Logic-Based Multi-Agent Path Finding with Continuous Movements and the Sum of Costs Objective

Pavel Surynek[0000−0001−7200−0542]

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

Abstract. Multi-agent path finding with continuous movements and time (denoted MAPF$^{\mathcal{R}}$) is addressed. The task is to navigate agents that move smoothly between predefined positions to their individual goals so that they do not collide. Recently a novel solving approach for obtaining makespan optimal solutions called SMT-CBS$^{\mathcal{R}}$ based on *satisfiability modulo theories* (SMT) has been introduced. We extend the approach further towards the sum-of-costs objective which is a more challenging case in the yes/no SMT environment due to more complex calculation of the objective. The new algorithm combines collision resolution known from conflict-based search (CBS) with previous generation of incomplete propositional encodings on top of a novel scheme for selecting decision variables in a potentially uncountable search space. We experimentally compare SMT-CBS$^{\mathcal{R}}$ and previous CCBS algorithm for MAPF$^{\mathcal{R}}$.

Keywords: path finding, multiple agents, robotic agents, logic reasoning, satisfiability modulo theory, sum-of-costs optimality

1 Introduction

In *multi-agent path finding* (MAPF) [15,27,24,30,37,26,25,6] 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 navigation task can be then expressed formally as transforming an initial configuration of agents $\alpha_0 : A \to V$ to a goal configuration $\alpha_+ : A \to V$ using instantaneous movements across edges assuming no collision occurs.

To reflect various aspects of real-life applications, variants of MAPF have been introduced such as those considering *kinematic constraints* [9], *large agents* [17], *generalized costs* of actions [36], or *deadlines* [19] - see [18,28] for more variants. Particularly in this work we are dealing with an extension of MAPF introduced only recently [1,33] that considers continuous movements and time (MAPF$^{\mathcal{R}}$). Agents move smoothly along predefined curves interconnecting predefined positions placed arbitrarily in some continuous space. 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 or an edge by two agents, collisions are defined as any spatial overlap of agents' bodies in MAPF$^{\mathcal{R}}$.

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 [7,10] where continuous reasoning is closer to the reality than the standard MAPF.

The contribution of this paper consists in generalizing the previous makespan optimal approach for MAPF$^{\mathcal{R}}$ [31,33] that uses satisfiability modulo theory (SMT) reasoning [5,20] for the sum-of-costs objective. 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 complex theory part that is restricted on the conjunctive fragment. Our SMT-based algorithm called SMT-CBS$^{\mathcal{R}}$ combines the Conflict-based Search (CBS) algorithm [8,25] with previous algorithms for solving the standard MAPF using incomplete encodings [32] and continuous reasoning.

1.1 Previous Work

Using reductions of planning problems to propositional satisfiability has been coined in the SATPlan algorithm and its variants [11,12,13,14]. Here we are trying to apply similar idea in the context of MAPF$^{\mathcal{R}}$. So far MAPF$^{\mathcal{R}}$ has been solved by a modified version of CBS that tries to solve MAPF lazily by adding collision avoidance constraints on demand. The adaptation of CBS for MAPF$^{\mathcal{R}}$ consists in implementing continuous collision detection while the high-level framework of the algorithm remains the same as demonstrated in the CCBS algorithm [1].

We follow the idea of CBS too but instead of searching the tree of possible collision eliminations at the high-level we encode the requirement of having collision free paths as a propositional formula [4] and leave it to the SAT solver as done in [34]. We construct the formula *lazily* by adding collision elimination refinements following [32] where the lazy construction of incomplete encodings has been suggested for the standard MAPF within the algorithm called SMT-CBS. SMT-CBS works with propositional variables indexed by *agent a*, *vertex v*, and *time step t* with the meaning that if the variable is *TRUE a* in *v* at time step *t*. In MAPF$^{\mathcal{R}}$ we however face major technical difficulty that we do not know necessary decision (propositional) variables in advance and due to continuous time we cannot enumerate them all. Hence we need to select from a potentially uncountable space those variables that are sufficient for finding the solution.

The previous application of SMT in MAPF$^{\mathcal{R}}$ [33] focused on the makespan optimal solutions where the shortest duration of the plan is required. The **sum-of-costs** is another important objective used in the context of MAPF [26,36]. Calculated as the summation over all agents of times they spend moving before arriving to the goal. Due to its more complex calculation, the sum-of-costs objective is more challenging to be integrated in the SMT-based solving framework.

1.2 MAPF with Continuous Movements and Time

We use the definition of MAPF with continuous movements and time denoted MAPF$^{\mathcal{R}}$ from [1]. MAPF$^{\mathcal{R}}$ shares components with the standard MAPF: 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 $\alpha_+ : A \to V$. A simple 2D variant of MAPF$^{\mathcal{R}}$ is as follows:

Definition 1. *(**MAPF**$^{\mathcal{R}}$) Multi-agent path finding with continuous time and space 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:*

- $\rho.x(v), \rho.y(v)$ *for* $v \in V$ *represent the position of vertex v in the 2D plane*
- $\rho.speed(a)$ *for* $a \in A$ *determines constant speed of agent 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*

For simplicity we assume circular agents with constant speed and instant acceleration. The major difference from the standard MAPF where agents move instantly between vertices (disappears in the source and appears in the target instantly) is that smooth continuous movement between a pair of vertices (positions) along the straight line interconnecting them takes place in MAPF$^{\mathcal{R}}$. Hence we need to be aware of the presence of agents at some point in the 2D plane at any time.

Collisions may occur between agents in MAPF$^{\mathcal{R}}$ due to their volume; that is, they collide whenever their bodies **overlap**. In contrast to MAPF, collisions in MAPF$^{\mathcal{R}}$ may occur not only in a single vertex or edge being shared by colliding agents but also on pairs of edges (lines interconnecting vertices) that are too close to each other and simultaneously traversed by large agents.

We can further extend the continuous properties by introducing the direction of agents and the need to rotate agents towards the target vertex before they start to move. Also agents can be of various shapes not only circular discs [17] and can move along various fixed curves. For simplicity we elaborate our implementations for the above simple 2D continuous extension with circular agents. We however note that all developed concepts can be adapted for MAPF with more continuous extensions.

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 $\pi(a) = [((\alpha_0(a), \alpha_1(a)), [t_0(a), t_1(a))); ((\alpha_1(a), \alpha_2(a)), [t_1(a), t_2(a))); ...; ((\alpha_{m(a)-1}(a), \alpha_{m(a)}(a)), [t_{m(a)-1}(a), t_{m(a)}(a)))]$ whe re $m(a)$ is the length of individual temporal plan and $t_{m(a)}$ is its duration. Each pair $(\alpha_i(a), \alpha_{i+1}(a)), [t_i(a), t_{i+1}(a)))$ 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 $t_{i+1}(a)$.

It holds that $t_i(a) < t_{i+1}(a)$ for $i = 0, 1, ..., m(a) - 1$. Moreover consecutive events in the individual temporal plan 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.

The duration of individual temporal plan $\pi(a)$ is called an *individual makespan*; denoted $\mu(\pi(a)) = t_{m(a)}$. The overall *makespan* of π is defined as $\max_{i=1}^{k} \{\mu(\pi(a_i))\}$. The individual makespan is sometimes called an *individual cost*. A *sum-of-cost* for given

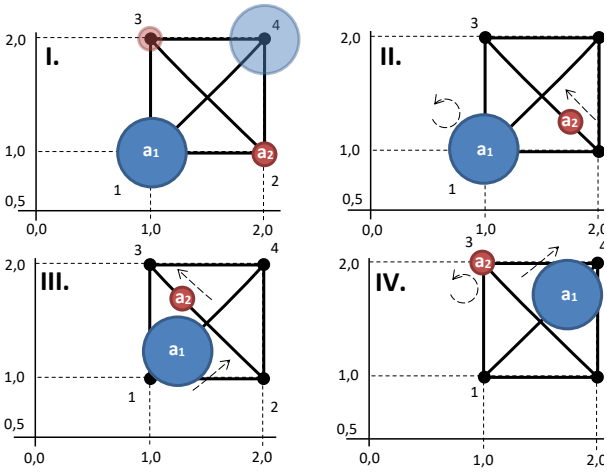

Fig. 1. An example of MAPF$^{\mathcal{R}}$ instance with two agents. A feasible makespan/sum-of-costs sub-optimal solution π (makespan $\mu(\pi) = 2.0$) and makespan/sum-of-costs optimal solution $\pi*$ (makespan $\mu(\pi*) = 1.980$) are shown.

temporal plan $\pi(a)$ is defined as $\sum_{i=1}^{k} \mu(\pi(a_i))$ An example of MAPF$^{\mathcal{R}}$ and makespan/sum-of-costs optimal solution is shown in Figure 1.

Through straightforward reduction of MAPF to MAPF$^{\mathcal{R}}$ it can be observed that finding a makespan or sum-of-costs optimal solution with continuous time is an NP-hard problem [22,38].

2 Solving MAPF with Continuous Time

Let us recall CCBS [1], a variant of CBS [25] modified for MAPF$^{\mathcal{R}}$. The idea of CBS algorithms is to resolve conflicts lazily.

2.1 Conflict-based Search

CCBS for finding the sum-of-costs optimal solution is shown in Algorithm 1. The high-level of CCBS searches a *constraint tree* (CT) using a priority queue ordered according to the sum-of-costs in the breadth first manner. CT is a binary tree where each node N contains a set of collision avoidance constraints $N.cons$ - a set of triples $(a_i, (u,v), [\tau_0, \tau_+))$ forbidding agent a_i to start smooth traversal of edge $\{u,v\}$ (line) at

Algorithm 1: CCBS algorithm for solving MAPF$^{\mathcal{R}}$ for the sum-of-costs objective.

CBS$^{\mathcal{R}}$ ($\Sigma^{\mathcal{R}} = (G = (V,E), A, \alpha_0, \alpha_+, \rho)$)
  $R.cons \leftarrow \emptyset$
  $R.\pi \leftarrow \{$shortest temporal plan from $\alpha_0(a_i)$ to $\alpha_+(a_i) \mid i = 1,2,...,k\}$
  $R.\xi \leftarrow \sum_{i=1}^{k} \mu(N.\pi(a_i))$
  OPEN $\leftarrow \emptyset$
  insert R into OPEN
  while OPEN $\neq \emptyset$ **do**
    $N \leftarrow \min_{\xi}$(OPEN)
    remove-Min$_{\xi}$(OPEN)
    $collisions \leftarrow$ validate-Plans($N.\pi$)
    if $collisions = \emptyset$ **then**
      return $N.\pi$
    let $(m_i \times m_j) \in collisions$ where $m_i = (a_i, (u_i, v_i), [t_i^0, t_i^+))$ and $m_j = (a_j, (u_j, v_j), [t_j^0, t_j^+))$
    $([\tau_i^0, \tau_i^+); [\tau_j^0, \tau_j^+)) \leftarrow$ resolve-Collision(m_i,m_j)
    for *each* $m \in \{(m_i, [\tau_i^0, \tau_i^+)), (m_j, [\tau_j^0, \tau_j^+))\}$ **do**
      let $((a, (u,v), [t_0, t_+)), [\tau_0, \tau_+)) = m$
      $N'.cons \leftarrow N.cons \cup \{(a, (u,v), [\tau_0, \tau_+))\}$
      $N'.\pi \leftarrow N.\pi$
      update(a, $N'.\pi$, $N'.cons$)
      $N'.\xi \leftarrow \sum_{i=1}^{k} \mu(N'.\pi(a_i))$
      insert N' into OPEN

any time between $[\tau_0, \tau_+)$, a solution $N.\pi$ - a set of k individual temporal plans, and the sum-of-costs $N.\xi$ of $N.\pi$.

The low-level in CCBS associated with node N searches for individual temporal plan with respect to set of constraints $N.cons$. 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 cannot start to traverse any $\{(u,v) \in E \mid (a_i, (u,v), [\tau_0, \tau_+)) \in N.cons \wedge t \in [\tau_0, \tau_+)\}$. Various intelligent single source shortest path algorithms such as SIPP [21] can be used here.

CCBS stores nodes of CT into priority queue OPEN sorted according to the ascending sum-of-costs. At each step CBS takes node N with the lowest makespan from OPEN and checks if $N.\pi$ represents non-colliding temporal plans. If there is no collision, the algorithms returns valid 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 a_i traversing (u_i, v_i) during $[t_i^0, t_i^+)$ and a_j traversing (u_j, v_j) during $[t_j^0, t_j^+)$. This collision can be avoided if either agent a_i or agent a_j waits after the other agent passes. We can calculate for a_i so called maximum *unsafe interval* $[\tau_i^0, \tau_i^+)$ such that whenever a_i starts to traverse (u_i, v_i) at some time $t \in [\tau_i^0, \tau_i^+)$ it ends up colliding with a_j assuming a_j did not try to avoid the collision. Hence a_i should wait until τ_i^+ to tightly avoid the collision with a_j. Similarly we can calculate maximum unsafe interval for a_j: $[\tau_j^0, \tau_j^+)$. These two

options correspond to new successor nodes of N: N_1 and N_2 that inherit set of constraints from N as follows: $N_1.cons = N.cons \cup \{(a_i,(u_i,v_i),[\tau_i^0,\tau_i^+))\}$ and $N_2.cons = N.cons \cup \{(a_j,(u_j,v_j),[\tau_j^0,\tau_j^+))\}$. $N_1.\pi$ and $N_1.\pi$ inherits plans from $N.\pi$ except those for agents a_i and a_j respectively that are recalculated with respect to the constraints. After this N_1 and N_2 are inserted into OPEN.

2.2 A Satisfiability Modulo Theory Approach

A recent algorithm called SMT-CBS$^{\mathcal{R}}$ [33] rephrases CCBS as problem solving in *satisfiability modulo theories* (SMT) [5,35]. The basic use of SMT divides the satisfiability problem in some complex theory T into a propositional part that keeps the Boolean structure of the problem and a simplified 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 solver 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. 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 the conflict but also new atoms can be added to Γ.

T will be represented by a theory with axioms describing movement rules of MAPF$^{\mathcal{R}}$; a theory we will denote $T_{MAPF^{\mathcal{R}}}$. $DECIDE_{MAPF^{\mathcal{R}}}$ can be naturally represented by the plan validation procedure from CCBS (validate-Plans).

2.3 RDD: Real Decision Diagram

The key question in the propositional logic-based approach is what will be the decision variables. In the standard MAPF, time expansion of G for every time step can be done resulting in a multi-value decision diagram (MDD) [34] representing possible positions of agents at any time step. Since MAPF$^{\mathcal{R}}$ is no longer discrete we cannot afford to use a decision variable for every time moment. We show how to restrict the decision variables on finitely many important moments only without compromising soundness nor optimality of the approach.

Analogously to MDD, we introduce *real decision diagram* (RDD). RDD$_i$ defines for agent a_i its space-time positions and possible movements. Formally, RDD_i is a directed graph (X^i,E^i) where X_i consists of pairs (u,t) with $u \in V$ and $t \in \mathbb{R}_0^+$ is time and E_i consists of directed edges of the form $((u,t_u);(v,t_v))$. Edge $((u,t_u);(v,t_v))$ correspond to agent's movement from u to v started at t_u and finished at t_v. Waiting in u is possible by introducing edge $((u,t_u);(u,t'_u))$. Pair $(\alpha_0(a_i),0) \in X_i$ indicates start and $(\alpha_+(a_i),t)$ for some t corresponds to reaching the goal position.

RDDs for individual agents are constructed with respect to collision avoidance constraints. If there is no collision avoidance constraint then RDD$_i$ simply corresponds to a shortest temporal plan for agent a_i. But if a collision avoidance constraint is present, say $(a_i,(u,v),[\tau_0,\tau_+))$, and we are considering movement starting in u at t that interferes with the constraint, then we need to generate a node into RDD$_i$ that allows agent

to wait until the unsafe interval passes by, that is node (u,τ^+) and edge $((u,\tau^0);(u,\tau^+))$ are added.

Similarly for wait constraints $(a_i,(u,u),[\tau_0,\tau_+))$ that forbid waiting in u during $[\tau_0,\tau_+)$. In such a case, we need to anticipate the constraint before entering u, that is we can wait until $\tau_+ - t_x$ in the source vertex before entering u where t_x is the time needed to traverse the edge towards u.

The process of building RDDs is described in details in [33]. An example of RDDs is shown in Figure 2.

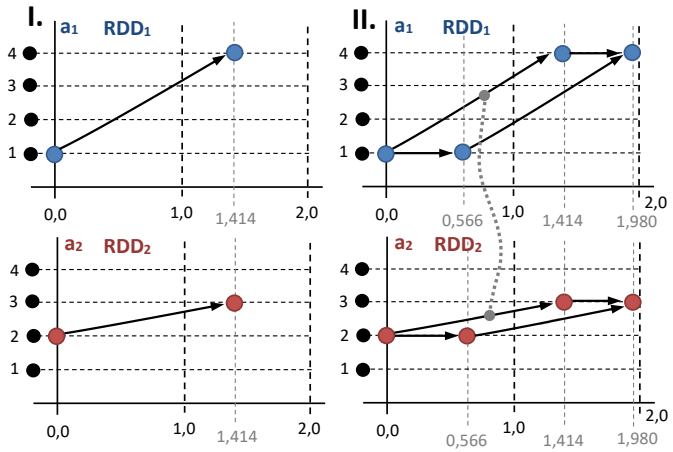

Fig. 2. Real decision diagrams (RDDs) for agents a_1 and a_2 from MAPF$^{\mathcal{R}}$ from Figure 1. Decisions corresponding to shortest paths for agents a_1 and a_2 moving diagonally towards their goals are shown: $a_1 : 1 \to 4$, $a_2 : 2 \to 3$ (left). This however results in a collision whose resolution is either waiting for agent a_1 in vertex 1 from 0.000 until 0.566 or waiting for agent a_2 in vertex 2 from 0.000 until 0.566; reflected in the next RDDs (right). Mutex is depicted using dotted line connecting arcs form RDD_1 and RDD_2.

2.4 SAT Encoding from RDD

We introduce a decision variable for each node and edge $[RDD_1,...,RDD_k]$; $\text{RDD}_i = (X^i,E^i)$: we have variable $\mathcal{X}_u^t(a_i)$ for each $(u,t) \in X^i$ and $\mathcal{E}_{u,v}^{t_u,t_v}(a_i)$ for each directed edge $((u,t_u);(v,t_v)) \in E^i$. The meaning of variables is that $\mathcal{X}_u^t(a_i)$ is *TRUE* if and only if agent a_i appears in u at time t and similarly for edges: $\mathcal{E}_{u,v}^{t_u,t_v}(a_i)$ is *TRUE* if and only if a_i moves from u to v starting at time t_u and finishing at t_v.

MAPF$^{\mathcal{R}}$ rules are encoded on top of these variables so that eventually we want to obtain formula $\mathcal{F}(\mu)$ that encodes existence of a solution of makespan μ to given MAPF$^{\mathcal{R}}$. We need to encode that agents do not skip but move along edges, do not disappear or appear from nowhere etc. We show below constraints stating that if agent a_i appears in vertex u at time step t_u then it has to leave through exactly one edge connected to u (constraint (2) although Pseudo-Boolean can be encoded using purely propositional means):

$$\mathcal{X}_u^{t_u}(a_i) \Rightarrow \bigvee_{(v,t_v) \mid ((u,t_u),(v,t_v)) \in E^i} \mathcal{E}_{u,v}^{t_u,t_v}(a_i), \tag{1}$$

$$\sum_{(v,t_v) \mid ((u,t_u),(v,t_v)) \in E^i} \mathcal{E}_{u,v}^{t_u,t_v}(a_i) \leq 1 \tag{2}$$

$$\mathcal{E}_{u,v}^{t_u,t_v}(a_i) \Rightarrow \mathcal{X}_v^{t_v}(a_i) \tag{3}$$

Analogously to (2) we have constraint allowing a vertex to accept at most one agent through incoming edges; plus we need to enforce agents starting in α_0 and finishing in α_+. Let us summarize soundness of the encoding in the following proposition (proof omitted).

Proposition 1. *Any satisfying assignment of $\mathcal{F}(\mu)$ correspond to valid individual temporal plans for $\Sigma^{\mathcal{R}}$ whose makespans are at most μ.*

We a-priori do not add constraints for eliminating collisions; these are added lazily after assignment/solution validation. Hence, $\mathcal{F}(\mu)$ constitutes an *incomplete model* for $\Sigma^{\mathcal{R}}$: $\Sigma^{\mathcal{R}}$ is solvable within makespan μ then $\mathcal{F}(\mu)$ is satisfiable. The opposite implication does not hold since satisfying assignment of $\mathcal{F}(\mu)$ may lead to a collision.

From the perspective of SMT, the propositional level does not understand geometric properties of agents so cannot know what simultaneous variable assignments are invalid. This information is only available at the level of theory $T = \text{MAPF}^{\mathcal{R}}$ through $DECIDE_{MAPF^{\mathcal{R}}}$. We also leave the bounding of the sum-of-costs at the level of $DECIDE_{MAPF^{\mathcal{R}}}$.

2.5 Lazy Encoding of Mutex Refinements and Sum-of-Costs Bounds

The SMT-based algorithm itself is divided into two procedures: SMT-CBS$^{\mathcal{R}}$ representing the main loop (Algorithm 2) and SMT-CBS-Fixed$^{\mathcal{R}}$ solving the input MAPF$^{\mathcal{R}}$ for a fixed maximum makespan μ and sum-of-costs ξ (Algorithm 3).

Procedures *encode-Basic* and *augment-Basic* in Algorithm 3 build formula $\mathcal{F}(\mu)$ according to given RDDs and the set of collected collision avoidance constraints. New collisions are resolved **lazily** by adding *mutexes* (disjunctive constraints). A collision is avoided in the same way as in CCBS; that is, one of the colliding agent waits. Collision eliminations are tried until a valid solution is obtained or until a failure for current μ and ξ which means to try bigger makespan and sum-of-costs.

For resolving a collision we need to: **(1)** eliminate simultaneous execution of colliding movements and **(2)** augment the formula to enable avoidance (waiting). Assume a collision between agents a_i traversing (u_i, v_i) during $[t_i^0, t_i^+)$ and a_j traversing (u_j, v_j) during $[t_j^0, t_j^+)$ which corresponds to variables $\mathcal{E}_{u_i,v_i}^{t_i^0,t_i^+}(a_i)$ and $\mathcal{E}_{u_j,v_j}^{t_j^0,t_j^+}(a_j)$. The collision can be eliminated by adding the following **mutex** (disjunction) to the formula: $\neg\mathcal{E}_{u_i,v_i}^{t_i^0,t_i^+}(a_i) \vee \neg\mathcal{E}_{u_j,v_j}^{t_j^0,t_j^+}(a_j)$. Satisfying assignments of the next $\mathcal{F}(\mu)$ can no longer lead to this collision. Next, the formula is augmented according to new RDDs that reflect the collision - decision variables and respective constraints are added.

After resolving all collisions we check whether the sum-of-costs bound is satisfied by plan π. This can be done easily by checking if $\mathcal{X}_u^t(a_i)$ variables across all agents together yield higher cost than ξ or not. If cost bound ξ is exceeded then corresponding *nogood* is recorded and added to $\mathcal{F}(\mu)$ and the algorithm continues by searching for a new satisfying assignment to $\mathcal{F}(\mu)$ now taking all recorded *nogoods* into account. The nogood says that $\mathcal{X}_u^t(a_i)$ variables that jointly exceed ξ cannot be simultaneously set to *TRUE*.

Formally, the nogood constraint can be represented as a set of variables $\{\mathcal{X}_{u_1}^{t_1}(a_1), \mathcal{X}_{u_2}^{t_2}(a_2), \ldots \mathcal{X}_{u_k}^{t_k}(a_k)\}$. We say the nogood to be *dominated* by another nogood $\{\mathcal{X}_{u_1}^{t'_1}(a_1), \mathcal{X}_{u_2}^{t'_2}(a_2), \ldots \mathcal{X}_{u_k}^{t'_k}(a_k)\}$ if and only if $t'_i \leq t_i$ for $i = 1,2,\ldots k$ and $\exists i \in \{1,2,\ldots,k\}$ such that $t'_i < t_i$. To make the nogood reasoning more efficient we do not need to store nogoods that are dominated by some previously discovered nogood. In such case however, the single nogood does not forbid one particular assignment but all assignments that could lead to dominated nogoods.

Algorithm 2: High-level of SMT-CBS$^{\mathcal{R}}$ for the sum-of-costs objective.

```
1 SMT-CBS^R (Σ^R = (G = (V,E), A, α_0, α_+, ρ))
2     constraints ← ∅
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)); ξ ← Σ_{i=1}^k μ(π(a_i))
5     while TRUE do
6         (π, constraints, μ_next, ξ_next) ← SMT-CBS-Fixed^R(Σ^R, constraints, μ, ξ)
7         if π ≠ UNSAT then
8             return π
9         μ ← μ_next; ξ ← ξ_next
```

The set of pairs of collision avoidance constraints is propagated across entire execution of the algorithm. Constraints originating from a single collision are grouped in pairs so that it is possible to introduce mutexes for colliding movements discovered in previous steps.

Algorithm 2 shows the main loop of SMT-CBS$^{\mathcal{R}}$. The algorithm checks if there is a solution for $\Sigma^{\mathcal{R}}$ of makespan μ and sum-of-costs ξ. It starts at the lower bound for μ and ξ obtained as the duration of the longest from shortest individual temporal plans ignoring other agents and the sum of these lengths respectively.

Then μ and ξ are iteratively increased in the main loop following the style of SAT-Plan [14]. The algorithm relies on the fact that the solvability of MAPF$^{\mathcal{R}}$ w.r.t. cumulative objective like the sum-of-costs or makespan behaves as a non decreasing function. Hence trying increasing makespan and sum-of-costs eventually leads to finding the optimum provided we do not skip any relevant value.

We need to ensure important property in the makespan/sum-of-costs increasing scheme: any solution of sum-of-costs ξ has the makespan of at most μ. The next sum-of-

costs to try is be obtained by taking the current sum-of-costs plus the smallest duration of the continuing movement (lines 17-27 of Algorithm 3).

The following proposition is a direct consequence of soundness of CCBS and soundness of the encoding (Proposition 1) and soundness of the makespan/sum-of-costs increasing scheme (proof omitted).

Proposition 2. *The SMT-CBS$^{\mathcal{R}}$ algorithm returns sum-of-costs optimal solution for any solvable MAPF$^{\mathcal{R}}$ instance $\Sigma^{\mathcal{R}}$.*

Algorithm 3: Low-level of SMT-CBS$^{\mathcal{R}}$

```
1  SMT-CBS-Fixed^R(Σ^R, cons, μ, ξ)
2      RDD ← build-RDDs(Σ^R, cons, μ)
3      F(μ) ← encode-Basic(RDD, Σ^R, cons, 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)
9              if collisions = ∅ then
10                 while TRUE do
11                     nogoods ← validate-Cost(π, ξ)
12                     if nogoods = ∅ then
13                         return (π, ∅, UNDEF, UNDEF)
14                     F(μ) ← F(μ) ∪ nogoods
15                     assignment ← consult-SAT-Solver(F(μ))
16                     if assignment = UNSAT then
17                         (μ_next, ξ_next) ← calc-Next-Bounds(μ, ξ, cons, RDD)
18                         return (UNSAT, cons, μ_next, ξ_next)
19                     π ← extract-Solution(assignment)
20             else
21                 for each (m_i × m_j) ∈ collisions where m_i = (a_i, (u_i, v_i), [t_i^0, t_i^+)) and
                     m_j = (a_j, (u_j, v_j), [t_j^0, t_j^+)) do
22                     F(μ) ← F(μ) ∧ (¬E_{u_i,v_i}^{t_i^0, t_i^+}(a_i) ∨ ¬E_{u_j,v_j}^{t_j^0, t_j^+}(a_j))
23                     ([τ_i^0, τ_i^+); [τ_j^0, τ_j^+)) ← resolve-Collision(m_i, m_j)
24                     cons ← cons ∪ {[(a_i, (u_i, v_i), [τ_i^0, τ_i^+)); (a_j, (u_j, v_j), [τ_j^0, τ_j^+))]}
25             RDD ← build-RDDs(Σ^R, cons, μ)
26             F(μ) ← augment-Basic(RDD, Σ^R, cons)
27     (μ_next, ξ_next) ← calc-Next-Bounds(μ, ξ, cons, RDD)
28     return (UNSAT, cons, μ_next, ξ_next)
```

3 Experimental Evaluation

We implemented SMT-CBS$^{\mathcal{R}}$ in C++ to evaluate its performance and compared it with CCBS [1].

SMT-CBS$^{\mathcal{R}}$ was implemented on top of Glucose 4 SAT solver [2] which ranks among the best SAT solvers according to recent SAT solver competitions [3]. The solver is consulted in the incremental mode if the formula is extended with new clauses. In case of CCBS, we used the existing C++ implementation [1].

3.1 Benchmarks and Setup

SMT-CBS$^{\mathcal{R}}$ and CCBS were tested on benchmarks from the `movinai.com` collection [29]. We tested algorithms on three categories of benchmarks:

(i) **small** empty grids (presented representative benchmark `empty-16-16`),
(ii) **medium** sized grids with regular obstacles (presented `maze-32-32-4`),
(iii) **large** game maps (presented `ost003d`, a map from Dragon Age game).

In each benchmark, we interconnected cells using the 2^K-neighborhood [23] for $K = 3,4,5$ - the same style of generating benchmarks as used in [1] ($K = 2$ corresponds to MAPF hence not omitted). Instances consisting of k agents were generated by taking first k agents from random scenario files accompanying each benchmark on `movinai.com`. Having 25 scenarios for each benchmarks this yields to 25 instances per number of agents.

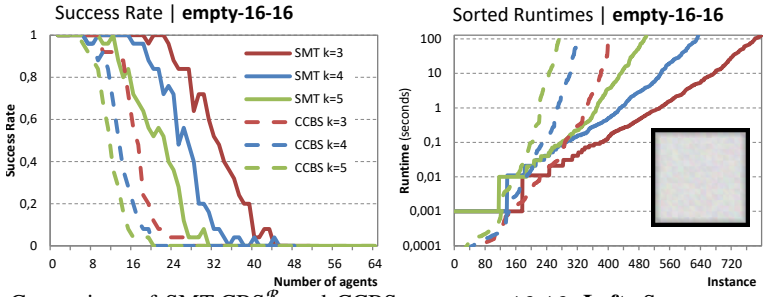

Fig. 3. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `empty-16-16`. **Left:** Success rate (the ratio of solved instances out of 25 under 120 seconds), the higher plot is better. **Right:** and sorted runtimes where the lower plot is better are shown.

Part of the results obtained in our experimentation is presented in this section[2]. For each presented benchmark we show *success rate* as a function of the number of agents.

[1] To enable reproducibility of presented results we will provide complete source code of our solvers on the author's website: http://users.fit.cvut.cz/surynpav/research/rcai2020.

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

That is, we calculate the ratio out of 25 instances per number of agents where the tested algorithm finished under the timeout of 120 seconds. In addition to this, we also show concrete runtimes sorted in the ascending order. Results for one selected representative benchmark from each category are shown in Figures 3, 4, and 5.

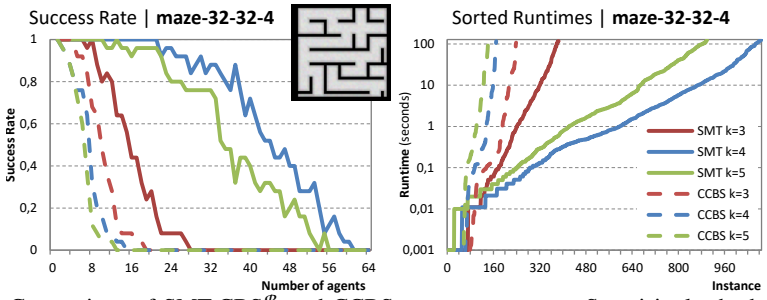

Fig. 4. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `maze-32-32-4`. Surprisingly the best performance with SMT-CBS$^{\mathcal{R}}$ highly connected neighborhoods ($K = 4, 5$ is easier than $K = 3$).

The observable trend is that the difficulty of the problem increases with increasing size of the $K-$neighborhood with notable exception of `maze-32-32-4` for $K = 4$ and $K = 5$ which turned out to be easier than $K = 3$ for SMT-CBS$^{\mathcal{R}}$.

Throughout all benchmarks SMT-CBS$^{\mathcal{R}}$ tends to outperform CCBS. The dominance of SMT-CBS$^{\mathcal{R}}$ is most visible in medium sized benchmarks. CCBS is, on the other hand, faster in instances containing few agents. The gap between SMT-CBS$^{\mathcal{R}}$ and CCBS is smallest in large maps where SMT-CBS$^{\mathcal{R}}$ struggles with relatively big overhead caused by the big size of the map (the encoding is proportionally big). Here SMT-CBS$^{\mathcal{R}}$ wins only in hard cases.

4 Discussion and Conclusion

We extended the approach based on *satisfiability modulo theories* (SMT) for solving MAPF$^{\mathcal{R}}$ from the makespan objective towards the sum-of-costs objective. 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 as it has been done in previous application of SMT in the standard MAPF. Bounding the sum-of-costs is done in similar lazy way through introducing nogoods incrementally. If it is detected that a conflict free solution exceeds given cost bound then decisions that jointly induce cost greater than given bound are forbidden via a nogood (that is, at least one of these decisions must not be taken).

We compared SMT-CBS$^{\mathcal{R}}$ with CCBS [1], currently the only alternative algorithm for MAPF$^{\mathcal{R}}$ that modifies the standard CBS algorithm, on a number of benchmarks. The outcome of our comparison is that SMT-CBS$^{\mathcal{R}}$ performs well against CCBS. The best results SMT-CBS$^{\mathcal{R}}$ are observable on medium sized benchmarks with regular obstacles. We attribute the better runtime results of SMT-CBS$^{\mathcal{R}}$ to more efficient handling of

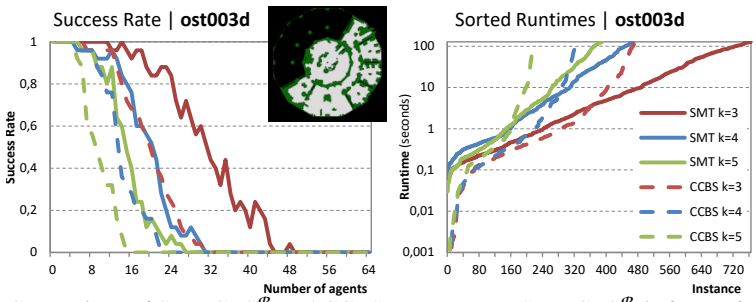

Fig. 5. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `ost003d`. SMT-CBS$^{\mathcal{R}}$ is fastest for $K = 3$ but for higher K the performance decreases significantly.

disjunctive conflicts in the underlying SAT solver through *propagation*, *clause learning*, and other mechanisms. On the other hand SMT-CBS$^{\mathcal{R}}$ is less efficient on large instances with few agents.

The important restriction which our concept rely on is that agents cannot move completely freely in the continuous space. We strongly assume that agents only move on the fixed embedding of finite graph $G = (V, E)$ into some continuous space where vertices are assigned points and edges are assigned curves on which the definition of smooth movement is possible. Hence for example using curves other than straight lines for interconnecting vertices does not change the high-level SMT-CBS$^{\mathcal{R}}$.

We plan to extend the RDD generation scheme to directional agents where we need to add the third dimension in addition to space (vertices) and time: *direction* (angle). The work on MAPF$^{\mathcal{R}}$ could be further developed into multi-robot motion planning in continuous configuration spaces [16].

Acknowledgements

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S. We would like to thank anonymous reviewers for their valuable comments.

References

1. Andreychuk, A., Yakovlev, K.S., Atzmon, D., Stern, R.: Multi-agent pathfinding with continuous time. In: Proceedings of IJCAI 2019. pp. 39–45 (2019)
2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. pp. 399–404 (2009)
3. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: Recent developments. In: AAAI 2017. pp. 5061–5063 (2017)
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS Press (2009)
5. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3), 273–303 (2012)

6. Botea, A., Surynek, P.: Multi-agent path finding on strongly biconnected digraphs. In: AAAI. pp. 2024–2030 (2015)
7. Cáp, M., Novák, P., Vokrínek, J., Pechoucek, M.: Multi-agent RRT: sampling-based cooperative pathfinding. In: Proceedings of AAMAS 2013. pp. 1263–1264 (2013)
8. Felner, A., Li, J., Boyarski, E., Ma, H., L.Cohen, Kumar, T.K.S., Koenig, S.: Adding heuristics to conflict-based search for multi-agent path finding. In: Proceedings of ICAPS 2018. pp. 83–87 (2018)
9. Hönig, W., Kumar, T.K.S., Cohen, L., Ma, H., Xu, H., Ayanian, N., Koenig, S.: Summary: Multi-agent path finding with kinematic constraints. In: Proceedings of IJCAI 2017. pp. 4869–4873 (2017)
10. Janovsky, P., Cáp, M., Vokrínek, J.: Finding coordinated paths for multiple holonomic agents in 2-d polygonal environment. In: Proceedings of AAMAS 2014. pp. 1117–1124 (2014)
11. Kautz, H.A.: Deconstructing planning as satisfiability. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, 2006. pp. 1524–1526. AAAI Press (2006)
12. Kautz, H.A., Selman, B.: Planning as satisfiability. In: Proceedings ECAI 1992. pp. 359–363 (1992)
13. Kautz, H.A., Selman, B.: Pushing the envelope: Planning, propositional logic and stochastic search. In: Proceedings of AAAI 1996. pp. 1194–1201 (1996)
14. Kautz, H.A., Selman, B.: Unifying sat-based and graph-based planning. In: Proceedings of IJCAI 1999. pp. 318–325 (1999)
15. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, 1984. pp. 241–250 (1984)
16. LaValle, S.M.: Planning algorithms. Cambridge University Press (2006)
17. Li, J., Surynek, P., Felner, A., Ma, H., Koenig, S.: Multi-agent path finding for large agents. In: Proceedings of AAAI 2019. AAAI Press (2019)
18. Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T.K.S., Uras, T., Xu, H., Tovey, C.A., Sharon, G.: Overview: Generalizations of multi-agent path finding to real-world scenarios. CoRR **abs/1702.05515** (2017), http://arxiv.org/abs/1702.05515
19. Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T.K.S., Koenig, S.: Multi-agent path finding with deadlines. In: Proceedings of IJCAI 2018. pp. 417–423 (2018)
20. Nieuwenhuis, R.: SAT modulo theories: Getting the best of SAT and global constraint filtering. In: Proceedings of CP 2010. pp. 1–2 (2010)
21. Phillips, M., Likhachev, M.: SIPP: safe interval path planning for dynamic environments. In: Proceedings of ICRA 2011. pp. 5628–5635 (2011)
22. Ratner, D., Warmuth, M.K.: Nxn puzzle and related relocation problem. J. Symb. Comput. **10**(2), 111–138 (1990)
23. Rivera, N., Hernández, C., Baier, J.A.: Grid pathfinding on the 2k neighborhoods. In: Proceedings of AAAI 2017. pp. 891–897 (2017)
24. Ryan, M.R.K.: Exploiting subgraph structure in multi-robot path planning. J. Artif. Intell. Res. (JAIR) **31**, 497–542 (2008)
25. Sharon, G., Stern, R., Felner, A., Sturtevant, N.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
26. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
27. Silver, D.: Cooperative pathfinding. In: AIIDE. pp. 117–122 (2005)
28. Stern, R.: Multi-agent path finding - an overview. In: Osipov, G.S., Panov, A.I., Yakovlev, K.S. (eds.) Artificial Intelligence - 5th RAAI Summer School. Lecture Notes in Computer Science, vol. 11866, pp. 96–115. Springer (2019)
29. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. Computational Intelligence and AI in Games **4**(2), 144–148 (2012)

30. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009. pp. 3613–3619 (2009)
31. Surynek, P.: Multi-agent path finding with continuous time and geometric agents viewed through satisfiability modulo theories (SMT). In: Surynek, P., Yeoh, W. (eds.) Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019. pp. 200–201. AAAI Press (2019)
32. Surynek, P.: Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In: Proceedings of IJCAI 2019. pp. 1177–1183 (2019)
33. Surynek, P.: On satisfisfiability modulo theories in continuous multi-agent path finding: Compilation-based and search-based approaches compared. In: Rocha, A.P., Steels, L., van den Herik, H.J. (eds.) Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 2, 2020. pp. 182–193. SCITEPRESS (2020)
34. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: ECAI. pp. 810–818 (2016)
35. Tinelli, C.: Foundations of satisfiability modulo theories. In: Proceedings is WoLLIC 2010. p. 58 (2010)
36. Walker, T.T., Sturtevant, N.R., Felner, A.: Extended increasing cost tree search for non-unit cost domains. In: Proceedings of IJCAI 2018. pp. 534–540 (2018)
37. Wang, K., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. JAIR **42**, 55–90 (2011)
38. Yu, J., LaValle, S.M.: Optimal multi-robot path planning on graphs: Structure and computational complexity. CoRR **abs/1507.03289** (2015)