead of forbidding all possible collisions between agents we start with initially empty set of collision forbidding constraints that gradually grows as new conflicts appear. CBS originally developed for MAPF can be modified for MAPF$^{\mathcal{R}}$ as shown in [Andreychuk et al., 2019]: let us call the modification CBS$^{\mathcal{R}}$.

2.1 Conflict-based Search

CBS$^{\mathcal{R}}$ is shown using pseudo-code in Algorithm 1. The high-level of CBS$^{\mathcal{R}}$ searches a *constraint tree* (CT) using a priority queue (ordered according to the makespan or other cumulative cost) in the breadth first manner. CT is a binary tree where each node N contains a set of collision avoidance constraints $N.constraints$ - a set of triples $(a_i, \{u,v\}, [t_0, t_+))$ forbidding occurrence of agent a_i in edge $\{u,v\}$ (or in vertex u if $u = v$) at any time between $[t_0, t_+)$, a solution $N.\pi$ - a set of k individual temporal plans, and the makespan $N.\mu$ of the current solution.

The low-level process in CBS$^{\mathcal{R}}$ associated with node N searches temporal plan for individual agent with respect to set of constraints $N.constraints$. For given agent a_i, this is the standard single source shortest path search from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ that at time t must avoid a set of edges (vertices) $\{\{u,v\} \in E \mid (a_i, \{u,v\}, [t_0, t_+)) \in N.constraints \wedge t \in [t_0, t_+)\}$. Various intelligent single source shortest path algorithms can be applied here such as A* [Hart et al., 1968] .

CBS$^{\mathcal{R}}$ stores nodes of CT into priority queue OPEN sorted according to the ascending makespan. At each step CBS takes node N with the lowest makespan from OPEN and checks if $N.\pi$ represent non-colliding temporal plans. If there is no collision, the algorithms returns valid MAPF$^{\mathcal{R}}$ solution $N.\pi$. Otherwise the search branches by creating a new pair of nodes in CT - successors of N. Assume that a collision occurred between agents a_i and a_j when a_i traversed $\{u,v\}$ during $[t_0, t_+)$ and a_j traversed $\{u',v'\}$ during $[t'_0, t'_+)$. This collision can be avoided if either agent a_i or agent a_j does not occupy $\{u,v\}$ or $\{u',v'\}$ respectively during $[t_0, t_+) \cap [t'_0, t'_+) = [\tau_0, \tau_+)$. These two options correspond to new successor nodes of N: N_1 and N_2 that inherit set of conflicts

Algorithm 1: Basic CBS$^{\mathcal{R}}$ algorithm for solving MAPF with continuous time

CBS$^{\mathcal{R}}$ $(\Sigma^{\mathcal{R}} = (G = (V,E), A, \alpha_0, \alpha_+, \rho))$
  $R.constraints \leftarrow \emptyset$
  $R.\pi \leftarrow \{$shortest temporal plan from $\alpha_0(a_i)$ to $\alpha_+(a_i) \mid i = 1,2,...,k\}$
  $R.\mu \leftarrow \max_{i=1}^{k} \mu(N.\pi(a_i))$
  OPEN $\leftarrow \emptyset$
  insert R into OPEN
  while OPEN $\neq \emptyset$ **do**
    $N \leftarrow \min_{\mu}$(OPEN)
    remove-Min$_{\mu}$(OPEN)
    $collisions \leftarrow$ validate-Plans($N.\pi$)
    if $collisions = \emptyset$ **then**
      return $N.\pi$
    let $(a_i, \{u,v\}, [t_0, t_+)) \times (a_j, \{u',v'\}, [t'_0, t'_+)) \in collisions$
    $[\tau_0, \tau_+) \leftarrow [t_0, t_+) \cap [t'_0, t'_+)$
    for *each* $(a, \{w,z\}) \in \{(a_i, \{u,v\}), (a_j, \{u',v'\})\}$ **do**
      $N'.constraints \leftarrow N.constraints \cup \{(a, \{w,z\}, [\tau_0, \tau_+))\}$
      $N'.\pi \leftarrow N.\pi$
      update(a, $N'.\pi$, $N'.conflicts$)
      $N'.\mu \leftarrow \sum_{i=1}^{k} \mu(N'.\pi(a_i))$
      insert N' into OPEN

from N as follows: $N_1.conflicts = N.conflicts \cup \{(a_i, \{u,v\}, [\tau_0, \tau_+))\}$ and $N_2.conflicts = N.conflicts \cup \{(a_j, \{u',v'\}, [\tau_0, \tau_+))\}$. $N_1.\pi$ and $N_1.\pi$ inherit plans from $N.\pi$ except those for agent a_i and a_j respectively that are recalculated with respect to the new sets of conflicts. After this N_1 and N_2 are inserted into OPEN.

Definition of collisions comes as a parameter to the algorithm though the implementation of validate-Plans procedure. We can switch to the less cautious definition of collisions that reports a collision after agents actually overlap their bodies. This can be done through changing the validate-Plans procedure while the rest of the algorithm remains the same.

2.2 A Satisfiability Modulo Theory (SMT) Approach

A close look at CBS reveals that it operates similarly as problem solving in *satisfiability modulo theories* (SMT) [Bofill et al., 2012, Nieuwenhuis, 2010]. The basic use of SMT divides a satisfiability problem in some complex theory T into an abstract propositional part that keeps the Boolean structure of the decision problem and a simplified decision procedure $DECIDE_T$ that decides fragment of T restricted on *conjunctive formulae*. A general T-formula Γ being decided for satisfiability is transformed to a *propositional skeleton* by replacing its atoms with propositional variables. The standard SAT-solving procedure then decides what variables should be assigned *TRUE* in order to satisfy the skeleton - these variables tells what atoms hold in Γ. $DECIDE_T$ then checks if the conjunction of atoms assigned *TRUE* is valid with respect to axioms of T. If so then satisfying assignment is returned and we are finished. Otherwise a conflict from $DECIDE_T$ (often called a *lemma*) is reported back to the SAT solver and the skeleton is extended with new constraints resolving the conflict. More generally not only new constraints are added to resolve a conflict but also new variables i.e. atoms can be added to Γ.

The above observation inspired us to the idea to rephrase CBS$^{\mathcal{R}}$ in terms of SMT. T will be represented by a theory with axioms describing movement rules of MAPF$^{\mathcal{R}}$; a theory we will denote $T_{MAPF^{\mathcal{R}}}$ [3].

A plan validation procedure known from CBS will act as $DECIDE_{MAPF^{\mathcal{R}}}$ and will report back a set of conflicts found in the current solution. The propositional part working with the skeleton will be taken from existing propositional encodings of the standard MAPF such as the MDD-SAT [Surynek et al., 2016] provided that constraints forbidding conflicts between agents will be omitted (at the beginning). In other words, we only preserve constraints ensuring that propositional assignments form proper paths for agents but each agent is treated as if it is alone in the instance.

2.3 Decision Variable Generation

MDD-SAT introduces decision variables $\mathcal{X}_v^t(a_i)$ and $\mathcal{E}_{u,v}^t(a_i)$ for discrete time-steps $t \in \{0,1,2,...\}$ describing occurrence of agent a_i in v or the traversal of edge $\{u,v\}$ by a_i at time-step t. We refer the reader to [Surynek et al., 2016] for the details of how to encode constraints of top of these variables. As an example we show here a constraint stating that if agent a_i appears in vertex u at time step t then it has to leave through exactly one edge connected to u or wait in u.

$$\mathcal{X}_u^t(a_i) \Rightarrow \bigvee_{v \mid \{u,v\} \in E} \mathcal{E}_{u,v}^t(a_i) \vee \mathcal{E}_{u,u}^t(a_i), \qquad (1)$$

[3] The formal details of the theory $T_{MAPF^{\mathcal{R}}}$ are not relevant from the algorithmic point of view. Nevertheless let us note that the signature of $T_{MAPF^{\mathcal{R}}}$ consists of non-logical symbols describing agents' positions at a time such as $at(a,u,t)$ - agent a at vertex u at time t.

$$\sum_{v \mid \{u,v\} \in E} \mathcal{E}^t_{u,v}(a_i) + \mathcal{E}^t_{u,u}(a_i) \leq 1 \qquad (2)$$

Vertex collisions expressed for example by the following constraint are omitted. The constraint says that in vertex v and time step t there is at most one agent.

$$\sum_{a_i \in A \mid v \in V} \mathcal{X}^t_v(a_i) \leq 1 \qquad (3)$$

A significant difficulty in MAPF$^{\mathcal{R}}$ is that we need decision variables with respect to continuous time. Fortunately we do not need a variable for any possible time but only for important moments.

If for example the duration of a conflict in neighbor v of u is $[t_0, t_+)$ and agent a_i residing in u at $t \geq t_0$ wants to enter v then the earliest time a_i can do so is t_+ since before it would conflict in v (according to the above definition of collisions). On the other hand if a_i does not want to waste time (let us note that we search for a makespan optimal solution), then waiting longer than t_+ is not desirable. Hence we only need to introduce decision variable $\mathcal{E}^{t_+}_{u,v}(a_i)$ to reflect the situation.

Generally when having a set of conflicts we need to generate decision variables representing occurrence of agents in vertices and edges of the graph at important moments with respect to the set of conflicts. The process of decision variable generation is formally described as Algorithm 2. It performs breadth-first search (BFS) on G using two types of actions: *edge traversals* and *waiting*. The edge traversal is the standard operation from BFS. Waiting is performed for every relevant period of time with respect to the end-times in the set of conflicts of neighboring vertices.

As a result each conflict during variable generation through BFS is treated as both present and absent which in effect generates all possible important moments.

Procedure generate-Decision generates decision variables that correspond to actions started on or before specified limit μ_{max}. For example variables corresponding to edge traversal started at $t < \mu_{max}$ and finished as $t' > \mu_{max}$ are included (line 10). Variables corresponding to times greater than μ_{max} enable determining what should be the next relevant makespan limit to test (see the high-level algorithm for details). Assume having a decision node corresponding to vertex u at time t at hand. The procedure first adds decision variables corresponding to edge traversals from u to neighbors denoted v (lines 11-14). Then all possible relevant waiting actions in u with respect to its neighbors v are generated. Notice that waiting with respect to conflicts in u are treated as well.

Algorithm 2: Generation of decision variables in the SMT-based algorithm for MAPF$^{\mathcal{R}}$ solving

```
1  generate-Decisions
   (Σ^R = (G = (V,E), A, α_0, α_+, ρ), a_i, conflicts,
   μ_max)
2      VAR ← ∅
3      for each a ∈ A do
4          OPEN ← ∅
5          insert (α_0(a), 0) into OPEN
6          VAR ← VAR ∪ {X^{t_0}_{α_0(a)}(a)}
7          while OPEN ≠ ∅ do
8              (u,t) ← min_t(OPEN)
9              remove-Min_t(OPEN)
10             if t ≤ μ_max then
11                 for each v such that {u,v} ∈ E do
12                     Δt ← dist(u,v)/v_a
13                     insert (v, t+Δt) into OPEN
14                     VAR ←
                         VAR ∪ {E^t_{u,v}(a), X^{t+Δt}_v(a)}
15                 for each v such that
                     {u,v} ∈ E ∪ {u,u} do
16                     for each (a, {u,v}, [t_0,t_+)) ∈
                         conflicts do
17                         if t_+ > t then
18                             insert (u, t_+) into
                                 OPEN
19                             VAR ←
                                 VAR ∪ {X^{t_+}_u(a)}
20     return VAR
```

2.4 Eliminating Branching in CBS by Disjunctive Refinements

The SMT-based algorithm itself is divided into two procedures: SMT-CBS$^{\mathcal{R}}$ representing the main loop and SMT-CBS-Fixed$^{\mathcal{R}}$ solving the input MAPF$^{\mathcal{R}}$ for a fixed maximum makespan μ. The major difference from the standard CBS is that there is no branching at the high-level. The set of conflicts is iteratively collected during the entire execution of the algorithm whenever a collision is detected.

Procedures *encode-Basic* and *augment-Basic* build formula $\mathcal{F}(\mu)$ over decision variables generated using the aforementioned procedure. The encoding is inspired by the MDD-SAT approach but ignores collisions between agents. That is, $\mathcal{F}(\mu)$ constitutes an *incomplete model* for a given input $\Sigma^{\mathcal{R}}$: $\Sigma^{\mathcal{R}}$ is solvable within makespan μ then $\mathcal{F}(\mu)$ is satisfiable.

Conflicts are resolved by adding disjunctive constraints (lines 13-15 in Algorithm 4). The collision is avoided in the same way as in the original CBS that

is one of the colliding agent does not perform the action leading to the collision. Consider for example a collision on two edges between agents a_i and a_j as follows: a_i traversed $\{u,v\}$ during $[t_0,t_+)$ and a_j traversed $\{u',v'\}$ during $[t'_0,t'_+)$.

These two movements correspond to decision variables $\mathcal{E}^{t_0}_{u,v}(a_i)$ and $\mathcal{E}^{t'_0}_{u',v'}(a_j)$ hence elimination of the collision caused by these two movements can be expressed as the following disjunction: $\neg\mathcal{E}^{t_0}_{u,v}(a_i) \vee \neg\mathcal{E}^{t'_0}_{u',v'}(a_j)$. At level of the propositional formula there is no information about the semantics of a conflict happening in the continuous space; we only have information in the form of above disjunctive refinements. The disjunctive refinements are propagated at the propositional level from $DECIDE_{MAPF^{\mathcal{R}}}$ that verifies solutions of incomplete propositional models.

The set of pairs of collected disjunctive conflicts is propagated across entire execution of the algorithm (line 16 in Algorithm 4).

Algorithm 3: High-level of the SMT-based MAPF$^{\mathcal{R}}$ solving

```
1  SMT-CBS^R (Σ^R = (G = (V,E), A, α_0, α_+, ρ))
2      conflicts ← ∅
3      π ← {π*(a_i) a shortest temporal plan from
         α_0(a_i) to α_+(a_i) | i = 1,2,...,k}
4      μ ← max_{i=1}^{k} μ(π(a_i))
5      while TRUE do
6          (π, conflicts, μ_next) ←
             SMT-CBS-Fixed^R(Σ^R, conflicts, μ)
7          if π ≠ UNSAT then
8              return π
9          μ ← μ_next
```

Algorithm 3 shows the main loop of SMT-CBS$^{\mathcal{R}}$. The algorithm checks if there is a solution for given MAPF$^{\mathcal{R}}$ $\Sigma^{\mathcal{R}}$ of makespan μ. The algorithm starts at the lower bound for μ that is obtained as the duration of the longest temporal plan from individual temporal plans ignoring other agents (lines 3-4).

Then μ is iteratively increased in the main loop (lines 5-9). The algorithm relies on the fact that the solvability of MAPF$^{\mathcal{R}}$ w.r.t. cumulative objective like the makespan behaves as a non decreasing monotonic function. Hence trying increasing makespans eventually leads to finding the optimal makespan provided we do not skip any relevant makespan μ. The next makespan to try will then be obtained by taking the current makespan plus the smallest duration of the continuing movement (line 19 of Algorithm 4). The iterative scheme for trying larger makespans follows MDD-SAT [Surynek et al., 2016].

Algorithm 4: Low-level of the SMT-based MAPF$^{\mathcal{R}}$ solving

```
1  SMT-CBS-Fixed^R(Σ^R, conflicts, μ)
2      VAR ← generate-Decisions(Σ^R, conflicts, μ)
3      F(μ) ← encode-Basic(VAR, Σ^R, conflicts, μ)
4      while TRUE do
5          assignment ← consult-SAT-Solver(F(μ))
6          if assignment ≠ UNSAT then
7              π ← extract-Solution(assignment)
8              collisions ← validate-Plans(π)   /*
                 DECIDE_{MAPF^R} */
9              if collisions = ∅ then
10                 return (π, ∅, UNDEF)
11             for each (a_i, {u,v}, [t_0, t_+)) ×
                 (a_j, {u',v;}, [t'_0, t'_+)) ∈ collisions do
12                 Y ← (u = v) ? X^{t_0}_u(a_i) : E^{t_0}_{u,v}(a_i)
13                 Z ← (u' = v') ? X^{t'_0}_{u'}(a_j) :
                     E^{t'_0}_{u',v'}(a_j)
14                 F(μ) ← F(μ) ∪ {¬Y ∨ ¬Z}
15                 [τ_0, τ_+) ← [t_0, t_+) ∩ [t'_0, t'_+)
16                 conflicts ← conflicts ∪
                     {(a_i, {u,v}, [τ_0, τ_+)), (a_j, {u',v'}, [τ_0, τ_+))}
17             VAR ← generate-Decisions(Σ^R,
                 conflicts, μ)
18             F(μ) ← augment-
                 Basic(F(μ), VAR, Σ^R, conflicts, μ)
19     μ_next ← min{t | X^t_u(a_i) ∈ VAR ∧ t > μ)}
20     return (UNSAT, conflicts, μ_next)
```

3 Experimental Evaluation

In this section we present results of the first preliminary experimentation with SMT-CBS$^{\mathcal{R}}$. We implemented SMT-CBS$^{\mathcal{R}}$ in C++ to evaluate its performance [4] SMT-CBS$^{\mathcal{R}}$ was implemented on top of Glucose 4 SAT solver [Audemard and Simon, 2009] which ranks among the best SAT solvers according to recent SAT solver competitions [Balyo et al., 2017]. Whenever possible the SAT solver was consulted in the incremental mode.

It turned out to be important to generate decision variables in a more advanced way than presented in Algorithm 2. We need to prune out decisions from that the goal vertex cannot be reached under given makespan bound μ_{max}. That is whenever we have a decision (u,t) such that $t+\Delta t > \mu_{max}$, where $\Delta t = dist_{estimate}(u,\alpha_+(a))/v_a$ and $dist_{estimate}$ is a lower bound estimate of the distance between a pair of vertices, we rule out that decision from further con-

[4] The complete source codes will be made available to enable reproducibility of presented results on the author's website: ...

sideration. Moreover we apply a postprocessing step in which we iteratively remove decisions that have no successors. The propositional model is generated only after this preprocessing.

In addition to SMT-CBS$^{\mathcal{R}}$ we re-implemented in C++ CBS$^{\mathcal{R}}$, currently the only alternative solver for MAPF$^{\mathcal{R}}$ based on own dedicated search [Andreychuk et al., 2019]. The distinguishing feature of CBS$^{\mathcal{R}}$ is that at the low-level it uses a more complex single source shortest path algorithm that searches for paths that avoid forbidden intervals, a so-called *safe-interval path planning* (SIPP) [Yakovlev and Andreychuk, 2017].

Our implementation of CBS$^{\mathcal{R}}$ used the standard heuristics to improve the performance such as the preference of resolving *cardinal conflicts* [Boyarski et al., 2015]. Without this heuristic, CBS$^{\mathcal{R}}$ usually exhibited poor performance. In the preliminary tests with SMT-CBS$^{\mathcal{R}}$, we initially tried to resolve against single cardinal conflict too but eventually it turned out to be more efficient to resolve against all discovered conflicts (the presented pseudo-code shows this variant). [5]

3.1 Benchmarks and Setup

SMT-CBS$^{\mathcal{R}}$ and CBS$^{\mathcal{R}}$ were tested on synthetic benchmarks consisting of *layered graphs*, *grids*, game maps [Sturtevant, 2012]. The layered graph of height h denoted $[l_1, l_2, ..., l_h]$-graph consists of h layers of vertices placed horizontally above each other in the 2D plane (see Figure 1 for $[3,1,3]$-graph). More precisely the i-th layer is placed horizontally at $y = i$. Layers are centered horizontally and the distance between consecutive points in the layer is 1.0. Size of all agents was 0.2 in radius.

We measured runtime and the number of decisions/iterations to compare the performance of SMT-CBS$^{\mathcal{R}}$ and CBS$^{\mathcal{R}}$. Small layered graphs consisting of 2 to 5 layers with up to 5 vertices per layer were used in tests. Three consecutive layers are always fully interconnected by edges. There is not edge across more than three layers of the graphs. That is in graphs with more than 3 layers agents cannot go directly to the goal vertex.

In all tests agents started in the 1-st layer and finished in the last h-th layer. To obtain instances of various difficulties random permutations of agents in the starting and goal configurations were used (the 1-st layer and h-th layer were fully occupied in the starting and goal configuration respectively). If for instance agents are ordered identically in the starting and goal configuration with $h \leq 3$, then the instance is relatively easy as it is sufficient that all agents move simultaneously straight into their goals.

We also used grids of sizes 8×8 and 16×16 with no obstacles in our tests. Initial and goal configuration of agents have been generated randomly. In contrast to MAPF benchmarks where grids are 4-connected we used interconnection with all vertices in the neighborhood up to certain distance called 2^k-neighborhood in [Andreychuk et al., 2019]. A similar setup has been used in game maps (Dragon Age). The difference here is that the game maps are larger and contain obstacles.

Ten random instances were generated for individual graph. The timeout for all tests has been set to 1 minute in layered graphs and small grids and 10 minutes for game maps. Results from instances finished under this timeout were used to calculate average runtimes.

3.2 Comparison of MAPF$^{\mathcal{R}}$ and MAPF Solving

Part of the results obtained in our experimentation with layered graphs is shown in Figure 2. The general observation from our runtime evaluation is that MAPF$^{\mathcal{R}}$ is significantly harder than the standard MAPF. When continuity is ignored, makespan optimal solutions consist of fewer steps. But due to regarding all edges to be unit in MAPF, the standard makespan optimal solutions yield significantly worse continuous makespan (this effect would be further manifested if we use longer edges).

SMT-CBS$^{\mathcal{R}}$ outperforms CBS$^{\mathcal{R}}$ on tested instances significantly. CBS$^{\mathcal{R}}$ reached the timeout many more times than SMT-CBS$^{\mathcal{R}}$. In the absolute runtimes, SMT-CBS$^{\mathcal{R}}$ is faster by factor of 2 to 10 than CBS$^{\mathcal{R}}$.

In terms of the number of decisions, SMT-CBS$^{\mathcal{R}}$ generates order of magnitudes fewer iterations than CBS$^{\mathcal{R}}$. This is however expected as SMT-CBS$^{\mathcal{R}}$ shrinks the entire search tree into a single branch in fact. We note that branching within the search space in case of SMT-CBS$^{\mathcal{R}}$ is deferred into the SAT solver where even more branches may appear.

In case of small grids and large maps (Figures 3, 4 and 5), the difference between CBS$^{\mathcal{R}}$ and SMT-CBS$^{\mathcal{R}}$ is generally smaller but still for harder instances SMT-CBS$^{\mathcal{R}}$ tends to have better runtime and success rate. We attribute the smaller difference between the two algorithms to higher regularity in grids compared to layered graphs that exhibit higher combinatorial difficulty.

[5] All experiments were run on a system with Ryzen 7 3.0 GHz, 16 GB RAM, under Ubuntu Linux 18.

Average **runtime** and **makespan** (**μ**) on selected layered graphs

Graph	CBS^R	SMT-CBS^R	μ MAPF^R	CBS	μ MAPF
[2,2]	2.78	**1.22**	2.41	0.01	2.00
[3,1,3]	17.91	**2.33**	3.65	0.02	2.75
[4,2,2,4]	19.34	**4.78**	3.80	0.02	2.67
[5,3,1,3,5]	57.23	**6.11**	6.78	0.03	3.15
[5,3,5,3,5]	-	**19.93**	5.39	0.03	3.75

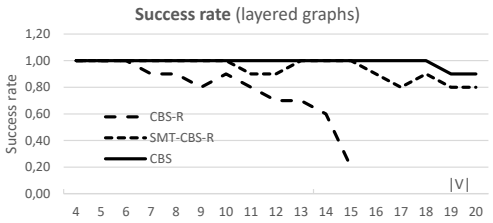

Figure 2: Comparison of CBS$^{\mathcal{R}}$ and SMT-CBS$^{\mathcal{R}}$ in terms of average runtime, makespan, and success rate on layered graphs. The standard CBS on the corresponding standard MAPF is shown too (times are in seconds). Makespan is shown for the case when the instance is interpreted as the standard MAPF and as MAPF$^{\mathcal{R}}$.

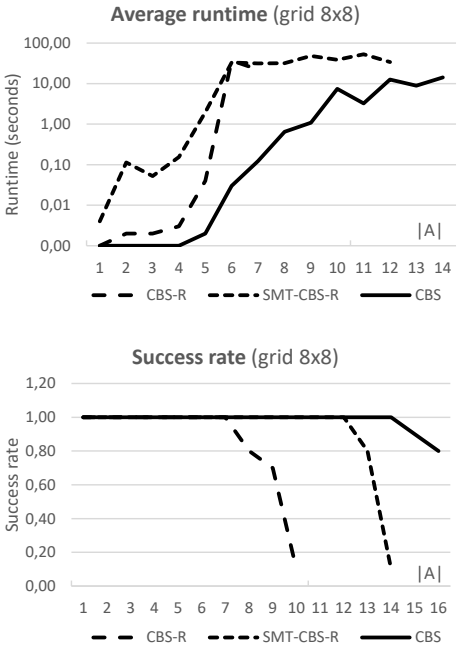

Figure 3: Comparison of CBS$^{\mathcal{R}}$ and SMT-CBS$^{\mathcal{R}}$ on 8×8 grid with 2^k neighborhood ($k = 3$).

4 Discussion and Conclusion

We suggested a novel approach for the makespan optimal solving of the multi-agent path finding problem with continuous time and space based on *satisfiability modulo theories* (SMT). Our approach is builds on the idea of treating constraints lazily as suggested in the CBS algorithm but instead of branching the search after encountering a conflict we refine the propositional model with the conflict elimination disjunctive constraint. The major obstacle in using SMT and propositional reasoning is that decision variables cannot be determined in advance straightforwardly in the continuous case. We hence suggested a novel decision variable generation approach that enumerates new decisions after discovering new conflicts. The propositional model is iteratively solved by the SAT solver. We call the novel algorithm SMT-CBS$^{\mathcal{R}}$.

We compared SMT-CBS$^{\mathcal{R}}$ to the only previous approach for MAPF$^{\mathcal{R}}$ that modifies the standard CBS algorithm [Andreychuk et al., 2019] and uses dedicated search on a number of benchmarks. The outcome of our comparison is that SMT-CBS$^{\mathcal{R}}$ is faster than CBS$^{\mathcal{R}}$. We observed the best performance of SMT-CBS$^{\mathcal{R}}$ on layered graphs that constitute combinatorially difficult case on a small graph - this is the case where SMT-CBS$^{\mathcal{R}}$ relies mostly on the SAT solver and less on decision variable generation and other other high-level mechanisms. We attribute the better runtime results of SMT-CBS$^{\mathcal{R}}$ to more efficient handling of disjunctive conflicts in the underlying SAT solver through *propagation*, *clause learning*, and other mechanisms.

Comparison with the standard MAPF version indicate that continuous reasoning is harder to solve but on the other hand provides more realistic solutions.

For the future work we assume extending the concept of SMT-based approach for MAPF$^{\mathcal{R}}$ with other cumulative cost functions other than the makespan such as the *sum-of-costs* [Sharon et al., 2013b]. We also plan to extend the node generation scheme to directional agents where we need to add the third dimension in addition to space (vertices) and time: *direction* (angle). We also regards the work in MAPF$^{\mathcal{R}}$ as stepping-stone to multi-robot motion planning in continuous configuration spaces [LaValle, 2006].

REFERENCES

Andreychuk, A., Yakovlev, K., Atzmon, D., and Stern, R. (2019). Multi-agent pathfinding (MAPF) with continuous time. *CoRR*, abs/1901.05506.

Audemard, G. and Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, pages 399–404.

Balyo, T., Heule, M. J. H., and Järvisalo, M. (2017). SAT competition 2016: Recent developments. In *AAAI*, pages 5061–5063.

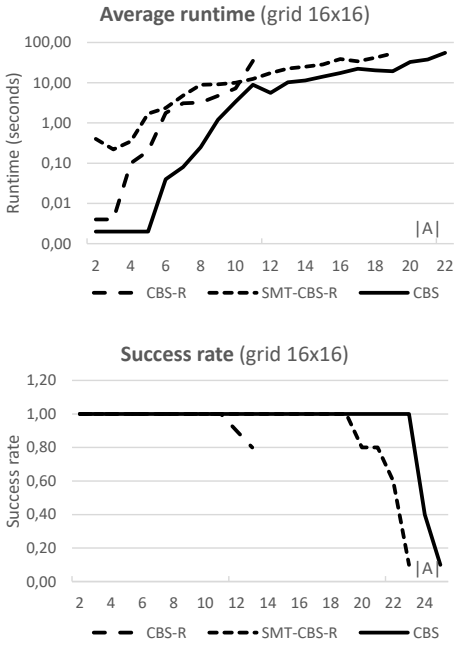

Figure 4: Comparison of CBS$^{\mathcal{R}}$ and SMT-CBS$^{\mathcal{R}}$ on 16×16 grid with 2^k neighborhood ($k = 3$).

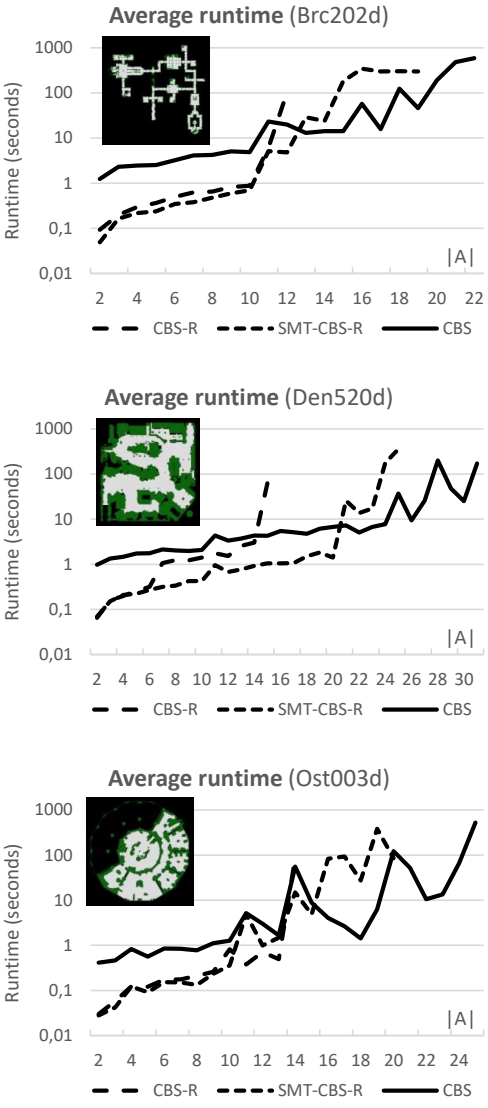

Figure 5: Comparison of CBS$^{\mathcal{R}}$ and SMT-CBS$^{\mathcal{R}}$ on game maps (Dragon Age) with 2^k neighborhood ($k = 3$).

Biere, A., Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bofill, M., Palahí, M., Suy, J., and Villaret, M. (2012). Solving constraint satisfaction problems with SAT modulo theories. *Constraints*, 17(3):273–303.

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

Cáp, M., Novák, P., Vokrínek, J., and Pechoucek, M. (2013). Multi-agent RRT: sampling-based cooperative pathfinding. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, 2013*, pages 1263–1264.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107.

Hönig, W., Kumar, T. K. S., Cohen, L., Ma, H., Xu, H., Ayanian, N., and Koenig, S. (2017). Summary: Multi-agent path finding with kinematic constraints. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4869–4873.

Janovsky, P., Cáp, M., and Vokrínek, J. (2014). Finding coordinated paths for multiple holonomic agents in 2-d polygonal environment. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, 2014*, pages 1117–1124.

Kornhauser, D., Miller, G. L., and Spirakis, P. G. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS, 1984*, pages 241–250.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Li, J., Surynek, P., Felner, A., Ma, H., and Koenig, S. (2019). Multi-agent path finding for large agents. In *AAAI*. AAAI Press.

Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T. K. S., Uras, T., Xu, H., Tovey, C. A., and Sharon, G. (2017). Overview: Generalizations

of multi-agent path finding to real-world scenarios. *CoRR*, abs/1702.05515.

Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T. K. S., and Koenig, S. (2018). Multi-agent path finding with deadlines. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 417–423.

Nieuwenhuis, R. (2010). SAT modulo theories: Getting the best of SAT and global constraint filtering. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 1–2.

Ratner, D. and Warmuth, M. K. (1990). Nxn puzzle and related relocation problem. *J. Symb. Comput.*, 10(2):111–138.

Ryan, M. R. K. (2008). Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res. (JAIR)*, 31:497–542.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2012). Conflict-based search for optimal multi-agent path finding. In *AAAI*.

Sharon, G., Stern, R., Goldenberg, M., and Felner, A. (2013a). The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, 195:470–495.

Sharon, G., Stern, R., Goldenberg, M., and Felner, A. (2013b). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495.

Silver, D. (2005). Cooperative pathfinding. In *AIIDE*, pages 117–122.

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

Surynek, P. (2009). A novel approach to path planning for multiple robots in bi-connected graphs. In *ICRA 2009*, pages 3613–3619.

Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In *AAAI 2010*. AAAI Press.

Surynek, P. (2012). Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI 2012*, pages 564–576. Springer.

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*, pages 810–818.

Walker, T. T., Sturtevant, N. R., and Felner, A. (2018). Extended increasing cost tree search for non-unit cost domains. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018.*, pages 534–540.

Wang, K. and Botea, A. (2011). MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *JAIR*, 42:55–90.

Yakovlev, K. and Andreychuk, A. (2017). Any-angle pathfinding for multiple agents based on SIPP algorithm. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, page 586.

Yu, J. and LaValle, S. M. (2015). Optimal multi-robot path planning on graphs: Structure and computational complexity. *CoRR*, abs/1507.03289.