Bounded Suboptimal Token Swapping

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

***Abstract*—Token swapping (TSWAP) represents a challenging problem underlying in many practical applications ranging from item relocation to quantum program compilation. In TSWAP, we are given an undirected graph with colored vertices. A colored token is placed in each vertex. A pair of tokens can be swapped between a pair of adjacent vertices. The goal is to perform a sequence of swaps so that token and vertex colors agree across the graph. The total number of swaps is usually required to be small. We study bounded sub-optimal algorithms for solving the TSWAP problem. We introduce a SAT-based algorithm based on lazy compilation of the problem to a Boolean formula and an alternative search-based algorithm using conflict based search. We analyze both algorithms experimentally on a number of benchmarks.**

***Index Terms*—token swapping, conflict-based search, SAT, bounded sub-optimal solutions, satisfiability modulo theory**

I. INTRODUCTION

The *token swapping problem* (TSWAP) (also known as *sorting on graphs*) [1]–[3] is a problem of swapping distinguishable tokens across edges of an undirected graph so that eventually desired configuration of tokens in vertices of the graph is obtained. TSWAP is an important challenging problem that arises in many applications ranging from logistics, robotics, warehouse operations [4], ship collision avoidance [5], or quantum program compilation [6].

TSWAP represents a generalization of sorting problems, a fundamental task in computer science [7]. While in the classical sorting problem we need to obtain linearly ordered sequence of elements by swapping elements at any pair of positions, that is, permitted swaps are defined by a complete graph over the set of positions. In the TSWAP problem, we are allowed to swap elements at selected pairs of positions only defined by the edges of the input undirected graph (generally sparser than a clique). Usually the minimum number of swaps is desirable both in the classical sorting and in TSWAP.

Using a modified notation from [8] the TSWAP problem is given by an undirected graph $G = (V, E)$ with vertex set V and edge set E. Each vertex in G is assigned a color in $C = \{c_1, c_2, ..., c_k\}$ via $\tau_+ : V \rightarrow C$. A token of a color in C is placed in each vertex. The task is to transform an input token placement into the one such that colors of tokens and respective vertices of their placement agree. Desirable token placement can be obtained by swapping tokens on adjacent vertices in G. See Figure 1 for an example instance of TSWAP.

Research presented in this paper is supported by GAČR, the Czech Science Foundation, grant registration number 19-17966S.

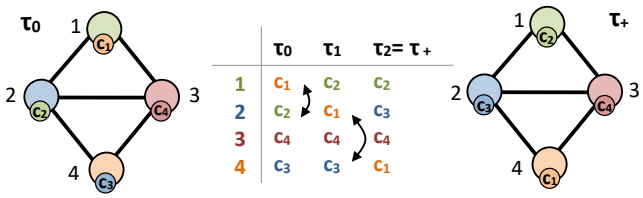

Fig. 1. A TSWAP instance. A solution consisting of two swaps is shown.

The TSWAP problem has been introduced only recently. So far theoretical results concerning computational complexity [3] and approximations [9], [10] have appeared. To our best knowledge there is no study dealing with practical solving of TSWAP optimally except [11].

In this paper, we further develop results from [11] where the similarity between TSWAP and a well studied problem of *multi-agent path finding* (MAPF) [12], [13] has been reported for the first time. It has been shown how to migrate two existing algorithms - *Conflict-Based Search* (CBS) [14], a search based algorithm, and MDD-SAT [15], a Boolean satisfiability [16] (SAT)-based algorithm - from MAPF to TSWAP solving. Later in [17] SMT-CBS, an algorithm using *satisfiability modulo theories* (SMT) to build Boolean models of MAPF lazily [18], has been migraged to the TSWAP problem too. These algorithms are all optimal in common cumulative objectives like *makespan* or the *sum-of-costs*. Finding optimal solutions however often requires considerable runtime. Hence we suggest in this work bounded-suboptimal algorithms for the TSWAP problem that enable the user to trade-off quality of solutions for shorter runtime.

Two novel algoritms are introduced in this work: **(i)** eCBS(TSWAP) representing a migrated version of existing bounded-suboptimal eCBS [19] from the MAPF domain to the TSWAP domain and **(ii)** eSMT-CBS(TSWAP) which is a new bounded sub-optimal algorithm for the TSWAP problem derived from optimal SMT-CBS based on satisfiability modulo theories [20], [21].

The paper is organized as follows: We introduce TSWAP problem formally and describe its theoretical properties first. Then we develop eCBS(TSWAP) as a modification of eCBS and derive eSMT-CBS(TSWAP) from existing SMT-CBS. Theoretical analysis that eSMT-CBS(TSWAP) generates bounded-suboptimal solutions is given afterwards. Finally an experimental evaluation of eCBS(TSWAP) and eSMT-CBS(TSWAP)

on a diverse set of TSWAP benchmarks is presented.

II. BACKGROUND

The formal definition of the *token swapping* (TSWAP) problem assumes an input undirected graph $G = (V, E)$ where a colored token resides in each vertex of G. Assignment of colors to tokens placed in vertices of G can be expressed using: $\tau : V \rightarrow C$ where C is a finite set of colors. That is, $\tau(v)$ for $v \in V$ is a color of a token placed in v. We note that the placement of colored tokens can be fully expressed by $\tau : V \rightarrow C$ hence we also call τ a *token placement*. In the rest of the paper we do not distinguish between the coloring of tokens and their placement.

The fundamental action in the TSWAP problem is *swapping* of a pair of tokens across an edge. Multiple token pairs can be swapperd in **parallel** as formally described in the following definition [8], [10]:

Definition 1: **Adjacency.** Token placements τ and τ' are said to be adjacent if there exist a subset of non-adjacent edges $F \subseteq E$ such that $\tau(v) = \tau'(u)$ and $\tau(u) = \tau'(v)$ for each $\{u, v\} \in F$ and for all other vertices $w \in V \setminus \bigcup_{\{u,v\} \in F} \{u, v\}$ it holds that $\tau(w) = \tau'(w)$. [1]

It is always assumed that the knowledge of adjacent token placements τ and τ' comes together with information what swapping actions transform τ to τ'.

Starting placement of tokens is denoted as τ_0; the goal token placement corresponds to τ_+. The task is to make swaps to obtain τ_+ from τ_0 as formalized in the following definition:

Definition 2: **Token Swapping.** A *token swapping* (TSWAP) problem is a 4-tuple $\Psi = (G = (V, E), C, \tau_0, \tau_+)$ where G is an undirected graph, C is a finite set of colors, and τ_0 is the starting placement of tokens and τ_+ is a goal placement of tokens. The task is to find a sequence of token placements $\psi = [\tau_0, \tau_1, ..., \tau_m]$ and respective swapping actions such that $\tau_m = \tau_+$ and τ_i and τ_{i+1} are adjacent for all $i = 0, 1, ..., m-1$.

The length of sequence of token placements m is called a *makespan* and denoted $\mu(\psi)$. The cost of solution is defined as the total number of swaps being used across entire ψ and is denoted $\sigma(\psi)$.

A. Basic Properties of Token Swapping

It has been shown that for any initial and goal placement of tokens τ_0 and τ_+ respectively there is a swapping sequence transforming τ_0 and τ_+ containing $\mathcal{O}(|V|^2)$ swaps [22]. The proof is based on the idea of swapping tokens over a spanning tree of G. Let us note that the above bound is tight as there are instances consuming $\Omega(|V|^2)$ swaps.

It is also known that finding swapping sequence that has as few swaps as possible, that is finding ψ with the minimum $\sigma(\psi)$, is an NP-hard problem [3], [22]. Analogous result can be shown for minimizing the makespan $\mu(\psi)$.

[1]The presented version of adjacency is sometimes called *parallel* while the term adjacency without an adjective is understood as a case with $|F| = 1$.

B. Related Concepts: the (N^2-1)-puzzle and MAPF

There are closely related problems similar to TSWAP. The first well studied related concept is a (N^2-1)-puzzle [23] and more generaly *graph pebbling* [24]. In the (N^2-1)-puzzle we are given a square board of the size N times N with $N^2 - 1$ labeled square tiles - usually numeric labels 1, ... $N^2 - 1$ are used. One position on the board is left blank in order to make possible moving of a tile towards the blank position. The task is to re-arrange tiles using allowed moves so that tiles are ordered from 1 to $N^2 - 1$ in the final configuration.

Solving the (N^2-1)-puzzle optimally, that is using the minimum number of moves, is NP-hard as shown by Ratner and Warmuth in [23]. This rather negative result is however complemented by multiple techniques that attempt to solve the puzzle, including on-line algorithms [25], [26], solution improvement methods [27], or even machine learning techniques [28].

The (N^2-1)-puzzle can be further generalized to the concept of graph pebbling where instead of having a square board we are given an undirected graph with pebbles placed in vertices with at most one pebble per vertex. The task is to reach a given goal configuration of pebbles in vertices of the graph.

Another important related concept to the token swapping problem is represented by *multi-agent path finding* (MAPF) [14], [29]–[31]. In MAPF, the task is to navigate distinguishable agents in an undirected graph towards agents' individual goal vertices while collisions in vertices and edges must be avoided. MAPF is almost the same as graph pebbling except the constraint that in MAPF we do not always need the target vertex of a move to be empty.

Formally, a configuration of agents from a set A such that $|A| < |V|$ is assignment of agents to vertices $\alpha : A \rightarrow V$ so that no two agents occupy the same vertex. We are given initial configuration of agents α_0 and goal configuration α_+. At each time step an agent can either *move* to an adjacent location or *wait* in its current location. The task is to find a sequence of move/wait actions for each agent a_i, moving it from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *conflict*, i.e., do not occupy the same location at the same time.

Definition 3: **MAPF.** A *multi-agent path finding problem* is a 4-tuple $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$. Solving the MAPF instance is to search for a sequence of configurations $\pi = [\alpha_0, \alpha_1, ..., \alpha_m]$ such that α_{i+1} results using valid movements from α_i for $i = 1, 2, ..., \mu - 1$, and $\alpha_m = \alpha_+$.

A valid movement is defined through a valid transformation of configuration α to α' which happens if and only if the following conditions hold: (i) $\alpha(a) = \alpha'(a)$ or $\{\alpha(a), \alpha'(a)\} \in E$ for all $a \in A$ (agents wait or move along edges); (ii)for all $a \in A$ it holds that $\alpha'(a) = \alpha(a')$ for some $a' \in A \Rightarrow \alpha'(a') \neq \alpha(a)$ (agent do make swaps across edges); and (iii) for all $a, a' \in A$ it holds that if $a \neq a' \Rightarrow \alpha'(a) \neq \alpha'(a')$ (no two agents enter the same target vertex).

In contrast to the upper bound complexity result for the TSWAP problem, it can be shown that any solvable MAPF

instance can be solved using $\mathcal{O}(|V|^3)$ moves [24], [32]. Moreover, this is a tight bound again as there as instances that need $\Omega(|V|^3)$ moves.

C. Practical Solvers for TSWAP

Currently limited number of solvers for the TSWAP problem exist that address the problem from the practical point of view when the number of swaps matters. There are two derivatives of optimal solvers originally designed for the MAPF problem and later adapted for TSWAP: CBS(TSWAP) and MDD-SAT(TSWAP) [11].

The former is a derivative of the Conflict-based search (CBS) algorithm [14] a major representative of optimal **search-based** solvers for MAPF. Search-based solvers consider the problem as a graph search problem. Some of these algorithms are variants of the A* algorithm that search in a global *search space* – all different ways to place agents/tokens into vertices are explored while keeping the constraint that one agent per vertex is always valid [29], [33].

Others algorithms such as ICTS [31] and CBS [14] search different search spaces and employ novel (non-A*) search tree. These solvers can be adapted to the bounded sub-optimal variant by relaxing the bound enforcing mechanism across the search process: eCBS [19] and bounded sub-optimal version of ICTS [34].

The MDD-SAT(TSWAP) is a derivative of so-called **reduction-based** solvers for MAPF that reduce the problem to known problems such as CSP [35], SAT [30], [36], [37], Inductive Logic Programming [38] and Answer Set Programming [39]. Namely the MDD-SAT(TSWAP) algorithm derived from [15] takes SAT as the target formalism.

Bounded sub-optimal versions of various reduction-based solvers exists. The related SAT-based bounded-suboptimal solver for MAPF called eMDD-SAT has been introduced in [40]. The most advanced reduction-based solver that builds the instance in the target formalism lazily via concepts from *satisfiability modulo theories* (SMT) [20], [21], [41], called SMT-CBS [18], has not yet been migrated to TSWAP nor any sub-optimal version of it has been introduced yet.

III. BOUNDED SUBOPTIMAL CBS FOR TSWAP

The first algorithm that we adapt for the TSWAP problem is bounded suboptimal CBS originally designed for MAPF, refered to as ECBS [19]. As the algorithm is bounded suboptimal by its original design we only need to consider how to integrate differences of MAPF and TSWAP into its framework. The modification takes places at the low level where the algorithm uses problem specific movement rules.

The base CBS algorithm resolves collisions between agents/tokens via adding constraints that forbid resolved collisions to happen again. The top level search of CBS uses priority queue that stores partial solutions together with a set of *conflicts*. The priority is determined by the value of the objective for a partial solution. In addition to this, CBS has a *low level search* that finds shortest paths connecting agent's/token's initial positions and goals while ignoring collisions between them. The low level search uses a set of conflicts that the path has to avoid. Conflicts are triples (a, v, t) with $a \in A$, $v \in V$, and timestep t which means that the path being searched for agent a must avoid v at timestep t.

Initially, we use shortest paths without considering any conflicts as an initial partial solution and store it into the priority queue. The top level search always takes the partial solution N with the lowest value of the objective and set of constraints forbidding conflicts $N.constraints$ associated with N. N is then checked for collisions between agents. If there are no collisions, then we can return $N.paths$ as an optimal solution. Otherwise we take the first collision; let this collision happened between agents a_i and a_j in vertex v at timestep t. Collision (a_i, a_j, v, t) will be resolved via branching at the top level search. One or the other involved agent must avoid v at t. This is carried out by adding two new states into the priority queue: partial solutions obtained by the low level search with respect to $N.constraints \cup \{(a_i, v, t)\}$ and $P.constrains \cup \{(a_j, v, t)\}$ associated with their extended conflict sets. For details on CBS we refer the reader to [14].

Considering TSWAP Specific Collisions: In TSWAP, collisions between tokens are understood differently than in MAPF. A collision between tokens c_i and c_j occurs in the following cases:

- tokens c_i and c_j both attempt to occupy vertex v at time step t (the analogical collision appears in MAPF too)
- token c_i traverses edge $\{u, v\}$ from vertex u to v at timestep t but c_j appearing in v at timestep t does not traverse $\{u, v\}$ from v to u at timestep t but goes to some $w \in V$ such that $w \neq u$ instead

Above cases represent collisions (c_i, c_j, v, t) and $(c_i, c_j, \{u, v, w\}, t)$ that are resolved by introducing conflicts (c_i, v, t) and (c_j, v, t) or $(c_i, \{u, v\}, t)$ and $(c_j, \{v, w\}, t)$ and by updating partial solutions accordingly. The collision of the second type as unique to TSWAP - it is a collision across edges $\{u, v\}$ and $\{v, w\}$ that is resolved in a similar way as a standard vertex collision. That is, either c_i or c_j is forbidden to traverse $\{u, v\}$ or $\{v, w\}$ respectively at timestep t. For more details on the CBS(TSWAP) algorithm we refer the reader to [11].

Making CBS Suboptimal: Making CBS suboptimal in order the trade-off the speed of solving and the quality of solutions is done by modifying its search both on the high level and the low level. The original search of the constraint tree relying on the priority queue is replaced with *focal search* [19], [42]. Similarly the search for a shortest path that avoids conflicts originally done by the A* algorithm with standard heuristics can use focal search instead.

IV. ADAPTING THE SMT-CBS ALGORITHM

A major alternative to CBS for TSWAP solving is represented by compilation of the problem to Boolean satisfiability (SAT) [11], [40], [43] and using the satisfiability modulo (SMT) theory approach [18].

A. SAT-based Approach

The idea in the basic SAT approach is to construct a Boolean formula whose satisfiability corresponds to the existence of a solution consisting of σ token swaps to a given TSWAP Ψ. Moreover, the approach is constructive; that is, the formula exactly reflects the TSWAP instance and if satisfiable, solution of TSWAP can be reconstructed from satisfying assignment of the formula.

There are two ways how to connect satisfiability of the formula and solvability of Ψ: using either **equivalence** or **implication**.

We say $\mathcal{F}(\sigma)$ to be a *complete Boolean model* of TSWAP.

Definition 4: **(complete Boolean model)** Boolean formula $\mathcal{F}(\sigma)$ is a *complete Boolean model* of TSWAP Ψ if the following condition holds:

- $\mathcal{F}(\sigma)$ is satisfiable $\Leftrightarrow$ Ψ has a solution of consisting of σ swaps.

A complete Boolean model was used in the MDD-SAT solver [15], the first sum-of-costs optimal SAT-based solver for the MAPF problem, and in its derivative MDD-SAT(TSWAP) for TSWAP [11].

A natural relaxation from the complete Boolean model is an *incomplete Boolean model* where instead of the equivalence between solving TSWAP and the formula we require an implication only. Incomplete models are inspired from the SMT paradigm are used in the recent sum-of-costs optimal solver SMT-CBS [18].

Definition 5: **(incomplete Boolean model).** Boolean formula $\mathcal{H}(\sigma)$ is an *incomplete Boolean model* of TSWAP Ψ if the following condition holds:

- $\mathcal{H}(\sigma)$ is satisfiable $\Leftarrow$ Ψ has a solution consisting of σ swaps.

Being able to construct formula $\mathcal{F}$ one can obtain optimal TSWAP solution by checking satisfiability of $\mathcal{F}(0)$, $\mathcal{F}(1)$, $\mathcal{F}(2)$,... until the first satisfiable $\mathcal{F}(\sigma)$ is met. This is possible due to monotonicity of TSWAP solvability with respect to increasing values of common cumulative objectives such as the number of swaps. In practice it is however impractical to start at 0; lower bound estimation is used instead - sum of lengths of shortest paths can be used in the case we optimize the number of swaps. The framework of the basic SAT-based solving is shown in pseudo-code in Algorithm 1.

The advantage of the SAT-based approach is that state-of-the-art SAT solvers can be used for determining satisfiability of $\mathcal{F}(\sigma)$ [44] and any progress in SAT solving hence can be utilized for increasing efficiency of TSWAP solving.

The SAT model of TSWAP relies on a *time expansion* of G. That is, we have a copy of G for every timestep [37]. Search for plans for individual tokens can then be interpreted as a search for non-conflicting directed paths in the time expanded graph (a token can visit a vertex multiple times hence plans for individual tokens cannot be easily expressed as simple paths in the unexpanded graph).

We introduce a Boolean variable $\mathcal{Y}(c)_v^t$ representing occurrence of a token of color c in vertex v at time step t and analogically $\mathcal{S}(c)_{u,v}^t$ for $c \in C$ representing swap at edge $\{u, v\}$ involving color c (color c starts in u). Constraints such as those enforcing that only one color can assigned to each vertex are described in details in [11]. Let us only briefly summarize important constraints.

Algorithm 1: Framework of SAT-based TSWAP

```
1  SAT-TSWAP(G = (V, E), C, τ_0, τ_+)
2      paths ← {shortest path from τ_0^{-1}(c_i) to
         τ_+^{-1}(c_i) | i = 1, 2, ..., k}
3      σ ← Σ_{i=1}^{k} length(paths(c_i))
4      μ ← max_{i=1}^{k} length(paths(c_i))
5      while True do
6          F(σ) ← encode-Complete(μ, σ, G, C, τ_0, τ_+)
7          assignment ← consult-SAT-Solver(F(σ))
8          if assignment ≠ UNSAT then
9              swaps ← extract-Solution(assignment)
10             return swaps
11         σ ← σ + 1; μ ← μ + 1
```

For instance, swapping tokens along edge $\{u, v\}$ at time step t means we need to replace an outgoing token with that from the target vertex:

$$\mathcal{S}(c)_{u,v}^t \Rightarrow \mathcal{Y}(c)_v^t \wedge \neg\mathcal{Y}(c)_v^{t+1} \qquad (1)$$

$$\mathcal{S}(c)_{u,v}^t \Rightarrow \bigvee_{d \in C} \mathcal{S}(d)_{v,u}^t$$

At any time step t assignment of variables must encode a valid placement of tokens in vertices of the graph. So at most one color is assigned to each vertex ensured by the following constraint for each vertex v and timestep t.

$$\sum_{c \in C} \mathcal{Y}(c)_v^t \leq 1$$

To illustrate other constraints we show how to enforce non-conflicting swaps of tokens. This can be expressed by *at-most-one* constraint over edge variables for a fixed vertex u and color c as follows. This constraint ensures that at most one swap occurs in a vertex a given timestep.

$$\sum_{\{u,v\} \in E} \mathcal{S}(c)_{u,v}^t \leq 1$$

The suggested encoding permits multiple non-conflicting swaps per single timestep (that is, $|F| > 1$ in Definition 1). Observe that rotations of tokens over *non-trivial cycles* (a swap is a trivial rotation over an edge) consisting of at least three edges is forbidden by the encoding as it would violate constraint (1).

Considering TSWAP Specific Objective: In TSWAP, we minimize the number of swaps. As the SAT-based approach always answers a given formula in a yes/no manner we need to translate the minimization of the number of swaps in TSWAP to a series of queries to the SAT-solver. We always build a formula using above constructs that encodes a question

whether there is a solution to TSWAP using specified number of swaps σ.

Encoding the σ bound can be done through various cardinality constraints in the formula [45]–[47] on top of edge variables $\mathcal{S}(c)^t_{u,v}$. Once an edge variable is set to $TRUE$ a corresponding swap is to be made. The following constraint hence need to be encoded in the formula through cardinality constraints [45]–[47].

$$\sum_{\{u,v\}\in E, c\in C, c<d, t=1,2,\ldots,\sigma} \mathcal{S}(c)^t_{u,v} \leq \sigma$$

As multiple token swaps can occur within the suggested encoding per single timestep, it is not necessary to make expansions for all steps up to σ. However careful setting of the number of time expansions with respect to given σ need to be done. We need to use sufficient number of expansions to ensure that if there is a swapping sequence of length σ we can find it within the given number of expansions. Precise calculation of the number of expansions depending on other parameters is shown in [15]. We omit the analogical calculation for TSWAP here for the sake of brevity.

B. SMT-based Approach

The second approach we modify for bounded sub-optimal TSWAP variant is SMT-CBS [18] that also builds a SAT model, but in this case an *incomplete* one. The incomplete model in constructed lazily inspired by techniques from satisfiability modulo theories (SMT).

The solving process in the context of SMT often divides satisfiability problem in some complex theory T into an abstract Boolean 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 is transformed to a *Boolean skeleton* by replacing atoms with Boolean variables. The standard SAT-solving procedure then decides what Boolean 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 and the skeleton is extended with a constraint forbidding the conflict.

The above observation stands behind rephrasing CBS in terms of SMT, the SMT-CBS algorithm. The paths validation procedure acts as $DECIDE_T$ and reports back a set of conflicts found in the current solution. Hence axioms of T are represented by the movement rules of TSWAP. SMT-CBS adapted for the TSWAP problem is listed as Algorithm 2.

The algorithm is divided into two procedures: SMT-CBS$_{\text{TSWAP}}$ representing the main loop and SMT-CBS-Fixed$_{\text{TSWAP}}$ solving the input TSWAP for a fixed number of swaps σ. The major difference from the standard CBS is that there is no branching at the high level. The high level SMT-CBS roughly correspond to the main loop of basic SAT-based solving though we are here rather working with the incomplete

Algorithm 2: SMT-based TSWAP solver

```
1  SMT-CBS_TSWAP (Ψ = (G = (V, E), C, τ_0, τ_+))
2      conflicts_V ← ∅; conflicts_E ← ∅
3      paths ← {path*(c_i) a shortest path from τ_0^{-1}(c_i) to
         τ_+^{-1}(c_i) | i = 1, 2, ..., k}
4      σ ← Σ_{i=1}^{k} length(paths(c_i))
5      μ ← max_{i=1}^{k} length(paths(c_i))
6      while TRUE do
7          (swaps, conflicts_V, conflicts_E) ←
             SMT-CBS-Fixed_TSWAP(conflicts_V, conflicts_E, μ, σ, Ψ)
8          if swaps ≠ UNSAT then
9              return swaps
10         σ ← σ + 1; μ ← μ + 1
11 SMT-CBS-Fixed_TSWAP(conflicts_V, conflicts_E μ, σ, Ψ)
12     H(σ) ←
         encode-Incomplete(conflicts_V, conflicts_E, μ, σ, Ψ)
13     while TRUE do
14         assignment ← consult-SAT-Solver(H(σ))
15         if assignment ≠ UNSAT then
16             swaps ← extract-Solution(assignment)
17             collisions_V ← validate-Vertex-Collisions(paths)
18             collisions_E ← validate-Edge-Collisions(paths)
19             if collisions_V ∪ collisions_E ≠ ∅ then
20                 return (swaps, conflicts_V, conflicts_E)
21             if collisions_V ≠ ∅ then
22                 for each (c_i, c_j, v, t) ∈ collisions_V do
23                     H(σ) ← H(σ) ∪ {¬Y(c_i)^t_v ∨ ¬Y(c_j)^t_v}
24                     conflicts_V ←
                         conflicts_V ∪ {[(c_i, v, t), (c_j, v, t)]}
25             if collisions_E ≠ ∅ then
26                 for each (c_i, c_j, {u, v, w}, t) ∈ collisions_V do
27                     H(σ) ←
                         H(σ) ∪ {¬S(c_i)^t_{u,v} ∨ ¬S(c_j)^t_{v,w}}
28                     conflicts_E ← conflicts_E ∪
                         {[(c_i, {u, v}, t), (c_j, {v, w}, t)]}
29         return (UNSAT, conflicts_V, conflicts_E)
```

model - procedure *encode-Complete* from the basic variant is replaced with *encode-Incomplete* that produces encoding that ignores specific swapping rules but in contrast to *encode-Complete* it encodes collected conflicts into $\mathcal{H}(\sigma)$. The set of conflicts is iteratively collected during the entire execution of the algorithm using variables $conflicts_V$ and $conflicts_E$ for vertex and edge conflicts respectively.

The conflict resolution in standard CBS implemented as high-level branching is here represented by refinement of $\mathcal{H}(\sigma)$ with disjunctions (lines 23 and 27). Branching is thus left to the SAT solver. The advantage of SMT-CBS is that it builds the formula lazily; that is, it adds constraints on demand after conflict occurs. Such approach may save resources as solution may be found before all constraint are added (that is before the complete model is constructed).

V. Bounded Suboptimal SMT-based Solver

The key to the new bounded-suboptimal SMT-based solver is the modification of the bound σ used in construction of $\mathcal{H}(\sigma)$. Inside $\mathcal{H}(\sigma$ the σ bound is used to bound the number of swaps using the *cardinality constraint* [47]. Since now the number of time expansions will not be derived directly

from σ parameter we will denote the formula as $\mathcal{H}(\mu, \sigma)$ where $\mu = \mu_0 + \Delta$ will be the number of time expansions, $\mu_0 = \max_{i=1}^{k} length(path^*(c_i))$ where $path^*(c_i)$ is the shortest path for c_i, and the second parameter will be the swap bound used in the cardinality constraint. Instead of incrementing σ we will increment Δ.

In SMT-CBS, σ is incremented by one in every iteration. Allowing $\sigma = \sigma_0 + \Delta$ parameter to be less restrictive, we will replace Δ with $\Delta' = \Delta + \delta$, where $\delta \geq 0$ is an integer value, produces formula of the same size but representing more solutions [2]. Since $\Delta' > \Delta$, we expect a formula with the number of swaps bounded by Δ' to be easier to solve than that with the original Δ.

The following proposition shows that for a solvable TSWAP Ψ the number of swaps of the solution obtained by the above process differs from the optimal one by at most δ.

Proposition 1: Let δ be a non-negative integer and let $\mathcal{H}(\mu_0 + \Delta, \sigma_0 + \Delta + \delta)$ be the first satisfiable formula corresponding to a valid TSWAP solution encountered in the sequence of formulae $\mathcal{H}(\mu_0, \sigma_0 + \delta)$, $\mathcal{H}(\mu_0 + 1, \sigma_0 + 1 + \delta)$,...,$\mathcal{H}(\mu_0 + \Delta - 1, \sigma_0 + \Delta + \delta - 1)$, $\mathcal{H}(\mu_0 + \Delta, \sigma_0 + \Delta + \delta)$. Then the solution represented by $\mathcal{H}(\mu_0 + \Delta, \sigma_0 + \Delta + \delta)$ has the number of swaps $\sigma \leq \sigma_{opt} + \delta$ where σ_{opt} is the optimal number of swaps for Ψ.

Proof: Formula $\mathcal{H}(\mu_0 + \Delta - 1, \Delta + \delta - 1)$ in the penultimate iteration could not be augmented by adding collision avoidance constraints to represent a valid solution and eventually became unsatisfiable. This means that no solution of makespan at most $\mu_0 + \Delta - 1$ and the number of swaps at most $\sigma_0 + \Delta + \delta - 1$ exists. But we also know that all solutions with $\sigma_0 + \Delta - 1$ swaps fit under the makespan of at most $\mu_0 + \Delta - 1$. Hence unsolvability of formula $\mathcal{H}(\mu_0 + \Delta - 1, \Delta + \delta - 1)$ together with $\delta \geq 0$ implies that there is no solution with $\sigma_0 + \Delta - 1$ swaps at all. Therefore, the optimal number of swaps is at least $\sigma_0 + \Delta$. The solvability of $\mathcal{H}(\mu_0 + \Delta, \Delta + \delta)$ says that there is a solution of Ψ consisting of $\sigma_0 + \Delta + \delta$ swaps which differs from the optimum by at most δ. ■

Observe that the only property of δ we used was that it is a non-negative integer but there is no requirement that it must be constant across individual iterations of the algorithm. Proposition 1 holds even if we use a non-negative δ as a function of Δ instead of a constant. This property can be used to modify the above SAT-based framework to an $(1+\epsilon)$-bounded suboptimal algorithm.

Corollary 1: Given an error $\epsilon > 0$ the SMT-based suboptimal framework can modified to an $(1+\epsilon)$-bounded suboptimal algorithm by appropriate setting of $\delta(\Delta)$.

Proof: Let $\delta(\Delta) = \epsilon \cdot (\sigma_0 + \Delta)$. Hence the sum-of-costs of the solution returned by the algorithm is at most $(1+\epsilon) \cdot (\sigma_0 + \Delta)$ while the optimum is at least $\sigma_0 + \Delta$ hence the ratio between the sum-of-costs of returned solution and the sum-of-costs of the optimal one is at most $(1 + \epsilon)$. ■

[2]The change from Δ to Δ' does not affect the number of clauses that represent the cardinality constraint, because we encode the cardinality constraints using a sequential counter, whose size is proportional to the number of Boolean variable involved but not to the value of the bound [46].

Algorithm 3: eSMT-CBS, an $(1 + \epsilon)$-bounded suboptimal SMT-based TSWAP solver

```
1  eSMT-CBS_TSWAP(Ψ = (G = (V, E), C, τ_0, τ_+))
2      conflicts_V ← ∅; conflicts_E ← ∅
3      paths ← {path*(c_i) a shortest path from τ_0^{-1}(c_i) to
         τ_+^{-1}(c_i) | i = 1, 2, ..., k}
4      σ_0 ← Σ_{i=1}^{k} length(paths(c_i))
5      μ_0 ← max_{i=1}^{k} length(paths(c_i))
6      Δ ← 0
7      while TRUE do
8          Δ' ← Δ + ε · (σ_0 + Δ)
9          (swaps, conflicts_V, conflicts_E) ←
             SMT-CBS-Fixed_TSWAP
10         (conflicts_V, conflicts_E, μ_0 + Δ, σ_0 + Δ', Ψ)
11         if swaps ≠ UNSAT then
12             return paths
13         Δ ← Δ + 1
```

The pseudo-code of the $(1 + \epsilon)$-bounded suboptimal SMT-based algorithm is presented as Algorithm 3. We refer to this algorithm as **eSMT-CBS**.

VI. EXPERIMENTAL EVALUATION

We performed an extensive evaluation of the eSMT-CBS algorithm on benchmarks for MAPF `movingai.com` [31], [48], [49]. Representative part of results is presented.

A. Benchmarks and Setup

We implemented eSMT-CBS in C++ using the existing implementation of SMT-CBS on top of the Glucose 3.0 SAT solver [50] that still ranks among the best SAT solvers according to recent SAT solver competitions [51]. Then we used existing implementation of ECBS [19] written in C# and modified it for the TSWAP problem which consisted of changing the rules for evaluating collisions.

All experiments were run on system consisting of Xeon 2.8 GHz cores, 32 GB RAM, running Ubuntu Linux 18 (for testing eSMT-CBS) and Windows 10 (for ECBS tests). [3]

The experimental evaluation has been done on diverse instances consisting of 4-connected *grid* maps of various sizes that we classify into 3 categories: small, medium, and large.

We varied the number of tokens in graphs while empty position has been treated as indistinguishable tokens of special color. This enables us to obtain instances of various difficulties. Initial configurations of tokens and graph colorings were generated randomly. Depending on the map size we used 64 to 128 tokens and 25 random instances per number of tokens. The timeout in all test was set to 128 seconds. Presented results were obtained from instances solved within this timeout. Both tested algorithms were used in 4 different setups with different error: $\epsilon = 1.00$ (optimal case), $\epsilon = 1.01$, $\epsilon = 1.05$, and $\epsilon = 1.10$.

[3]To enable reproducibility of presented results we provide complete source code of our solvers and detailed experimental data on author's web: `http://users.fit.cvut.cz/~surynpav/research/ictai2020`.

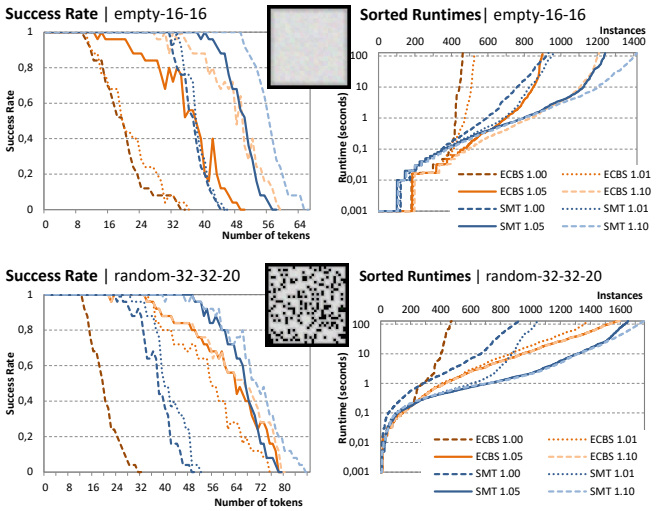

Fig. 2. Success rate and runtime comparison on small-sized maps.

B. Runtime Results

Results are presented in Figures 2, 3, and 4. We present *success rate* and *sorted runtimes*. Success rate shows the ratio of instances solved under the time limit of 128 seconds out of 25 instances per number of tokens. Sorted runtimes are inspired by cactus plots from the SAT Competition. We took runtimes of all instances solved under the time limit by a given algorithm and sorted them along x-axis; so the x-th data-point represents the runtime of x-th easiest instance for the given algorithm.

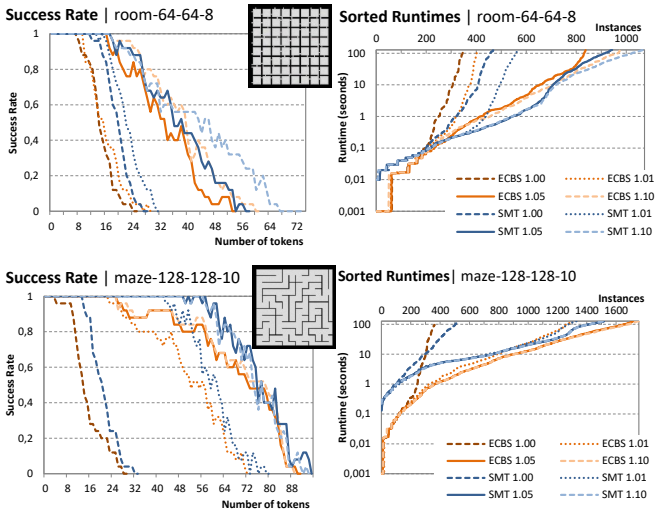

Fig. 3. Success rate and runtime comparison on medium-sized maps.

The general trend observable across all tested values of ϵ and all maps from sorted runtimes is that ECBS is faster for easy instances but its performance quickly degrades as instances gets harder. This is quite expectable since ECBS compared to SMT-CBS has smaller overhead but on the other hand lacks advanced learning and propagation mechanisms that help SMT-based in harder cases.

As instances get harder SMT-CBS starts to perform better than ECBS. In some cases ECBS experience sharp increase in the runtime after crossing certain level of difficulty (this is well observable in `empty-16-16` with $\epsilon = 1.01$ and $\epsilon = 1.05$, and in `empty-64-64-8` with $\epsilon = 1.01$ and $\epsilon = 1.05$).

There is significant difference in how relaxing the ϵ parameter reduces the difficulty. The most significant change happens by switching from the optimal TSWAP ($\epsilon = 1.00$) to slightly sub-optimal ($\epsilon = 1.01$) which dramatically reduces the difficulty for both tested solvers (in case of ECBS on `random-32-32-20` this change is the most dramatic). On the other hand, changing ϵ from 1.05 to 1.10 sometimes has almost no effect (especially in large instances like `warehouse-10-20-10-2-1` and `lak303d`).

The shortcoming of the basic variant of the SAT-based TSWAP solver was worse scalability for large maps compared to CBS/EBCS which was caused by the need to construct a huge formula. This shortcoming has been eliminated in SMT-CBS that constructs incomplete Boolean model that is significantly smaller than the complete model even on large maps. Therefore eSMT-CBS maintains its leads over ECBS in large maps like `lak303d`.

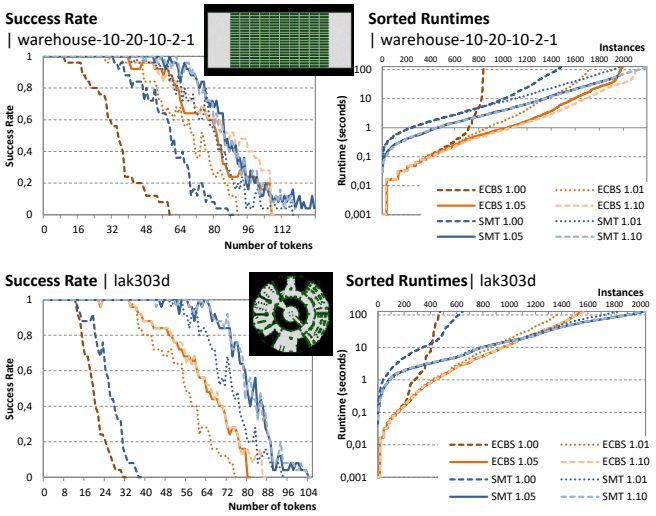

Fig. 4. Success rate and runtime comparison on large-sized maps.

VII. CONCLUSION

We introduced eSMT-CBS(TSWAP), a novel bounded sub-optimal algorithm for token swapping based on satisfiability modulo theories (SMT). The new algorithm combines strengths of SAT-based solving, which due to powerful clause learning mechanism and Boolean constraint propagation can successfully solve combinatorially hard cases of the token swapping problem, with lazy construction of the Boolean formula using SMT-inspired mechanism that enables scalability of the solver even for large instances. In addition to this, we adapted ECBS from its original multi-agent path finding domain, for the token swapping problem. The resulting ECBS(TSWAP) algorithm represents bounded sub-optimal search-based algorithm which shows its strength in token swapping especially when large error is permitted.

One of possible future research directions is to develop token swapping solvers both optimal and sub-optimal for special cases of the problem on grids or trees.

REFERENCES

[1] K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa, and T. Uno, "Swapping labeled tokens on graphs," in *FUN 2014 Proceedings*, ser. LNCS, vol. 8496. Springer, 2014, pp. 364–375.

[2] J. Kawahara, T. Saitoh, and R. Yoshinaka, "The time complexity of the token swapping problem and its parallel variants," in *WALCOM 2017 Proceedings.*, ser. LNCS, vol. 10167. Springer, 2017, pp. 448–459.

[3] É. Bonnet, T. Miltzow, and P. Rzazewski, "Complexity of token swapping and its variants," in *STACS 2017*, ser. LIPIcs, vol. 66. Schloss Dagstuhl, 2017, pp. 16:1–16:14.

[4] F. Basile, P. Chiacchio, and J. Coppola, "A hybrid model of complex automated warehouse systems - part I: modeling and simulation," *IEEE Trans. Automation Science and Engineering*, vol. 9, no. 4, pp. 640–653, 2012.

[5] D.-G. Kim, K. Hirayama, and G.-K. Park, "Collision avoidance in multiple-ship situations by distributed local search," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 18, pp. 839–848, 09 2014.

[6] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Proceedings of SOCS 2018*. AAAI Press, 2018, pp. 138–142.

[7] M. Thorup, "Randomized sorting in o(n log log n) time and linear space using addition, shift, and bit-wise boolean operations," *J. Algorithms*, vol. 42, no. 2, pp. 205–230, 2002.

[8] K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa, and T. Uno, "Swapping labeled tokens on graphs," *Theor. Comput. Sci.*, vol. 586, pp. 81–94, 2015.

[9] T. Miltzow, L. Narins, Y. Okamoto, G. Rote, A. Thomas, and T. Uno, "Approximation and hardness of token swapping," in *ESA 2016*, ser. LIPIcs, vol. 57. Schloss Dagstuhl, 2016, pp. 66:1–66:15.

[10] K. Yamanaka, E. D. Demaine, T. Horiyama, A. Kawamura, S. Nakano, Y. Okamoto, T. Saitoh, A. Suzuki, R. Uehara, and T. Uno, "Sequentially swapping colored tokens on graphs," in *WALCOM 2017 Proceedings.*, ser. LNCS, vol. 10167. Springer, 2017, pp. 435–447.

[11] P. Surynek, "Finding optimal solutions to token swapping by conflict-based search and reduction to SAT," in *Proceedings of ICTAI 2018*. IEEE, 2018, pp. 592–599.

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

[13] M. Ryan, "Exploiting subgraph structure in multi-robot path planning," *JAIR*, vol. 31, pp. 497–542, 2008.

[14] 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.

[15] 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.

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

[17] P. Surynek, "Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach," in *Proceedings of IROS 2019*. IEEE, 2019, pp. 3282–3287.

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

[19] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *ECAI 2014*, vol. 263. IOS Press, 2014, pp. 961–962.

[20] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t)," *J. ACM*, vol. 53, no. 6, pp. 937–977, 2006.

[21] C. W. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[22] K. Yamanaka, T. Horiyama, D. Kirkpatrick, Y. Otachi, T. Saitoh, R. Uehara, and Y. Uno, "Computational complexity of colored token swapping problem," in *IPSJ SIG Technical Report*, vol. 156, no. 2, 2016.

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

[24] 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.

[25] I. Parberry, "Memory-efficient method for fast computation of short 15-puzzle solutions," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 2, pp. 200–203, 2015.

[26] ——, "Solving the (n^2 - 1)-puzzle with 8/3 n^3 expected moves," *Algorithms*, vol. 8, no. 3, pp. 459–465, 2015.

[27] P. Surynek and P. Michalík, "The joint movement of pebbles in solving the (n^2 - 1)-puzzle suboptimally and its applications in rule-based cooperative path-finding," *Auton. Agents Multi Agent Syst.*, vol. 31, no. 3, pp. 715–763, 2017.

[28] V. Cahlík and P. Surynek, "On the design of a heuristic based on artificial neural networks for the near optimal solving of the (n2-1)-puzzle," in *Proceedings of IJCCI 2019*. ScitePress, 2019, pp. 473–478.

[29] T. Standley, "Finding optimal solutions to cooperative pathfinding problems." in *AAAI*, 2010, pp. 173–178.

[30] P. Surynek, "Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs," in *ICTAI*, 2014, pp. 875–882.

[31] 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.

[32] R. Luna and K. Bekris, "Efficient and complete centralized multi-robot path planning," in *IROS*, 2011, pp. 3268–3275.

[33] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artif. Intell.*, vol. 219, pp. 1–24, 2015.

[34] F. Aljalaud and N. R. Sturtevant, "Finding bounded suboptimal multi-agent path planning solutions using increasing cost tree search (extended abstract)," in *Proceedings of SOCS 2013*. AAAI Press, 2013.

[35] M. Ryan, "Constraint-based multi-robot path planning," in *ICRA*, 2010, pp. 922–928.

[36] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *PRICAI*, 2012, pp. 564–576.

[37] ——, "Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems," *Ann. Math. Artif. Intell.*, vol. 81, no. 3-4, pp. 329–375, 2017.

[38] J. Yu and S. LaValle, "Planning optimal paths for multiple robots on graphs," in *ICRA*, 2013, pp. 3612–3617.

[39] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller, "A general formal framework for pathfinding problems with multiple agents," in *AAAI*, 2013.

[40] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Sub-optimal sat-based approach to multi-agent path-finding problem," in *Proceedings of SOCS 2018*. AAAI Press, 2018, pp. 90–105.

[41] 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.

[42] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 4, no. 4, pp. 392–399, 1982.

[43] P. Surynek, "On propositional encodings of cooperative path-finding," in *Proceedings of ICTAI 2012*. IEEE Computer Society, 2012, pp. 524–531.

[44] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *Proceedings of SAT 2013*, 2013, pp. 309–317.

[45] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *CP*, 2003, pp. 108–122.

[46] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *CP*, 2005.

[47] J. Silva and I. Lynce, "Towards robust CNF encodings of cardinality constraints," in *CP*, 2007, pp. 483–497.

[48] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. Shimony, "ICBS: improved conflict-based search algorithm for multi-agent pathfinding," in *IJCAI*, 2015, pp. 740–746.

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

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

[51] T. Balyo, M. J. H. Heule, and M. Järvisalo, "SAT competition 2016: Recent developments," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 5061–5063.