

On Satisfiability 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^R), agents move in a metric space along predefined trajectories 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 where positions are interconnected by straight lines. 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^R called SMT-CBS^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 behind 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^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 $A = \{a_1, a_2, \dots, a_k\}$ are placed in vertices so that there is at most one agent per vertex. The initial configuration of agents in vertices of the graph can be written as simple assignment $\alpha_0 : A \rightarrow V$ and similarly the goal configuration as $\alpha_+ : A \rightarrow V$. The task of navigating agents can be then expressed as a task of transforming the initial configuration of agents $\alpha_0 : A \rightarrow V$ into the goal configuration $\alpha_+ : A \rightarrow V$.

In the standard MAPF, movements are instantaneous and are usually possible into vacant neighbors

assuming no other agent is entering the same target vertex¹. We usually denote the configuration of agents at discrete time step t as $\alpha_t : A \rightarrow V$. Non-conflicting movements transform configuration α_t *instantaneously* into next configuration α_{t+1} so we do not consider what happens between time steps 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., 2019b; Andreychuk et al.,

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

2019a; Walker et al., 2018) that considers continuous time and space and continuous movements of agents between predefined positions placed arbitrarily in the metric space. The continuous version will be denoted as $\text{MAPF}^{\mathcal{R}}$. It is natural in $\text{MAPF}^{\mathcal{R}}$ 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 usually defined as the simultaneous occupation of a vertex by two (or more) agents, collisions are defined as any spatial overlap of agents' bodies in $\text{MAPF}^{\mathcal{R}}$. Agents in $\text{MAPF}^{\mathcal{R}}$ move along predefined trajectories that interconnects predefined positions. Assuming certain individual speed of an agent we can assign the agent a position in the space at any moment. Different shapes of agents' bodies play a role. Hence for example a movement along two distinct trajectories that is collision free when done with small agents may turn into a collision if performed with large agents. For simplicity we will assume agents moving along straight lines but the presented techniques are applicable to different trajectories as well.

The motivation behind introducing $\text{MAPF}^{\mathcal{R}}$ 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 $\text{MAPF}^{\mathcal{R}}$ 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 $\text{MAPF}^{\mathcal{R}}$ 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., 2019b; Andreychuk et al., 2019a).

In the SMT-based approach we are trying to build an *incomplete* propositional model so that if a given $\text{MAPF}^{\mathcal{R}}$ $\Sigma^{\mathcal{R}}$ 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 if the model is solvable).

The propositional model in the SMT-based approach is constructed *lazily* through conflict elimination refinements as done in CBS. The lazy approach of propositional model construction for the standard MAPF was first introduced in (Surynek, 2019). Here we further generalize the SMT-based approach for $\text{MAPF}^{\mathcal{R}}$. Similar techniques for lazy introduction of constraints are also known in integer programming and were successfully applied in the standard MAPF solving as well (Lam et al., 2019).

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 $\text{SMT-CBS}^{\mathcal{R}}$. The major difference of $\text{SMT-CBS}^{\mathcal{R}}$ from CBS is that instead of branching the search we only add a disjunctive constraint to eliminate the conflict in $\text{SMT-CBS}^{\mathcal{R}}$. Hence, $\text{SMT-CBS}^{\mathcal{R}}$ does not branch the search at all at the high-level. The propositional 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 that generates decision variables on demand.

The paper is organized as follows: we first introduce $\text{MAPF}^{\mathcal{R}}$ formally. Then we recall a variant of CBS for $\text{MAPF}^{\mathcal{R}}$. Details of the novel SMT-based

solving algorithm SMT-CBS^R follow. Finally, an experimental evaluation of SMT-CBS^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^R from (Andreychuk et al., 2019b) and (Walker et al., 2018). MAPF^R shares several components with the standard MAPF: the underlying undirected graph $G = (V, E)$, set of agents $A = \{a_1, a_2, \dots, a_k\}$, and the initial and goal configuration of agents: $\alpha_0 : A \rightarrow V$ and $\alpha_+ : A \rightarrow V$.

Definition 1. (MAPF^R) *Multi-agent path finding with continuous time (MAPF^R) is a 5-tuple $\Sigma^R = (G = (V, E), A, \alpha_0, \alpha_+, \rho)$ 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 $\rho.y(v)$
- $\rho.velocity(a)$ 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^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.

As we will see in the definition of collisions having predefined positions interconnected by straight lines is used for simplification only. Any fixed trajectory that allows us to map an agent from continuous time to a position in the trajectory is possible.

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^R may occur not only in a single vertex or edge but also on pairs of edges (on pairs of lines/trajectories interconnecting vertices). If for example two lines 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 overlap².

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 translation movement 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.

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

It holds that $t_i(a) < t_{i+1}(a)$ for all $i = 0, 1, \dots, m(a) - 1$. 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 $\pi(b) = [((\alpha_j(b), \alpha_{j+1}(b)), [t_j(b), t_{j+1}(b)])]_{j=0}^{m(b)}$ occurs if the following condition holds:

- $\exists i \in \{0, 1, \dots, m(a)\}$ and $\exists j \in \{0, 1, \dots, m(b)\}$ such that:

$$- dist([x_{\alpha_i(a)}, y_{\alpha_i(a)}; x_{\alpha_{i+1}(a)}, y_{\alpha_{i+1}(a)}]; [x_{\alpha_j(b)}, y_{\alpha_j(b)}; x_{\alpha_{j+1}(b)}, y_{\alpha_{j+1}(b)}]) < r_a + r_b$$

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

$$- [t_i(a), t_{i+1}(a)) \cap [t_j(b), t_{j+1}(b)) \neq \emptyset$$

(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^R solution $\pi = [\pi(a_1), \pi(a_2), \dots, \pi(a_k)]$ 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^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^R.

Through the straightforward reduction of MAPF to MAPF^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^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 in-

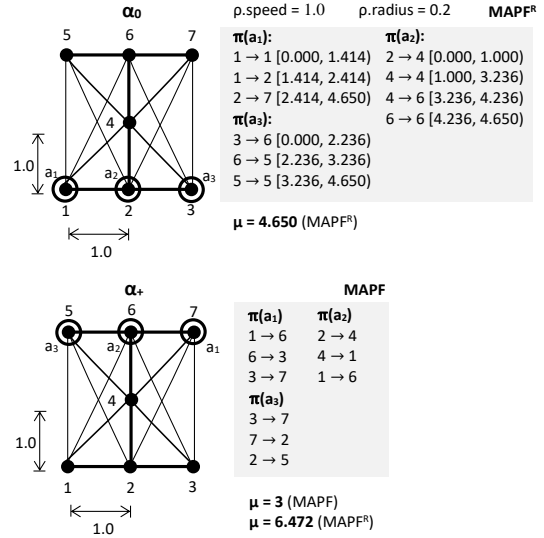


Figure 1: An example of MAPF^R instance on a $[3, 1, 3]$ -graph with three agents and its makespan optimal solution (an optimal solution of the corresponding standard MAPF is shown too).

stance 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 that was originally developed for MAPF, can be modified for MAPF^R as shown in (Andreychuk et al., 2019a): let us denote the modification CBS^R.

2.1 Conflict-Based Search

CBS^R is shown using pseudo-code in Algorithm 1. The high-level of CBS^R searches a *constraint tree* (CT) using a priority queue (ordered according to the makespan or other cumulative cost) in the breadth first search 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^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).

Algorithm 1: Basic CBS^R algorithm for solving MAPF with continuous time.

```

1 CBSR( $\Sigma^R = (G = (V, E), A, \alpha_0, \alpha_+, \rho)$ )
2    $R.constraints \leftarrow \emptyset$ 
3    $R.\pi \leftarrow \{\text{shortest temporal plan from } \alpha_0(a_i) \text{ to } \alpha_+(a_i) \mid i = 1, 2, \dots, k\}$ 
4    $R.\mu \leftarrow \max_{i=1}^k \mu(N.\pi(a_i))$ 
5   OPEN  $\leftarrow \emptyset$ 
6   insert  $R$  into OPEN
7   while OPEN  $\neq \emptyset$  do
8      $N \leftarrow \min_{\mu}(\text{OPEN})$ 
9     remove- $\text{Min}_{\mu}(\text{OPEN})$ 
10     $collisions \leftarrow \text{validate-Plans}(N.\pi)$ 
11    if  $collisions = \emptyset$  then
12      return  $N.\pi$ 
13    let  $(a_i, \{u, v\}, [t_0, t_+]) \times (a_j, \{u', v'\}, [t'_0, t'_+]) \in collisions$ 
14     $[\tau_0, \tau_+] \leftarrow [t_0, t_+] \cap [t'_0, t'_+]$ 
15    for each  $(a, \{w, z\}) \in \{(a_i, \{u, v\}), (a_j, \{u', v'\})\}$ 
16      do
17         $N'.constraints \leftarrow N.constraints \cup \{(a, \{w, z\}, [\tau_0, \tau_+])\}$ 
18         $N'.\pi \leftarrow N.\pi$ 
19        update( $a, N'.\pi, N'.conflicts$ )
20         $N'.\mu \leftarrow \sum_{i=1}^k \mu(N'.\pi(a_i))$ 
21        insert  $N'$  into OPEN

```

CBS^R stores nodes of CT into priority queue OPEN sorted according to the ascending makespan. At each step the algorithm 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 algorithm returns valid MAPF^R solution $N.\pi$. Otherwise the search branches by creating a new pair of nodes in CT - successors of N , each resolving the collision by diverting one of the colliding agents. 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 $[t_0, t_+) \cap [t'_0, t'_+) = [\tau_0, \tau_+)$ or more precisely we can calculate unsafe time interval for each agent so that whenever the agent commences the movement during this time interval the movement will result in a collision assuming the other agent following its original plan. These two options correspond to new successor nodes of N : N_1 and N_2 that inherit set of conflicts 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 $N_2.\pi$ 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 the 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. In a more general case, not only new constraints are added to resolve the conflict but also new variables i.e. atoms can be added to Γ .

The above observation inspired us to the idea to rephrase CBS^R in terms of SMT similarly as it has been previously done with CBS and its reformulation in SMT suggested in (Surynek, 2019). T will be represented by a theory with axioms describing movement rules of MAPF^R; a theory we will denote T_{MAPF^R} ³.

A plan validation procedure known from CBS will act as $DECIDE_{MAPF^R}$ 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 stan-

³The formal details of the theory T_{MAPF^R} are not relevant from the algorithmic point of view so we will omit them here. Nevertheless let us note that the signature of T_{MAPF^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 .

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 (we only forbid agents to jump but we ignore collisions between them).

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 $t \in \{0, 1, 2, \dots\}$ 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 .

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

$$\sum_{v | \{u,v\} \in E} \mathcal{E}_{u,v}^t(a_i) + \mathcal{E}_{u,u}^t(a_i) \leq 1 \quad (2)$$

Vertex collisions expressed for example by the following constraint are omitted when the encoding is being built lazily in the SMT-style. The constraint says that in vertex v and time step t there is at most one agent.

$$\sum_{a_i \in A | v \in V} \mathcal{X}_v^t(a_i) \leq 1 \quad (3)$$

A significant difficulty in MAPF^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 $\mathcal{E}_{u,v}^{t_+}(a_i)$ 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:

Algorithm 2: Generation of decision variables in the SMT-based algorithm for MAPF^R solving.

```

1 generate-Decisions
   ( $\Sigma^{\mathcal{R}} = (G = (V, E), A, \alpha_0, \alpha_+, \rho), a_i, conflicts,$ 
    $\mu_{max}$ )
2   VAR  $\leftarrow \emptyset$ 
3   for each  $a \in A$  do
4     OPEN  $\leftarrow \emptyset$ 
5     insert  $(\alpha_0(a), 0)$  into OPEN
6     VAR  $\leftarrow$  VAR  $\cup \{\mathcal{X}_{\alpha_0(a)}^{t_0}(a)\}$ 
7     while OPEN  $\neq \emptyset$  do
8        $(u, t) \leftarrow \text{min}_t(\text{OPEN})$ 
9       remove-Mint(OPEN)
10      if  $t \leq \mu_{max}$  then
11        for each  $v$  such that  $\{u, v\} \in E$  do
12           $\Delta t \leftarrow \text{dist}(u, v) / v_a$ 
13          insert  $(v, t + \Delta t)$  into OPEN
14          VAR  $\leftarrow$ 
15            VAR  $\cup \{\mathcal{E}_{u,v}^t(a), \mathcal{X}_v^{t+\Delta t}(a)\}$ 
16          for each  $v$  such that
17             $\{u, v\} \in E \cup \{u, u\}$  do
18            for each  $(a, \{u, v\}, [t_0, t_+)) \in$ 
19              conflicts do
20                if  $t_+ > t$  then
21                  insert  $(u, t_+)$  into
22                    OPEN
23                  VAR  $\leftarrow$ 
24                    VAR  $\cup \{\mathcal{X}_u^{t_+}(a)\}$ 
25  return VAR

```

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 how the makespan limit is used). 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.

2.4 Eliminating Branching in CBS by Disjunctive Refinements

The SMT-based algorithm itself is divided into two procedures: SMT-CBS^R representing the main loop and SMT-CBS-Fixed^R solving the input MAPF^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 extended during the entire execution of the algorithm whenever a collision is detected.

Procedures *encode-Basic* and *augment-Basic* build formula $\mathcal{F}(\mu)$ over the 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 Σ^R : Σ^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 is diverted from performing 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 $E_{u,v}^{t_0}(a_i)$ and $E_{u',v'}^{t'_0}(a_j)$ hence elimination of the collision caused by these two movements can be expressed as the following disjunction: $\neg E_{u,v}^{t_0}(a_i) \vee \neg E_{u',v'}^{t'_0}(a_j)$. At the level of 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 $DECIDE_{MAPF^R}$ 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 shows the main loop of SMT-CBS^R. The algorithm checks if there is a solution for given MAPF^R Σ^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^R w.r.t. cumulative objective like the makespan behaves as a non decreasing monotonic function. Hence trying increasing makespans eventu-

Algorithm 3: High-level of the SMT-based MAPF^R solving.

```

1 SMT-CBSR ( $\Sigma^R = (G = (V, E), A, \alpha_0, \alpha_+, \rho)$ )
2    $conflicts \leftarrow \emptyset$ 
3    $\pi \leftarrow \{\pi^*(a_i) \text{ a shortest temporal plan from } \alpha_0(a_i) \text{ to } \alpha_+(a_i) \mid i = 1, 2, \dots, k\}$ 
4    $\mu \leftarrow \max_{i=1}^k \mu(\pi(a_i))$ 
5   while TRUE do
6      $(\pi, conflicts, \mu_{next}) \leftarrow$ 
7       SMT-CBS-FixedR( $\Sigma^R, conflicts, \mu$ )
8     if  $\pi \neq UNSAT$  then
9       return  $\pi$ 
9      $\mu \leftarrow \mu_{next}$ 

```

ally leads to finding the optimal makespan in case of solvable MAPF^R 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 the idea used in MDD-SAT (Surynek et al., 2016) and before introduced in SAT-based classical planning algorithm SATPlan (Kautz and Selman, 1992).

3 EXPERIMENTAL EVALUATION

In this section we present results of the first experimentation with SMT-CBS^R. We implemented SMT-CBS^R in C++ to evaluate its performance⁴.

SMT-CBS^R was implemented on top of Glucose 3 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 consideration. Moreover we apply a postprocessing step in which we iteratively remove decisions that have

⁴The complete source codes will be made available to enable reproducibility of presented results on the author's website: <http://users.fit.cvut.cz/surynpav/research/icaart2020>.

Algorithm 4: Low-level of the SMT-based MAPF^R solving.

```

1 SMT-CBS-FixedR( $\Sigma^R, conflicts, \mu$ )
2   VAR  $\leftarrow$  generate-Decisions( $\Sigma^R, conflicts, \mu$ )
3    $\mathcal{F}(\mu) \leftarrow$  encode-Basic(VAR,  $\Sigma^R, conflicts, \mu$ )
4   while TRUE do
5     assignment  $\leftarrow$  consult-SAT-Solver( $\mathcal{F}(\mu)$ )
6     if assignment  $\neq$  UNSAT then
7        $\pi \leftarrow$  extract-Solution(assignment)
8       collisions  $\leftarrow$  validate-Plans( $\pi$ )
9       /* DECIDEMAPFR */
10      if collisions =  $\emptyset$  then
11        return ( $\pi, \emptyset, UNDEF$ )
12      for each ( $a_i, \{u, v\}, [t_0, t_+)$ )  $\times$ 
13        ( $a_j, \{u', v'\}, [t'_0, t'_+)$ )  $\in$  collisions do
14           $\mathcal{Y} \leftarrow (u = v) ? X_u^{t_0}(a_i) : \mathcal{E}_{u,v}^{t_0}(a_i)$ 
15           $\mathcal{Z} \leftarrow (u' = v') ? X_{u'}^{t'_0}(a_j) :$ 
16             $\mathcal{E}_{u',v'}^{t'_0}(a_j)$ 
17           $\mathcal{F}(\mu) \leftarrow \mathcal{F}(\mu) \cup \{\neg \mathcal{Y} \vee \neg \mathcal{Z}\}$ 
18           $[\tau_0, \tau_+) \leftarrow [t_0, t_+) \cap [t'_0, t'_+)$ 
19          conflicts  $\leftarrow$  conflicts  $\cup$ 
20            {( $a_i, \{u, v\}, [\tau_0, \tau_+)$ ),
21              ( $a_j, \{u', v'\}, [\tau_0, \tau_+)$ )}
22      VAR  $\leftarrow$  generate-Decisions( $\Sigma^R,$ 
23        conflicts,  $\mu$ )
24       $\mathcal{F}(\mu) \leftarrow$  augment-
25        Basic( $\mathcal{F}(\mu), VAR, \Sigma^R, conflicts, \mu$ )
26
27   $\mu_{next} \leftarrow \min\{t \mid X_u^t(a_i) \in VAR \wedge t > \mu\}$ 
28  return (UNSAT, conflicts,  $\mu_{next}$ )

```

no successors. The propositional model is generated only after this preprocessing.

In addition to SMT-CBS^R we re-implemented in C++ CBS^R, currently the only alternative solver for MAPF^R based on own dedicated search (Andreychuk et al., 2019a). The distinguishing feature of CBS^R is that at the low-level it uses a more sophisticated 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^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^R usually exhibited poor performance. In the preliminary tests with SMT-CBS^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

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

3.1 Benchmarks and Setup

SMT-CBS^R and CBS^R were tested on synthetic benchmarks consisting of *layered graphs*, *grids*, game maps (Sturtevant, 2012). The layered graph of height h denoted $[l_1, l_2, \dots, l_h]$ -graph is a special construct enabling detailed study of movements of agents in continuous space. The layered 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^R and CBS^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 no 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., 2019a). 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^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^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).

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

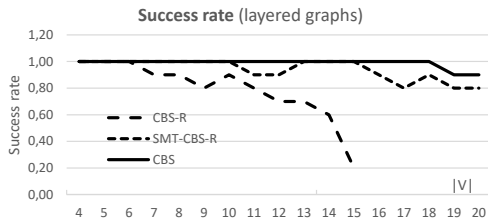


Figure 2: Comparison of CBS^R and SMT-CBS^R in terms of average runtime, makespan, and success rate on layered graphs. The standard CBS on the corresponding standard MAPF is shown too (times are in seconds). Makespan is shown for the case when the instance is interpreted as the standard MAPF and as MAPF^R.

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

In terms of the number of decisions, SMT-CBS^R generates order of magnitudes fewer iterations than CBS^R. This is however expected as SMT-CBS^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^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^R and SMT-CBS^R is generally smaller but still for harder instances SMT-CBS^R tends to have better runtime and success rate. We attribute the smaller difference between the two algorithms to higher regularity in grids

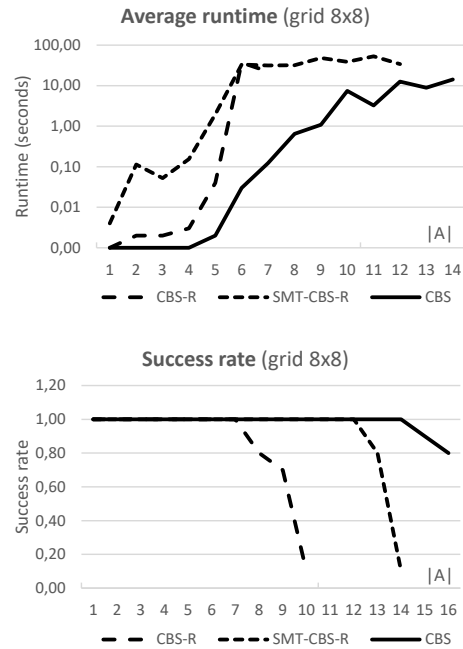


Figure 3: Comparison of CBS^R and SMT-CBS^R on 8×8 grid with 2^k neighborhood ($k = 3$).

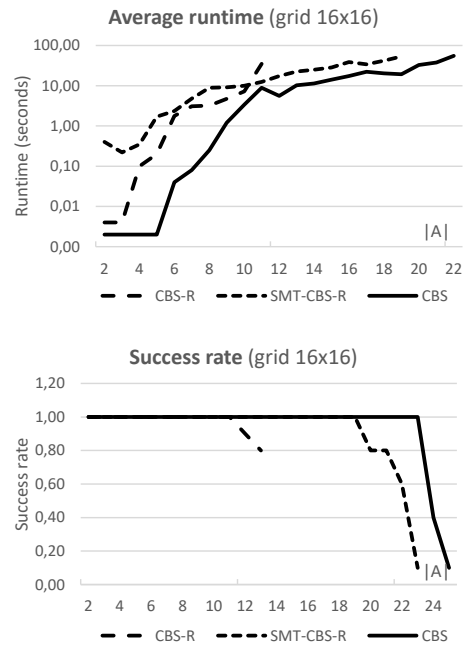


Figure 4: Comparison of CBS^R and SMT-CBS^R on 16×16 grid with 2^k neighborhood ($k = 3$).

compared to layered graphs that exhibit higher combinatorial difficulty.

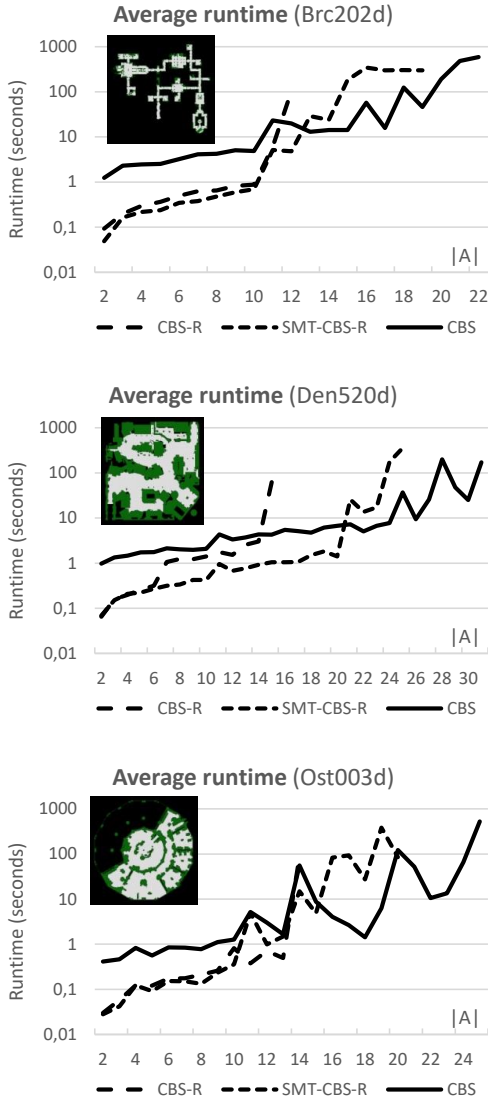


Figure 5: Comparison of $\text{CBS}^{\mathcal{R}}$ and $\text{SMT-CBS}^{\mathcal{R}}$ on game maps (Dragon Age) with 2^k neighborhood ($k = 3$).

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 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 $\text{SMT-CBS}^{\mathcal{R}}$.

We compared $\text{SMT-CBS}^{\mathcal{R}}$ to the only previous approach for $\text{MAPF}^{\mathcal{R}}$ that modifies the standard CBS algorithm (Andreychuk et al., 2019a) and uses dedicated search on a number of benchmarks. The outcome of our comparison is that $\text{SMT-CBS}^{\mathcal{R}}$ is faster than $\text{CBS}^{\mathcal{R}}$. We observed the best performance of $\text{SMT-CBS}^{\mathcal{R}}$ on layered graphs that constitute combinatorially difficult case on a small graph - this is the case where $\text{SMT-CBS}^{\mathcal{R}}$ 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 $\text{SMT-CBS}^{\mathcal{R}}$ 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.

Closer analysis of assumptions introduced by the definition of $\text{MAPF}^{\mathcal{R}}$ shows the importance of predefined interconnections between positions in the space. Currently neither $\text{CBS}^{\mathcal{R}}$ nor $\text{SMT-CBS}^{\mathcal{R}}$ can handle agents that are moving freely in the space along any trajectory. Such a less restrictive variant of continuous MAPF takes place in completely different search space and hence searching for a solution would require different reasoning.

For the future work we assume extending the concept of SMT-based approach for $\text{MAPF}^{\mathcal{R}}$ 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 regard the work in $\text{MAPF}^{\mathcal{R}}$ as stepping-stone to multi-robot motion planning in continuous configuration spaces (LaValle, 2006).

ACKNOWLEDGEMENTS

This paper has been supported by the Czech Science Foundation (application number 19-17966S). The author would like to thank anonymous reviewers for their effort to provide valuable comments.

REFERENCES

- Andreychuk, A., Yakovlev, K., Atzmon, D., and Stern, R. (2019a). Multi-agent pathfinding (MAPF) with continuous time. *CoRR*, abs/1901.05506.
- Andreychuk, A., Yakovlev, K. S., Atzmon, D., and Stern, R. (2019b). Multi-agent pathfinding with continuous time. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 39–45. ijcai.org.
- 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.
- 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.
- Boffill, M., Palahí, M., Suy, J., and Villaret, M. (2012). Solving constraint satisfaction problems with SAT modulo theories. *Constraints*, 17(3):273–303.
- Boyariski, 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., Vokřínek, 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 Vokřínek, 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.
- Kautz, H. A. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI 1992*, pages 359–363. John Wiley and Sons.
- 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.
- Lam, E., Bodic, P. L., Harabor, D. D., and Stuckey, P. J. (2019). Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, 2019*, pages 1289–1296. ijcai.org.
- 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 *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, 2005*, pages 117–122. AAAI Press.
- 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 *2009 IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 3613–3619. IEEE.
- Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press.
- Surynek, P. (2012). Towards optimal cooperative path planning in hard setups through satisfiability solving.

In *PRICAI 2012: Trends in Artificial Intelligence - 12th Pacific Rim International Conference on Artificial Intelligence, 2012. Proceedings*, volume 7458 of *Lecture Notes in Computer Science*, pages 564–576. Springer.

- Surynek, P. (2019). Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, 2019*, pages 1177–1183. ijcai.org.
- 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.*, page 586.
- Yu, J. and LaValle, S. M. (2015). Optimal multi-robot path planning on graphs: Structure and computational complexity. *CoRR*, abs/1507.03289.