Swarms of Mobile Agents: From Discrete to Continuous Movements in Multi-Agent Path Finding

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

***Abstract*—A variant of multi-agent path finding in continuous space and time with geometric agents MAPF$^{\mathcal{R}}$ is addressed in this paper. The task is to navigate agents that move smoothly between predefined positions to their individual goals so that they do not collide. We introduce a novel solving approach for obtaining makespan optimal solutions called SMT-CBS$^{\mathcal{R}}$ based on *satisfiability modulo theories* (SMT). The new algorithm combines collision resolution known from conflict-based search (CBS) with previous generation of incomplete SAT 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 the previous CCBS algorithm for MAPF$^{\mathcal{R}}$.**

I. Introduction

In *multi-agent path finding* (MAPF) [1]–[10] 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 a set $A = \{a_1, a_2, ..., a_k\}$ are placed in vertices with at most one agent per vertex. The initial configuration can be written as $\alpha_0 : A \to V$ and similarly the goal configuration as $\alpha_+ : A \to V$. The task of navigating agents can be then expressed formally as transforming α_0 into α_+ while movements are instantaneous and are possible across edges assuming no other agent is entering the same target vertex and no two agents traversing the same edge in opposite directions in the standard MAPF.

In order to reflect various aspects of real-life applications, variants of MAPF have been introduced such as those considering *kinematic constraints* [11], *large agents* [12], or *deadlines* [13] - see [14]–[16] for more variants. Particularly in this work we are dealing with an extension of MAPF introduced only recently [17], [18] that considers continuous time and space (MAPF$^{\mathcal{R}}$) where agents move smoothly along predefined curves interconnecting predefined positions placed arbitrarily in some continuous (metric) 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 [19]–[21] or teams of heterogeneous robots [22] where continuous reasoning is closer to the reality than the standard MAPF. Despite path planning in continuous space and time is subject to many studies in robotics at all conceptual levels ranging from planning to execution and control [23]–[25], the multi-agent perspective is often not fully adopted at the planning conceptual level.

We contribute by showing how to apply *satisfiability modulo theory* (SMT) reasoning [26]–[28] in makespan optimal MAPF$^{\mathcal{R}}$ solving. The SMT paradigm often 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 [3], [29] with previous algorithms for solving the standard MAPF using **lazy** constructions of **incomplete encodings** [30]–[32] and continuous reasoning. The merit of incomplete encoding is that not all constraints are introduced in the model making it representing superset of solutions. After deciding the incomplete model an external procedure decides if the solution is correct or not and the model is refined if needed. The advantage of this lazy generation of incomplete models is that a solution can be found well before a model is specified completely which can save lot of computational effort.

A. Related Work and Organization

Using reductions of planning problems to *propositional satisfiability* [33] has been coined in the SATPlan algorithm and its variants [34], [35]. 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 [18].

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* and leave it to the SAT solver as done in [36], [37]. We construct the formula *lazily* by adding collision elimination refinements following [30] 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 $TRUE$, then a is 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 as in the standard MAPF. Hence we need to select from a potentially uncountable space those variables that are sufficient for finding the makespan optimal solution.

The organization is as follows: we first introduce MAPF$^{\mathcal{R}}$. Then we recall CCBS, a variant of CBS for MAPF$^{\mathcal{R}}$. Details of the novel SMT-based solving algorithm SMT-CBS$^{\mathcal{R}}$ follow. Finally, a comparison SMT-CBS$^{\mathcal{R}}$ with CCBS is shown.

B. MAPF in Continuous Time and Space

We use the definition of MAPF in continuous time and space denoted MAPF$^{\mathcal{R}}$ from [17] and [18]. 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 (MAPF$^{\mathcal{R}}$) is a 5-tuple $\Sigma^{\mathcal{R}} = (G = (V, E), A, \alpha_0, \alpha_+, \rho)$ where G, A, α_0, α_+ are from the standard MAPF and ρ determines continuous extensions:

- $\rho.x(v), \rho.y(v)$ 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

We assume that agents have 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 [12] and can move along various fixed curves.

For simplicity we elaborate our implementations for the above simple 2D continuous extension with circular agents.

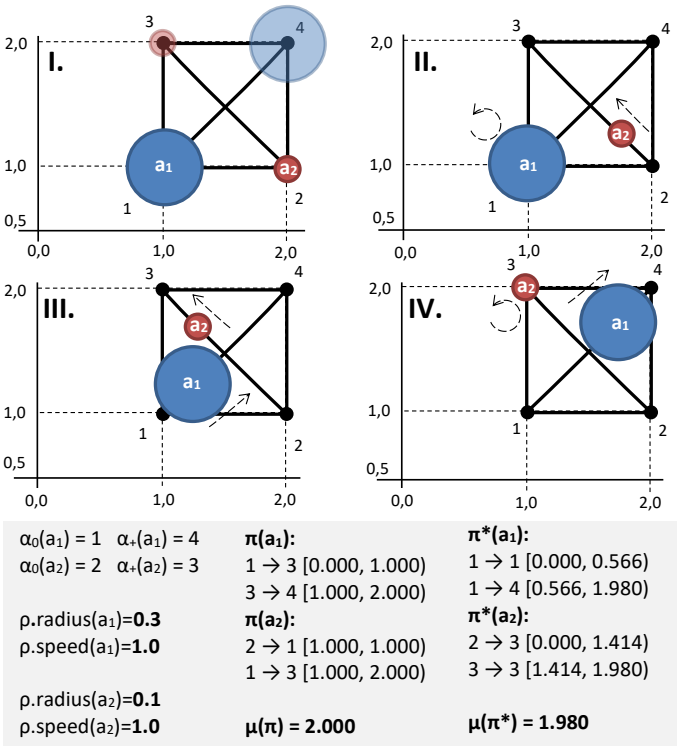

Fig. 1. An example of MAPF$^{\mathcal{R}}$ instance with two agents. A feasible makespan sub-optimal solution π (makespan $\mu(\pi) = 2.0$) and makespan optimal solution $\pi*$ (makespan $\mu(\pi*) = 1.980$) are shown.

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 $\pi(a) = [((\alpha_0(a), \alpha_1(a)), [t_0(a), t_1(a))); ((\alpha_1(a), \alpha_2(a)), [t_1(a), t_2(a))); ...; ((\alpha_{m(a)-1}, \alpha_{m(a)}(a)), [t_{m(a)-1}, t_{m(a)}))]$ where $m(a)$ is the length of individual temporal plan and each pair $(\alpha_i(a), \alpha_{i+1}(a)), [t_i(a), t_{i+1}(a)))$ 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 $\max_{i=1}^{k}(\mu(\pi(a_i)))$. We focus on finding makespan optimal solutions. An example of MAPF$^{\mathcal{R}}$ and makespan optimal solution is shown in Figure 1.

Through straightforward reduction of MAPF to MAPF$^{\mathcal{R}}$ it can be observed that finding a makespan optimal solution with continuous time is an NP-hard problem [38], [39].

II. Solving MAPF with Continuous Time

Let us recall CCBS [18], a variant of CBS [3] modified for MAPF$^{\mathcal{R}}$. The idea of CBS algorithms is to resolve conflicts

Algorithm 1: CCBS algorithm for solving MAPF$^{\mathcal{R}}$.

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 $\alpha_+(a_i) \mid i = 1, 2, ..., k\}$
  $R.\mu \leftarrow \max_{i=1}^{k} \mu(N.\pi(a_i))$
  OPEN $\leftarrow \emptyset$
  insert R into OPEN
  while OPEN $\neq \emptyset$ **do**
    $N \leftarrow \min_{\mu}$(OPEN)
    remove-Min$_{\mu}$(OPEN)
    $collisions \leftarrow$ validate-Plans($N.\pi$)
    if $collisions = \emptyset$ **then**
      return $N.\pi$
    let $(m_i \times m_j) \in collisions$ 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, $N'.\pi$, $N'.cons$)
      $N'.\mu \leftarrow \max_{i=1}^{k} \mu(N'.\pi(a_i))$
      insert N' into OPEN

lazily. CBS algorithms are usually developed for the *sum-of-costs* [4] objective but using other cumulative costs like makespan is possible too.

A. Conflict-based Search

CCBS for finding the makespan 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 makespan 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 $(a_i, (u, v), [t_0, t_+))$ forbidding agent a_i to start smooth traversal of edge $\{u, v\}$ (line) at any time between $[t_0, t_+)$, a solution $N.\pi$ - a set of k individual temporal plans, and the makespan $N.\mu$ 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 $\{(u, v) \in E \mid (a_i, (u, v), [t_0, t_+)) \in N.cons \wedge t \in [t_0, t_+)\}$. Various intelligent single source shortest path algorithms such as SIPP [40] can be used here.

CCBS stores nodes of CT into priority queue OPEN sorted according to the ascending makespan. At each step CBS takes node N with the lowest makespan from OPEN and checks if $N.\pi$ 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.

B. A Satisfiability Modulo Theory Approach

We will use for the specific case of CCBS the idea introduced in [30] that rephrases the algorithm as problem solving in *satisfiability modulo theories* (SMT) [26], [27]. 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).

C. RDD: Real Decision Diagram

The important question when using the logic approach is what will be the decision variables. In the standard MAPF, time expansion of G for every time step has been done resulting in a multi-value decision diagram (MDD) [37] representing possible positions of agents at any time step. Since MAPF$^{\mathcal{R}}$ is inherently continuous we cannot afford to consider every time moment but we need to restrict on important moments only.

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 $((u, t_u); (v, t'_u))$. Pair

Algorithm 2: Building of RDD for MAPF$^{\mathcal{R}}$

```
1  build-RDDs(Σ^R, cons, μ_max)
2      for i = 1, 2, ..., k do
3          X^i ← ∅, E^i ← ∅, OPEN ← ∅
4          insert (α_0(a_i), 0) into OPEN
5          X^i ← X^i ∪ {(α_0(a_i), 0)}
6          while OPEN ≠ ∅ do
7              (u, t) ← min_t(OPEN)
8              remove-Min_t(OPEN)
9              if t ≤ μ_max then
10                 for each v | {u, v} ∈ E do
11                     Δt ← dist(u, v)/v_{a_i}
12                     insert (v, t + Δt) into OPEN
13                     X^i ← X^i ∪ {(v, t + Δt)}
14                     E^i ← E^i ∪ {[(u, t); (v, t + Δt)]}
15                     for each (a_i, (u, v), [τ^0, τ^+)) ∈ cons do
16                         if t ≥ τ^0 and t < τ^+ then
17                             insert (u, τ_+) into OPEN
18                             X^i ← X^i ∪ {(u, τ^+)}
19                             E^i ← E^i ∪ {[(u, t); (u, τ^+)]}
20     return [(X^1, E^1), (X^2, E^2), ..., (X^k, E^k)]
```

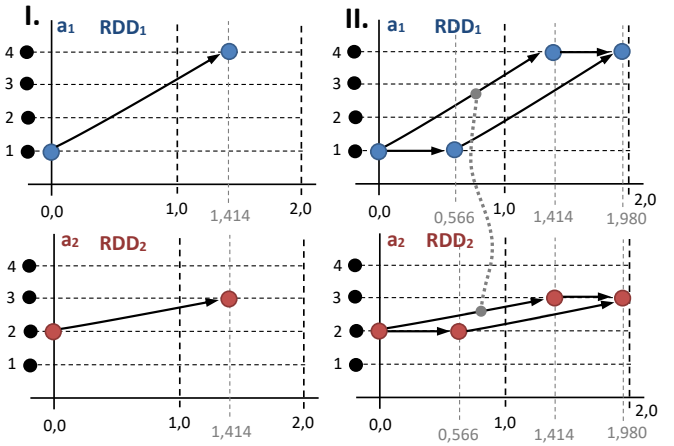

Fig. 2. Real decision diagrams (RDDs) for agents a_1 and a_2 from MAPF$^{\mathcal{R}}$ from Figure 1. Decisions corresponding to shortest paths for agents a_1 and a_2 moving diagonally towards their goals are shown: $a_1 : 1 \rightarrow 4$, $a_2 : 2 \rightarrow 3$ (left). This however results in a collision 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.

$(\alpha_0(a_i), 0) \in X_i$ 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 $((u, \tau^+); (u, \tau^+))$ are added.

The process of building RDDs is formalized in Algorithm 2. It performs breadth-first search (BFS). For each possible edge traversal the algorithm generates a successor node and corresponding edge (lines 12-15) but also considers all possible wait action w.r.t. interfering collision avoidance constraints (lines 17-20). As a result each constraint is treated as both present and absent. In other words, RDD_i represents union of all paths for agent a_i examined in all branches of the corresponding CT. The stop condition is specified by the maximum makespan μ_{max} beyond which no more actions are performed. An example of RDDs is shown in Figure 2.

D. 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_+.

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

We apriori 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 $DECIDE_{MAPF^{\mathcal{R}}}$.

E. Lazy Encoding via Mutex Refinements

The SMT-based algorithm itself is divided into two procedures: SMT-CBS$^{\mathcal{R}}$ representing the main loop and SMT-CBS-Fixed$^{\mathcal{R}}$ solving the input MAPF$^{\mathcal{R}}$ for a fixed maximum makespan μ. The major difference from the standard CBS is that there is **no branching** at the high-level.

Procedures *encode-Basic* and *augment-Basic* in Algorithm 4 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 (line 11) or until a failure for current μ (line 20) which means to try bigger makespan.

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: $\neg\mathcal{E}_{u_i,v_i}^{t_i^0,t_i^+}(a_i) \vee \neg\mathcal{E}_{u_j,v_j}^{t_j^0,t_j^+}(a_j)$ (line 13 in Algorithm 4). 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.

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

```
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))
5      while TRUE do
6          (π, constraints, μ_next) ←
7            SMT-CBS-Fixed^R(Σ^R, constraints, μ)
8          if π ≠ UNSAT then
9              return π
10         μ ← μ_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 3 shows the main loop of SMT-CBS$^{\mathcal{R}}$. The algorithm checks if there is a solution for $\Sigma^{\mathcal{R}}$ of makespan μ. It starts at the lower bound for μ obtained as the duration of the longest from shortest individual temporal plans ignoring other agents (lines 3-4). Then μ is iteratively increased in the main loop (lines 5-9) following the style of SATPlan [35]. The algorithm relies on the fact that the solvability of MAPF$^{\mathcal{R}}$ w.r.t. cumulative objective like the makespan behaves as a non decreasing function. Hence trying increasing makespans eventually leads to finding the optimum provided we do not skip any relevant makespan.

Algorithm 4: Low-level of SMT-CBS$^{\mathcal{R}}$

```
1  SMT-CBS-Fixed^R(Σ^R, cons, μ)
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                 return (π, ∅, UNDEF)
11             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
12                 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))
13                 ([τ_i^0, τ_i^+); [τ_j^0, τ_j^+)) ←
                     resolve-Collision(m_i, m_j)
14                 cons ← cons ∪ {[(a_i, (u_i, v_i), [τ_i^0, τ_i^+));
                     (a_j, (u_j, v_j), [τ_j^0, τ_j^+))]}
15             RDD ← build-RDDs(Σ^R, cons, μ)
16             F(μ) ← augment-Basic(RDD, Σ^R, cons)
17     μ_next ← min{t | (u, t) ∈ X_i ∧ t > μ
18       where RDD_i = (X_i, E_i) for i = 1, 2, ..., k)}
19     return (UNSAT, cons, μ_next)
```

The next makespan to try will then be obtained by taking the current makespan plus the smallest duration of the continuing movement (lines 17-18 of Algorithm 4). The following proposition is a direct consequence of soundness of CCBS and soundness of the encoding (Proposition 1).

Propositon 2: The SMT-CBS$^{\mathcal{R}}$ algorithm returns makespan optimal solution for any solvable MAPF$^{\mathcal{R}}$ instance $\Sigma^{\mathcal{R}}$.

III. Experimental Evaluation

We implemented SMT-CBS$^{\mathcal{R}}$ in C++ to evaluate its performance and compared it with a version of CCBS adapted for the makespan objective [1]

SMT-CBS$^{\mathcal{R}}$ was implemented on top of Glucose 4 SAT solver [41] which ranks among the best SAT solvers according to recent SAT solver competitions [42]. Whenever possible the SAT solver was consulted in the incremental mode.

The actual implementation builds RDDs in a more sophisticated way than presented pedagogically in Algorithm 2. The implementation prunes out decisions from that the goal vertex cannot be reached under given makespan bound μ_{max}: 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.

In case of CCBS, we used the existing C++ implementation for the sum-of-costs objective [18] and modified it for makespan while we tried to preserve its heuristics from the sum-of-costs case.

[1]To enable reproducibility of presented results we will provide complete source code of our solvers and experimental data on author's web: `http://users.fit.cvut.cz/surynpav/research` and git repository: `https://github.com/surynek`.

A. Benchmarks and Setup

SMT-CBS$^{\mathcal{R}}$ and CCBS were tested on benchmarks from the `movinai.com` collection [43]. 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`).

In each benchmark, we interconnected cells using the 2^K-neighborhood [44] for $K = 3, 4, 5$ - the same style of generating benchmarks as used in [18]. Each node of the grid is connected with its 2^K nearest neighbor node. The standard 4-connected grid corresponds to $K = 2$ (omitted here), $K = 2$ is an 8-connected grid with diagonal moves allowed. 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.

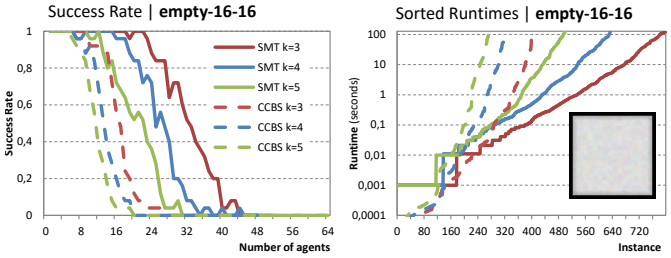

Fig. 3. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `empty-16-16`.

Since it is hard to verify validity of continuous solutions by hand we implemented automated verification procedure. The verification procedure calculates positions of all agents as a function of time. The time is incremented by small steps throughout the verification and overlaps between agents is checked. If any overlap should happen then the verification fails indicating that the planner produces incorrect solutions.

B. Runtime Results

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. 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.

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$^{\mathcal{R}}$.

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

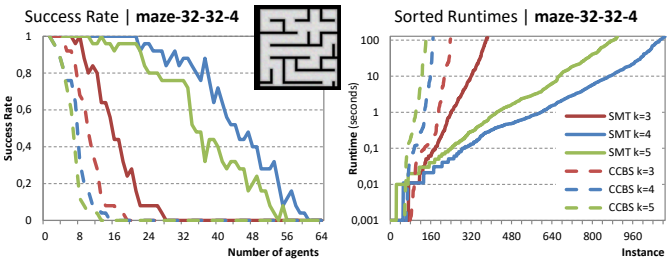

Fig. 4. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `maze-32-32-4`.

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.

IV. Discussion and Conclusion

We suggested a novel algorithm for the makespan optimal solving of the multi-agent path finding problem in continuous time and space called SMT-CBS$^{\mathcal{R}}$ 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 as it has been done in previous application of SMT in the standard MAPF.

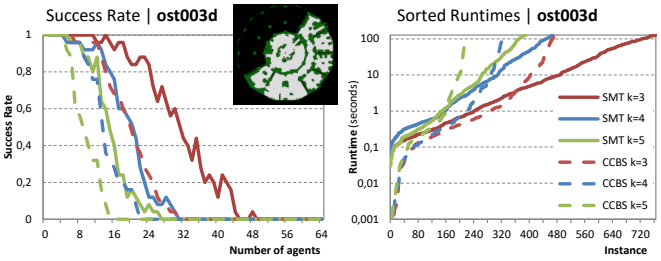

Fig. 5. Comparison of SMT-CBS$^{\mathcal{R}}$ and CCBS on `ost003d`.

The major obstacle in using SMT and propositional reasoning not faced previously with the standard MAPF 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 collisions.

We compared SMT-CBS$^{\mathcal{R}}$ with CCBS [18], 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 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.

For future work, we assume extending the concept of SMT-based approach for MAPF$^{\mathcal{R}}$ with other cumulative cost functions other than the makespan such as the *sum-of-costs* [4]. We also 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 [45].

V. Acknowledgement

This work has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S.

References

[1] D. Kornhauser, G. L. Miller, and P. G. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *FOCS, 1984*, 1984, pp. 241–250.

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

[3] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, 2015.

[4] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, 2013.

[5] D. Silver, "Cooperative pathfinding," in *AIIDE*, 2005, pp. 117–122.

[6] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *ICRA 2009*, 2009, pp. 3613–3619.

[7] K. Wang and A. Botea, "MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees," *JAIR*, vol. 42, pp. 55–90, 2011.

[8] P. Surynek, "Solving abstract cooperative path-finding in densely populated environments," *Comput. Intell.*, vol. 30, no. 2, pp. 402–450, 2014.

[9] B. de Wilde, A. ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *J. Artif. Intell. Res.*, vol. 51, pp. 443–492, 2014.

[10] M. Kulich, T. Novák, and L. Preucil, "Push, stop, and replan: An application of pebble motion on graphs to planning in automated warehouses," in *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*. IEEE, 2019, pp. 4456–4463.

[11] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Summary: Multi-agent path finding with kinematic constraints," in *Proceedings of IJCAI 2017*, 2017, pp. 4869–4873.

[12] J. Li, P. Surynek, A. Felner, H. Ma, and S. Koenig, "Multi-agent path finding for large agents," in *Proceedings of AAAI 2019*. AAAI Press, 2019.

[13] H. Ma, G. Wagner, A. Felner, J. Li, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding with deadlines," in *Proceedings of IJCAI 2018*, 2018, pp. 417–423.

[14] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. K. S. Kumar, T. Uras, H. Xu, C. A. Tovey, and G. Sharon, "Overview: Generalizations of multi-agent path finding to real-world scenarios," *CoRR*, vol. abs/1702.05515, 2017.

[15] H. Ma and S. Koenig, "AI buzzwords explained: multi-agent path finding (MAPF)," *AI Matters*, vol. 3, no. 3, pp. 15–19, 2017.

[16] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017*. AAAI Press, 2017, pp. 29–37.

[17] T. T. Walker, N. R. Sturtevant, and A. Felner, "Extended increasing cost tree search for non-unit cost domains," in *Proceedings of IJCAI 2018*, 2018, pp. 534–540.

[18] A. Andreychuk, K. S. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proceedings of IJCAI 2019*, 2019, pp. 39–45.

[19] P. Janovsky, M. Cáp, and J. Vokrínek, "Finding coordinated paths for multiple holonomic agents in 2-d polygonal environment," in *Proceedings of AAMAS 2014*, 2014, pp. 1117–1124.

[20] M. Cáp, P. Novák, J. Vokrínek, and M. Pechoucek, "Multi-agent RRT: sampling-based cooperative pathfinding," in *Proceedings of AAMAS 2013*, 2013, pp. 1263–1264.

[21] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans. Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[22] M. Debord, W. Hönig, and N. Ayanian, "Trajectory planning for heterogeneous robot teams," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018*. IEEE, 2018, pp. 7924–7931.

[23] J. Wang, "Speed-assigned position tracking control of SRM with adaptive backstepping control," *IEEE CAA J. Autom. Sinica*, vol. 5, no. 6, pp. 1128–1135, 2018.

[24] A. Mustafa, N. K. Dhar, and N. K. Verma, "Event-triggered sliding mode control for trajectory tracking of nonlinear systems," *IEEE CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 307–314, 2020.

[25] K. Majd, M. Razeghi-Jahromi, and A. Homaifar, "A stable analytical solution method for car-like robot trajectory tracking and optimization," *IEEE CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 39–47, 2020.

[26] C. Tinelli, "Foundations of satisfiability modulo theories," in *Proceedings is WoLLIC 2010*, 2010, p. 58.

[27] M. Bofill, M. Palahí, J. Suy, and M. Villaret, "Solving constraint satisfaction problems with SAT modulo theories," *Constraints*, vol. 17, no. 3, pp. 273–303, 2012.

[28] R. Nieuwenhuis, "SAT modulo theories: Getting the best of SAT and global constraint filtering," in *Proceedings of CP 2010*, 2010, pp. 1–2.

[29] A. Felner, J. Li, E. Boyarski, H. Ma, L.Cohen, T. K. S. Kumar, and S. Koenig, "Adding heuristics to conflict-based search for multi-agent path finding," in *Proceedings of ICAPS 2018*, 2018, pp. 83–87.

[30] P. Surynek, "Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories," in *Proceedings of IJCAI 2019*, 2019, pp. 1177–1183.

[31] ——, "Conflict handling framework in generalized multi-agent path finding: Advantages and shortcomings of satisfiability modulo approach," in *Proceedings of the 11th Intl. Conf. on Agents and Artificial Intelligence, ICAART 2019, Volume 2*. SciTePress, 2019, pp. 192–203.

[32] ——, "Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach," in *2019 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS 2019*. IEEE, 2019, pp. 3282–3287.

[33] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*. IOS Press, 2009.

[34] H. A. Kautz and B. Selman, "Pushing the envelope: Planning, propositional logic and stochastic search," in *Proceedings of AAAI 1996*, 1996, pp. 1194–1201.

[35] ——, "Unifying sat-based and graph-based planning," in *Proceedings of IJCAI 1999*, 1999, pp. 318–325.

[36] P. Surynek, "Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs," in *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014*. IEEE Computer Society, 2014, pp. 875–882.

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

[38] D. Ratner and M. K. Warmuth, "Nxn puzzle and related relocation problem," *J. Symb. Comput.*, vol. 10, no. 2, pp. 111–138, 1990.

[39] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," *CoRR*, vol. abs/1507.03289, 2015.

[40] M. Phillips and M. Likhachev, "SIPP: safe interval path planning for dynamic environments," in *Proceedings of ICRA 2011*, 2011, pp. 5628–5635.

[41] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI*, 2009, pp. 399–404.

[42] T. Balyo, M. J. H. Heule, and M. Järvisalo, "SAT competition 2016: Recent developments," in *AAAI 2017*, 2017, pp. 5061–5063.

[43] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[44] N. Rivera, C. Hernández, and J. A. Baier, "Grid pathfinding on the $2k$ neighborhoods," in *Proceedings of AAAI 2017*, 2017, pp. 891–897.

[45] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.