Conflict Handling Framework in Generalized Multi-agent Path Finding: Advantages and Shortcomings of Satisfiability Modulo Approach

Pavel Surynek
Faculty of Information Technology, Czech Technical University in Prague, Thakurova 9, 160 00 Praha 6, Czech Republic

Keywords: Conflicts, MAPF, Token Swapping, Token Rotation, Token Permutation, SMT, SAT.

Abstract: We address conflict reasoning in generalizations of multi-agent path finding (MAPF). We assume items placed in vertices of an undirected graph with at most one item per vertex. Items can be relocated across edges while various constraints depending on the concrete type of MAPF must be satisfied. We recall a general problem formulation that encompasses known types of item relocation problems such as multi-agent path finding (MAPF) and token swapping (TSWAP). We show how to express new types of relocation problems in the general problem formulation. We thoroughly evaluate a novel solving method for item relocation that combines satisfiability modulo theory (SMT) with conflict-based search (CBS). CBS is interpreted in the SMT framework where we start with the basic model and refine the model with a collision resolution constraint whenever a collision between items occurs. The key difference between the standard CBS and the SMT-based modification of CBS (SMT-CBS) is that the standard CBS branches the search to resolve the collision while SMT-CBS iteratively adds a single disjunctive collision resolution constraint. Our experimental evaluation revealed that although SMT-CBS performs better than CBS in small densely occupied instances of variants of MAPF, it is outperformed on large sparsely occupied environments. The performed analysis shows that individual paths in large environments of relocation instances can be found faster using simple A*-based algorithm than by the SMT solver. On the other hand the SMT solver performs better when many conflicts between items need to be resolved.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Item relocation problems in graphs such as *token swapping* (TSWAP) (Kawahara et al., 2017; Bonnet et al., 2017), *multi-agent path finding* (MAPF) (Ryan, 2007; Standley, 2010; Yu and LaValle, 2013), or pebble motion on graphs (PMG) (Wilson, 1974; Kornhauser et al., 1984) represent important combinatorial problems in artificial intelligence with specific applications in coordination of multiple robots and other areas such as quantum circuit compilation (Botea et al., 2018). We assume multiple distinguishable items placed in vertices of an undirected graph such that at most one item is placed in each vertex. Items can be moved between vertices across edges while problem specific rules must be observed. For example, PMG and MAPF usually assume that items (pebbles/agents) are moved to unoccupied neighbors only. Sometimes in MAPF it is also possible that items form a train (sequence of items) and the entire train moves simultaneously while only the leading item needs to enter a vacant vertex[1]. TSWAP on the other hand permits only swaps of pairs of tokens along edges while more complex movements involving more than two tokens are forbidden. The task in item relocation problems is to reach a given goal configuration of items from a given starting configuration using allowed movements.

We focus here on the optimal solving of item relocation problems with respect to common cumulative objective functions. Two cumulative objective functions are used in MAPF and TSWAP - *sum-of-costs* (Sharon et al., 2013; Miltzow et al., 2016) and *makespan* (Surynek, 2014a; Yu and LaValle, 2016). The sum-of-costs corresponds to the total cost of all movements performed until the goal configuration is reached - the traversal of an edge by an item has unit cost. The makespan calculates the total number of time-steps until the goal is reached. In both cases we trying to minimize the objective which in the case of sum-of-costs intuitively corresponds to energy mini-

[1]This variant of MAPF is sometimes called a parallel MAPF or parallel PMG (Surynek, 2010).

Surynek, P.
Conflict Handling Framework in Generalized Multi-agent Path finding: Advantages and Shortcomings of Satisfiability Modulo Approach.
DOI: 10.5220/0007374201920203
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 192-203
ISBN: 978-989-758-350-6
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

mization while the minimization of makespan corresponds to minimization of time.

Many practical problems from robotics involving multiple robots can be interpreted as an item relocation problems. Examples include discrete multi-robot navigation and coordination (Luna and Bekris, 2010), item rearrangement in automated warehouses (Basile et al., 2012), ship collision avoidance (Kim et al., 2014), or formation maintenance and maneuvering of aerial vehicles (Zhou and Schwager, 2015). Examples not only include problems concerning physical items but problems occurring in virtual spaces of simulations (Kapadia et al., 2013), computer games (Wender and Watson, 2014), or quantum systems (Botea et al., 2018).

The contribution of this paper consists in a thorough experimental evaluation of a general framework for defining and solving item relocation problems based on *satisfiability modulo theories* (SMT) (Bofill et al., 2012; Surynek, 2018b) and *conflict-based search* (CBS) (Sharon et al., 2015).

The framework has been used to define two problems derived from TSWAP: *token rotation* (TROT) and *token permutation* (TPERM) where instead of swapping pairs of tokens, rotations along non-trivial cycles and arbitrary permutations of tokens respectively are permitted. We show how to deal with all MAPF related item relocation problems through conflict reasoning in two algorithms suitable for this task - SMT-CBS and CBS algorithms. Tests on various benchmarks revealed that there is no universal winner in solving item relocation problems among the tested algorithms. While SMT-CBS turned out to be better in small densely-populated instances, CBS exhibited better performance in large environments containing few items. These results are in line with previous results for the SAT-based MAPF solving where SAT-based solvers usually perform well on small instances and worse on larger ones (Surynek et al., 2016a; Surynek et al., 2016b).

The organization of the paper is as follows. We first introduce TSWAP and MAPF problems formally. Then prerequisites for conflict handling in item relocation problems formulated in the SMT framework are recalled, that is, we recall the CBS algorithm and the MDD-SAT algorithm. On top of this, the combination of CBS and MDD-SAT is developed - the SMT-CBS algorithm. Finally, a thorough experimental evaluation of CBS and SMT-CBS on various benchmarks including both small and large instances is presented.

2 BACKGROUND

We briefly recall *multi-agent path finding* and *token swapping* in this section.

Multi-agent path finding (MAPF) problem (Silver, 2005; Ryan, 2008) consists of an undirected graph $G = (V, E)$ and a set of agents $A = \{a_1, a_2, ..., a_k\}$ such that $|A| < |V|$. Each agent is placed in a vertex so that at most one agent resides in each vertex. The placement of agents is denoted $\alpha : A \rightarrow V$. Next we are given nitial 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. Typically, an agent can move into adjacent unoccupied vertex provided no other agent enters the same target vertex but other rules for movements are used as well. An example of MAPF instance is shown in Figure 1.

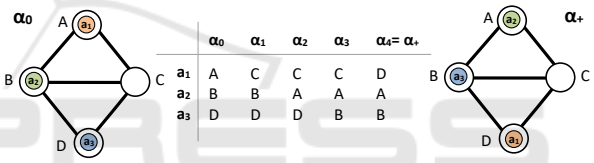

Figure 1: A MAPF instance with three agents a_1, a_2, and a_3.

The following definition formalizes the commonly used *move-to-unoccupied* movement rule in MAPF.

Definition 1. Movement in MAPF. *Configuration* α' *results from* α *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 if* $\alpha(a) \neq \alpha'(a) \Rightarrow \alpha'(a) \neq \alpha(a')$ *for all* $a' \in A$ *(target vertex must be empty); 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).*

Solving the MAPF instance is to search for a sequence of configurations $[\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that α_{i+1} results using valid movements from α_i for $i = 1, 2, ..., \mu - 1$, and $\alpha_\mu = \alpha_+$.

In many aspects, a *token swapping problem (TSWAP)* (also known as *sorting on graphs*) (Yamanaka et al., 2014) is similar to MAPF. It represents a generalization of sorting problems (Thorup, 2002). While in the classical sorting problem we need to obtain linearly ordered sequence of elements by swapping any pair of elements, in the TSWAP problem we

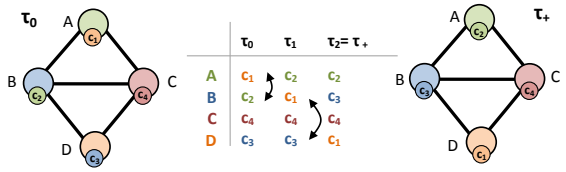

Figure 2: A TSWAP instance. A solution consisting of two swaps is shown.

are allowed to swap elements at selected pairs of positions only.

Using a modified notation from (Yamanaka et al., 2015) the TSWAP each vertex in G is assigned a color in $C = \{c_1, c_2, ..., c_h\}$ via $\tau_+ : V \to C$. A token of a color in C is placed in each vertex. The task is to transform a current 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 2 for an example instance of TSWAP.

We denote by $\tau : V \to C$ colors of tokens placed in vertices of G. That is, $\tau(v)$ for $v \in V$ is a color of a token placed in v. Starting placement of tokens is denoted as τ_0; the goal token placement corresponds to τ_+. Transformation of one placement to another is captured by the concept of *adjacency* defined as follows (Yamanaka et al., 2015; Yamanaka et al., 2017):

Definition 2. Adjacency in TSWAP. *Token placements τ and τ' are said to be adjacent if there exists 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 $\tau(w) = \tau'(w)$.* [2]

The task in TSWAP is to find a swapping sequence of token placements $[\tau_0, \tau_1, ..., \tau_m]$ such that $\tau_m = \tau_+$ and τ_i and τ_{i+1} are adjacent for all $i = 0, 1, ..., m-1$. 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 $O(|V|^2)$ swaps (Yamanaka et al., 2016). The proof is based on swapping tokens on 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 a swapping sequence that has as few swaps as possible is an NP-hard problem.

If each token has a different color we do not distinguish between tokens and their colors c_i; that is, we will refer to a token c_i.

Observe, that the operational meaning of agents and tokens in MAPF and TSWAP is similar. They both occupy vertices of the graph and no two of them can share a vertex. Hence works studying relation of both problems from the practical solving perspective have appeared recently (Surynek, 2018a).

3 RELATED WORK

Although many works sudying TSWAP from the theoretical point of view exist (Yamanaka et al., 2016; Miltzow et al., 2016; Bonnet et al., 2017) practical solving of the problem started only lately. In (Surynek, 2018a) optimal solving of TSWAP by adapted algorithms from MAPF has been suggested. Namely *conflict-based search* (CBS) (Sharon et al., 2012; Sharon et al., 2015) and *propositional satisfiability-based* (SAT) (Biere et al., 2009) MDD-SAT (Surynek et al., 2016a; Surynek et al., 2016b) originally developed for MAPF have been modified for TSWAP.

3.1 Search for Optimal Solutions

We will commonly use the *sum-of-costs* objective funtion in all problems studied in this paper. The following definition introduces the sum-of-costs objective in MAPF. However, analogous definition can be introduced for TSWAP too.

Definition 3. Sum-of-costs *(denoted ξ) is the summation, over all agents, of the number of time steps required to reach the goal vertex (Dresner and Stone, 2008; Standley, 2010; Sharon et al., 2013; Sharon et al., 2015). Formally, $\xi = \sum_{i=1}^{k} \xi(path(a_i))$, where $\xi(path(a_i))$ is an individual path cost of agent a_i connecting $\alpha_0(a_i)$ calculated as the number of edge traversals and wait actions.* [3]

Observe that in the sum-of-costs we accumulate the cost of wait actions for items not yet reaching their goal vertices. Also observe that one swap in the TSWAP problem correspond to the cost of 2 as two tokens traverses single edge. Let us note that all algorithms and concepts we use can be modified for different cummulative objective functions like makespan or the total number of moves/swaps etc.

A feasible solution of a solvable MAPF instance can be found in polynomial time (Wilson, 1974; Kornhauser et al., 1984); precisely the worst case time complexity of most practical algorithms for finding feasible solutions is $O(|V|^3)$ (asymptotic size of the solution is also $O(|V|^3)$) (Surynek, 2009b; Surynek,

[2] The presented version of adjacency is sometimes called *parallel* while a term adjacency is reserved for the case with $|F| = 1$.

[3] The notation $path(a_i)$ refers to path in the form of a seqeunce of vertices and edges connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$ while ξ assigns the cost to a given path.

2009a; Surynek, 2014b; Luna and Bekris, 2011a; Luna and Bekris, 2011b; de Wilde et al., 2014). This is also asymptotically best possible as there are MAPF instances requiring $\Omega(|V|^2)$ moves. As with TSWAP, finding optimal MAPF solutions with respect to various cummulative objectives is NP-hard (Ratner and Warmuth, 1986; Surynek, 2010; Yu and LaValle, 2015).

3.2 Conflict-based Search

CBS uses the idea of resolving conflicts lazily; that is, a solution of MAPF instance is not searched against the complete set of movement constraints that forbids collisions between agents but with respect to initially empty set of collision forbidding constraints that gradually grows as new conflicts appear. The advantage of CBS is that it can find a valid solution before all constraints are added.

The high level of CBS searches a *constraint tree* (CT) using a priority queue in breadth first manner. CT is a binary tree where each node N contains a set of collision avoidance constraints $N.constraints$ - a set of triples (a_i, v, t) forbidding occurrence of agent a_i in vertex v at time step t, a solution $N.paths$ - a set of k paths for individual agents, and the total cost $N.\xi$ of the current solution.

The low level process in CBS associated with node N searches paths for individual agents with respect to set of constraints $N.constraints$. For a given agent a_i, this is a standard single source shortest path search from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ that avoids a set of vertices $\{v \in V | (a_i, v, t) \in N.constraints\}$ whenever working at time step t. For details see (Sharon et al., 2015).

CBS stores nodes of CT into priority queue OPEN sorted according to ascending costs of solutions. At each step CBS takes node N with lowest cost from OPEN and checks if $N.paths$ represents paths that are valid with respect to movements rules in MAPF. That is, if there are any collisions between agents in $N.paths$. If there is no collision, the algorithms returns valid MAPF solution $N.paths$. Otherwise the search branches by creating a new pair of nodes in CT - successors of N. Assume that a collision occurred between agents a_i and a_j in vertex v at time step t. This collision can be avoided if either agent a_i or agent a_j does not reside in v at timestep t. These two options correspond to new successor nodes of N - N_1 and N_2 that inherits set of conflicts from N as follows: $N_1.conflicts = N.conflicts \cup \{(a_i, v, t)\}$ and $N_2.conflicts = N.conflicts \cup \{(a_j, v, t)\}$. $N_1.paths$ and $N_1.paths$ inherit path from $N.paths$ except those for agent a_i and a_j respectively. Paths for a_i and a_j are recalculated with respect to extended sets of conflicts $N_1.conflicts$ and $N_2.conflicts$ respectively and new costs for both agents $N_1.\xi$ and $N_2.\xi$ are determined. After this N_1 and N_2 are inserted into the priority queue OPEN.

Algorithm 1: Basic CBS algorithm for MAPF solving.

```
1  CBS (G = (V,E), A, α0, α+)
2      R.constraints ← ∅
3      R.paths ← {shortest path from α0(ai) to
       α+(ai) | i = 1,2,...,k}
4      R.ξ ← Σ_{i=1}^{k} ξ(N.paths(ai))
5      insert R into OPEN
6      while OPEN ≠ ∅ do
7          N ← min(OPEN)
8          remove-Min(OPEN)
9          collisions ← validate(N.paths)
10         if collisions = ∅ then
11             return N.paths
12         let (ai, aj, v, t) ∈ collisions
13         for each a ∈ {ai, aj} do
14             N'.constraints ←
               N.constraints ∪ {(a, v, t)}
15             N'.paths ← N.paths
16             update(a, N'.paths, N'.conflicts)
17             N'.ξ ← Σ_{i=1}^{k} ξ(N'.paths(ai))
18             insert N' into OPEN
```

The pseudo-code of CBS is listed as Algorithm 1. One of crucial steps occurs at line 16 where a new path for colliding agents a_i and a_j is constructed with respect to an extended set of conflicts. Notation $N.paths(a)$ refers to the path of agent a.

The CBS algorithm ensures finding sum-of-costs optimal solution. Detailed proofs of this claim can be found in (Sharon et al., 2015).

3.3 SAT-based Approach

An alternative approach to optimal MAPF solving as well as to TSWAP solving is represented by reduction of MAPF to propositional satisfiability (SAT) (Surynek, 2012b; Surynek, 2012a). The idea is to construct a propositional formula such $\mathcal{F}(\xi)$ such that it is satisfiable if and only if a solution of a given MAPF of sum-of-costs ξ exists. Moreover, the approach is constructive; that is, $\mathcal{F}(\xi)$ exactly reflects the MAPF instance and if satisfiable, solution of MAPF can be reconstructed from satisfying assignment of the formula.

Being able to construct such formula $\mathcal{F}$ one can obtain optimal MAPF solution by checking satisfiability of $\mathcal{F}(0)$, $\mathcal{F}(1)$, $\mathcal{F}(2)$,... until the first satisfiable $\mathcal{F}(\xi)$ is met. This is possible due to monotonicity of MAPF solvability with respect to increasing values of

Algorithm 2: Framework of SAT-based MAPF solving.

```
1  SAT-Based (G = (V,E), A, α0, α+)
2      paths ← {shortest path from α0(ai) to
       α+(ai) | i = 1,2,...,k}
3      ξ ← Σ_{i=1}^{k} ξ(N.paths(ai))
4      while True do
5          F(ξ) ← encode(ξ, G, A, α0, α+)
6          assignment ← consult-SAT-Solver(F(ξ))
7          if assignment ≠ UNSAT then
8              paths ← extract-Solution(assignment)
9              return paths
10         ξ ← ξ + 1
```

common cummulative objectives such as the sum-of-costs. 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 of sum-of-costs. The framework of SAT-based solving is shown in pseudo-code in Algorithm 2.

The advantage of the SAT-based approach is that state-of-the-art SAT solvers can be used for determinig satisfiability of $\mathcal{F}(\xi)$ (Audemard et al., 2013) and any progress in SAT solving hence can be utilized for increasing efficiency of MAPF solving.

3.4 Multi-value Decision Diagrams - MDD-SAT

Construction of $\mathcal{F}(\xi)$ relies on time expansion of underlying graph G (Surynek, 2017). Having ξ, the basic variant of time expansion determines the maximum number of time steps μ (also refered to as a *makespan*) such that every possible solution of the given MAPF with the sum-of-costs less than or equal to ξ fits within μ timestep (that is, no agent is outside its goal vertex after μ timestep if the sum-of-costs ξ is not to be exceeded).

Time expansion itself makes copies of vertices V for each timestep $t = 0,1,2,...,\mu$. That is, we have vertices v^t for each $v \in V$ time step t. Edges from G are converted to directed edges interconnecting timesteps in time expansion. Directed edges (u^t, v^{t+1}) are introduced for $t = 1,2,...,\mu-1$ whenever there is $\{u,v\} \in E$. Wait actions are modeled by introducing edges (u^t, t^{t+1}). A directed path in time expansion corresponds to trajectory of an agent in time. Hence the modeling task now consists in construction of a formula in which satisfying assignments correspond to directed paths from $\alpha_0^0(a_i)$ to $\alpha_+^\mu(a_i)$ in time expansion.

Assume that we have time expansion $TEG_i = (V_i, E_i)$ for agent a_i. Propositional variable $\mathcal{X}_v^t(a_j)$ is introduced for every vertex v^t in V_i. The semantics of $\mathcal{X}_v^t(a_i)$ is that it is *True* if and only if agent a_i resides in v at time step t. Similarly we introduce $\mathcal{E}_u, v^t(a_i)$ for every directed edge (u^t, v^{t+1}) in E_i. Analogously the meaning of $\mathcal{E}_{u,v}^t(a_i)$ is that is *True* if and only if agent a_i traverses edge $\{u,v\}$ between time steps t and $t+1$.

Finally constraints are added so that truth assignment are restricted to those that correspond to valid solutions of a given MAPF. The detailed list of constraints is given in (Surynek et al., 2016a). We here just illustrate the modeling by showing few representative constraints. For example there is a constraint stating that if agent a_i appears in vertex u at time step t then it has to leave through exactly one edge (u^t, v^{t+1}). This can be established by following constraints:

$$\mathcal{X}_u^t(a_i) \Rightarrow \bigvee_{(u^t,v^{t+1}) \in E_i} \mathcal{E}_{u,v}^t(a_i), \tag{1}$$

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

Similarly, the target vertex of any movement except wait action must be empty. This is ensured by the following constraint for every $(u^t, v^{t+1}) \in E_i$:

$$\mathcal{E}_{u,v}^t(a_i) \Rightarrow \bigwedge_{a_j \in A \wedge a_j \neq a_i \wedge v^t \in V_j} \neg \mathcal{X}_v^t(a_j) \tag{3}$$

Other constraints ensure that truth assignments to variables per individual agents form paths. That is if agent a_i enters an edge it must leave the edge at the next time step.

$$\mathcal{E}_{u,v}^t(a_i) \Rightarrow \mathcal{X}_v^t(a_i) \wedge \mathcal{X}_v^{t+1}(a_i) \tag{4}$$

Agents do not collide with each other; the following constraint is introduced for every $v \in V$ and timestep t:

$$\sum_{i=1,2,...,k | v^t \in V_i} \mathcal{X}_v^t(a_i) \tag{5}$$

A common measure how to reduce the number of decision variables derived from the time expansion is the use of *multi-value decision diagrams* (MDDs) (Sharon et al., 2013). The basic observation that holds for MAPF and other item relocation problems is that a token/agent can reach vertices in the distance d (distance of a vertex is measured as the length of the shortest path) from the current position of the agent/token no earlier than in the d-th time step. Analogical observation can be made with respect to the distance from the goal position.

Above observations can be utilized when making the time expansion of G. For a given agent or token, we do not need to consider all vertices at time step t

but only those that are reachable in t timesteps from the initial position and that ensure that the goal can be reached in the remaining $\sigma - t$ timesteps. This idea can reduce the size the expansion graph significantly and consequently can reduce the size of the Boolean formula by eliminating $\mathcal{X}(a)_v^t$ and $\mathcal{E}(a)_{u,v}^t$ variables correspoding to unreachable vertices u and v.

A comparison of standard time expansion and MDD expansion in MAPF for agent (a_i) is shown in Figure 3.

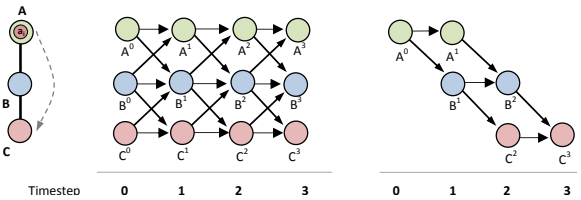

Figure 3: An example of time expansion and MDD expansion for agent a_1.

The combination of SAT-based approach and MDD time expansion led to the MDD-SAT algorithm described in (Surynek et al., 2016a) that currently represent state-of-the-art in SAT-based MAPF solving.

4 GENERALIZATIONS OF ITEM RELOCATION

Although the differences between MAPF and TSWAP led to different worst case time complexities in algorithms for finding feasible solutions, problems differ only in local understanding of conflicts reflected in different movement rules in fact. This immediately inspired us to suggest various modifications of movement rules.

We define two problems derived from MAPF and TSWAP: *token rotation* (TROT) and *token permutation* (TPERM) [4].

4.1 Token Rotation and Token Permutation

A swap of pair of tokens can be interpreted as a rotation along a trivial cycle consisting of single edge. We can generalize this towards longer cycles. The TROT problem permits rotations along longer cycles but forbids trivial cycles; that is, rotations along triples, quadruples, ... of vertices is allowed but swap along edges are forbidden.

Definition 4. Adjacency in TROT. *Token placements τ and τ' are said to be adjacent in TROT if there exists a subset of edges $F \subseteq E$ such that components $C_1, C_2, ..., C_p$ of induced sub-graph $G[F]$ satisfy following conditions:*

(i) $C_j = (V_j^C, E_j^C)$ *such that* $V_j^C = w_1^j, w_2^j, ..., w_{n_j}^j$ *with* $n_j \leq 3$ *and*
$E_j^C = \{\{w_1^j, w_2^j\}; \{w_2^j, w_3^j\}; ...; \{w_{n_j}^j, w_1^j\}\}$
(components are cycles of length at least 3)

(ii) $\tau(w_1^j) = \tau'(w_2^j),\ \tau(w_3^j) = \tau'(w_3^j),\ ...,\ \tau(w_{n_j}^j) = \tau'(w_1^j)$
(colors are rotated in the cycle one position forward/backward)

The rest of the definition of a TROT instance is analogical to TSWAP.

Similarly we can define TPERM by permitting all lengths of cycles. The formal definition of *adjacency* in TPERM is almost the same as in TROT except relaxing the constraint on cycle lenght, $n_j \leq 2$.as

We omit here complexity considerations for TROT and TPERM for the sake of brevity. Again it holds that a feasible solution can be found in polynomial time but the optimal cases remain intractable in general.

Both approaches - SAT-based MDD-SAT as well as CBS - can be adapted for solving TROT and TPERM without modifying their top level design. Only local modification of how movement rules of each problem are reflected in algorithms is necessary. In case of CBS, we need to define what does it mean a conflict in TROT and TPERM. In MDD-SAT different movement constraints can be encoded directly.

Motivation for studying these item relocation problems is the same as for MAPF. In many real-life scenarios it happens that items or agents enters positions being simultaneously vacated by other items (for example mobile robots often). This is exactly the property captured formally in above definitions.

4.2 Adapting CBS and MDD-SAT

Both CBS and MDD-SAT can be modified for optimal solving of TSWAP, TROT, and TPERM (with respect to sum-of-costs but other cumulative objectives are possible as well). Different movement rules can be reflected in CBS and MDD-SAT algorithms without modifying their high level framework.

[4] These problems have been considered in the literature in different contexts already (for example in (Yu and Rus, 2014)). But not from the practical solving perspective focused on finding optimal solutions.

4.2.1 Different Conflicts in CBS

In CBS, we need to modify the understanding of conflict between agents/tokens. In contrast to the original CBS we need to introduce edge conflicts to be able to handle conflicts properly in TSWAP and TROT.

Consider an example from Figure 4 that concerns TSWAP being solved by CBS. The situation when token c_1 traverses from A to B and simultaneously token c_2 traverses from B to C cannot occur in TSWAP. However we cannot properly branch in CBS using vertex conflicts to tackle this situation. Although all movements of c_2 from B into any other vertex than A can be ruled out by vertex conflicts, wrong movement of c_2 into A from a vertex other than B remains allowed.

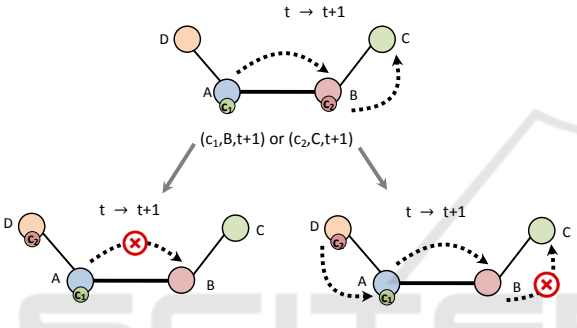

Figure 4: A collision that leads to wrong reasoning with *vertex conflicts* in TSWAP. Vertex conflict (c_2,C,t) does not properly forbid simultaneous movements from A to B and from B to C. At some later stage in the sub-tree after (c_2,C,t) it would be still possible that although c_2 cannot move into any other vertex than A at t it can move into A from other vertex than B which is not desirable.

Therefore *edge conflicts* have been introduced to tackle conflicting situations in TSWAP and TROT properly within CBS and SMT-CBS. An edge conflict is triple $(c_i,(u,v),t)$ with $c_i \in C$, $u,v \in V$ and timestep t. The interpretation of $(c_i,(u,v),t)$ is that token c_i cannot move across $\{u,v\}$ from u to v between timesteps t and $t+1$.

Edge conflict can be used to resolve collision from 4. Instead of forbidding c_2 to enter any vertex other than A at timestep t movements of c_2 across all edges other than (B,A) at timestep t are forbidden. Analogical collision resolution using edge conflicts can be applied in the TROT problem.

Conflict reasoning in individual item relocation problems derived from MAPF is described in the following paragraphs. Proofs of soundness of conflict reasoning are omitted here.

TPERM: The easiest case is TPERM as it is least restrictive. We merely forbid simultaneous occurrence of multiple tokens in a vertex - this situation is understood as a collision in TPERM and conflicts are derived from it. If a collision (c_i,c_j,v,t) between tokens c_i and c_j occurs in v at time step t then we introduce conflicts (c_i,v,t) and (c_j,v,t) for c_i and c_j respectively. [5]

TSWAP: This problem takes conflicts from TPERM but adds new conflicts that arise from doing something else than swapping (Surynek, 2018a). Each time edge $\{u,v\}$ is being traversed by token c_i between time steps t and $t+1$, a token residing in v at time step t, that is $\tau_t(v)$, must go in the opposite direction from v to u. If this is not the case, then a so called *edge collision* involving edge $\{u,v\}$ occurs and corresponding *edge conflicts* $(c_i,(u,v),t)$ and $(\tau_t(v),(v,u),t)$ are introduced for agents c_i and $\tau_t(v)$ respectively.

Edge conflicts must be treated at the low level of CBS. Hence in addition to forbidden vertices at given time-steps we have forbidden edges between given time-steps.

TROT: The treatment of conflicts will be complementary to TSWAP in TROT. Each time edge $\{u,v\}$ is being traversed by token c_i between time steps t and $t+1$, a token residing in v at time step t, that is $\tau_t(v)$, must go anywhere else but not to u. If this is not the case, then we again have edge collision $(c_i,\tau_t(v),\{u,v\},t)$ which is treated in the same way as above.

4.2.2 Encoding Changes in MDD-SAT

In MDD-SAT, we need to modify encoding of movement rules in the propositional formula $\mathcal{F}(\xi)$. Again, proofs of soundness of the following changes are omitted.

TPERM: This is the easiest case for MDD-SAT too. We merely remove all constrains requiring tokens to move into vacant vertices only. That is we remove clauses (3).

TSWAP: It inherits changes from TPERM but in addition to that we need to carry out swaps properly. For this edge variables $\mathcal{E}^t_{u,v}(c_i)$ will be utilized. Following constraint will be introduced for every $\{u^t,v^{t+1}\} \in E_i$ (intuitively, if token c_i traverses $\{u,v\}$ some other token c_j traverses $\{u,v\}$ in the opposite direction):

$$\mathcal{E}^t_{u,v}(c_i) \Rightarrow \bigvee_{j=1,2,\ldots,k | j\neq i \wedge (u^t,v^{t+1}) \in E_j} \mathcal{E}^t_{v,u}(c_j) \qquad (6)$$

TROT: TROT is treated in a complementary way to TSWAP. Instead of adding constraints (6) we add constraints forbidding simultaneous traversal in the opposite direction as follows:

[5] Formally this is the same as in MAPF, but in addition to this MAPF checks vacancy of the target vertex which may cause more colliding situations.

$$\mathcal{E}^t_{u,v}(c_i) \Rightarrow \bigwedge_{j=1,2,...,k|j\neq i\wedge(u^t,v^{t+1})\in E_j} \neg\mathcal{E}^t_{v,u}(c_j) \quad (7)$$

5 COMBINING SAT-BASED APPROACH AND CBS

Close look at CBS reveals that it operates similarly as problem solving in *satisfiability modulo theories* (SMT) (Bofill et al., 2012). SMT divides satisfiability problem in some complex theory T into an abstract propositional part that keeps the Boolean structure of the problem and simplified decision procedure $DECIDE_T$ that decides conjunctive formulae over T. A general T-formula is transformed to *propositional skeleton* by replacing atoms with propositional variables. The standard SAT-solving procedure then decides what variables should be assigned $TRUE$ in order to satisfy the skeleton - these variables tells what atoms holds in T. $DECIDE_T$ if the conjunction of satisfied atoms is satisfiable. If so then solution is returned. Otherwise conflict from $DECIDE_T$ is reported back and the skeleton is extended with a constraint forbidding the conflict.

The above observation let us to the idea to implement CBS in the SMT manner. The abstract propositional part working with the skeleton will be taken from MDD-SAT except that only constraints ensuring that assignments form valid paths interconnecting starting positions with goals will be preserved. Other constraints for collision avoidance will be omitted initially. Paths validation procedure will act as $DECIDE_T$ and will report back a set of conflicts found in the current solution. We call this algorithm SMT-CBS and it is shown in pseudo-code as Algorithm 3 (it is formulated for MAPF; but is applicable for TSWAP, TPERM, and TROT after replacing conflict resolution part).

The algorithm is divided into two procedures: SMT-CBS representing the main loop and SMT-CBS-Fixed solving the input MAPF for a fixed cost ξ. The major difference from the standard CBS is that there is no branching at the high level. The high level SMT-CBS rougly correspond to the main loop of MDD-SAT. The set of conflicts is iteratively collected during entire execution of the algorithm. Procedure *encode* from MDD-SAT is replaced with *encode-Basic* that produces encoding that ignores specific movement rules (collisions between agents) but on the other hand encodes collected conflicts into $\mathcal{F}(\xi)$.

The conflict resolution in standard CBS implemented as high-level branching is here represented by refinement of $\mathcal{F}(\xi)$ with disjunction (line 20). Branching is thus deferred into the SAT solver. The

Algorithm 3: SMT-CBS algorithm for solving MAPF.

```
1  SMT-CBS (Σ = (G = (V,E), A, α0, α+))
2      conflicts ← ∅
3      paths ← {shortest path from α0(ai) to
       α+(ai) | i = 1,2,...,k}
4      ξ ← Σ_{i=1}^{k} ξ(paths(ai))
5      while True do
6          (paths, conflicts) ←
           SMT-CBS-Fixed(conflicts, ξ, Σ)
7          if paths ≠ UNSAT then
8              return paths
9          ξ ← ξ + 1
10 SMT-CBS-Fixed(conflicts, ξ, Σ)
11     F(ξ)) ← encode-Basic(conflicts, ξ, Σ)
12     while True do
13         assignment ← consult-SAT-Solver(F(ξ))
14         if assignment ≠ UNSAT then
15             paths ← extract-Solution(assignment)
16             collisions ← validate(paths)
17             if collisions = ∅ then
18                 return (paths, conflicts)
19             for each (ai, aj, v, t) ∈ collisions do
20                 F(ξ) ← ¬X^t_v(ai) ∨ ¬X^t_v(aj)
21                 conflicts ←
                   conflicts ∪ {[(ai, v, t), (aj, v, t)]}
22         return (UNSAT, conflicts)
```

presented SMT-CBS process builds in fact equisatisfiable formula to that built by MDD-SAT. 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.

6 EXPERIMENTAL EVALUATION

We performed an extensive evaluation of all presented algorithms on standard synthetic benchmarks (Boyarski et al., 2015; Sharon et al., 2013). A representative part of results is presented in this section.

6.1 Benchmarks and Setup

We used the implementation of SMT-CBS in C++ on top of the Glucose 4 SAT solver (Audemard et al., 2013; Audemard and Simon, 2009) that ranks among the best SAT solvers according to recent SAT solver competitions (Balyo et al., 2017). The standard CBS has been re-implemented from scratch since the original implementation written in Java does support only grids but not general graphs (Sharon et al., 2015) that we need in our tests. Regarding MDD-SAT we used a version applicable on TSWAP, TPERM, and TROT

that is implemented in C++ (Surynek et al., 2016a). All experiments were run on a Ryzen 7 CPU 3.0 Ghz under Kubuntu linux 16 with 16GB RAM[6]

Figure 5: Example of 4-connected *grid*, *star*, *path*, and *clique*.

We divided the experimental evaluation into two categories of tests. The first part of experimental evaluation has been done on diverse instances consisting of **small** graphs: 4-connected *grid* of size 8×8 and 16×16, *random graphs* containing 20% of random edges, *star* graphs, *paths*, and *cliques* (see Figure 5). Initial and goal configurations of tokens/agents was set at random in all tests. We used a clique, a random graph, a path, and a star consisting of 16 vertices.

The second part of experimental evaluation took place on large 4-connected maps taken from *Dragon Age* (Sharon et al., 2015; Sturtevant, 2012) - three maps we used in our experiments are shown in Figure 7. In contrast to small instances, these were only sparsely populated with items. Initial and goal configuration were generated at random again.

We varied the number of items in relocation instances to obtain instances of various difficulties; that is, the underlying graph was not fully occupied - which in MAPF has natural meaning while in token problems we use one special color $\perp \in C$ that stands for any empty vertex (that is, we understand v as empty if and only if $\tau(v) = \perp$). For each number of items in the relocation instance we generated 10 random instances. For example, a *clique* consisting of 16 vertices gives 160 instances in total.

The timeout in all test was set to 60 seconds. Presented results were obtained from instances finished under this timeout.

6.2 Comparison on Small Graphs

Tests on small graphs were focused on the runtime comparison and evaluation of the size of encodings in case of MDD-SAT and SMT-CBS. Part of results we obtained is presented in Figures 6, 8, and 9 - mean runtime out of 10 random instances is reported per each number of items. Surprisingly instances turned out to be relatively hard even for small graphs.

In all tests CBS turned out to be uncompetitive against MDD-SAT and SMT-CBS on instances containing more agents. This is an expectable result as it is known that performance of CBS degrades on densely occupied instances (Surynek et al., 2016b). In the rest of experiments we omitted MDD-SAT.

[6]To enable reproducibility of presented results we will provide complete source codes and data on author's web: http://users.fit.cvut.cz/surynpav/research/icaart2019.

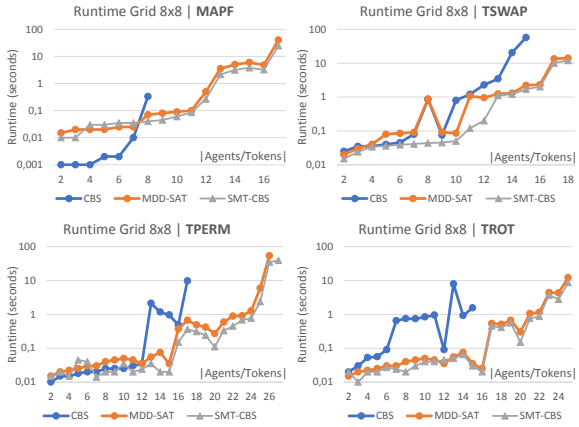

Figure 6: Runtime comparison of CBS, MDD-SAT, and SMT-CBS algorithms solving MAPF, TSWAP, TPERM, and TROT on 8×8 *grid*.

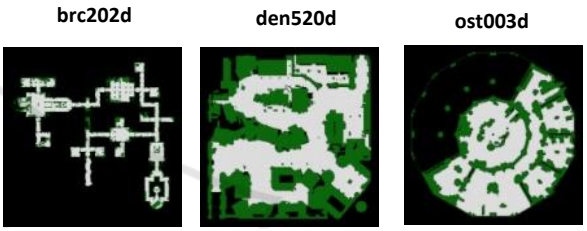

Figure 7: Three structurally diverse `Dragon-Age` maps used in the experimental evaluation. This selection includes: narrow corridors in `brc202d`, large open space in `den520d`, and open space with almost isolated rooms in `ost003d`.

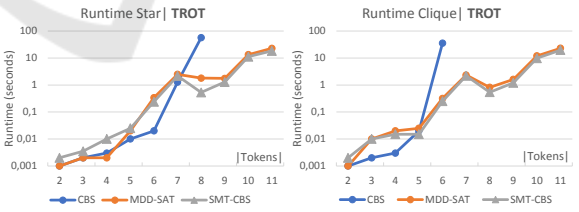

Figure 8: Comparison of TROT solving by CBS, MDD-SAT, and SMT-CBS on a *star* and *clique* graphs consisting of 16 vertices.

Figures 10 and 11 show sorted runtimes of CBS and SMT-CBS solving TROT on a *random graph* and a *star* and TSWAP on a *clique*, and a *path* all consisting of 16 vertices. In all cases, CBS clearly dominates in easier instances but its performance degrades faster as instances gets harder where SMT-CBS tends to dominate. Eventually SMT-CBS solved more out of 160 instances per test than CBS under the given timeout of 60 seconds. Path is particularly interesting

Number of generated clauses

|Agents|	4	8	12	16	20
MDD-SAT	556	56 652	1 347 469	3 087 838	2 124 941
SMT-CBS	468	31 973	598 241	1 256 757	803 671

Figure 9: Comparison of the size of encodings generated by MDD-SAT and SMT-CBS (number of clauses is shown) on MAPF instances.

case as the performance of CBS and SMT-CBS differs greatly there.

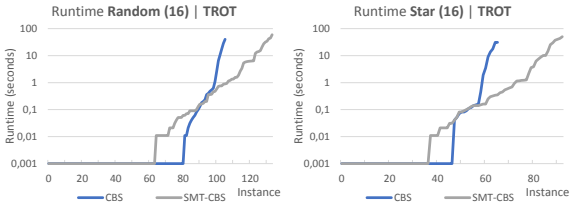

Figure 10: Sorted runtimes of CBS and SMT-CBS solving TROT on *random* and *star* graphs consisting of 16 vertices.

Solving of all types of relocation problems on the same type of graph - 8×8 grid in this case - is shown in Figure 12. A different pattern can be observed in these results, SMT-CBS dominates across all difficulties of instances over CBS. Grids contained up to 40 items, so having 10 random instances per number of agents, we had 400 instances in total, but only about 250 were solved under 60 seconds in the case of TPERM problem using SMT-CBS.

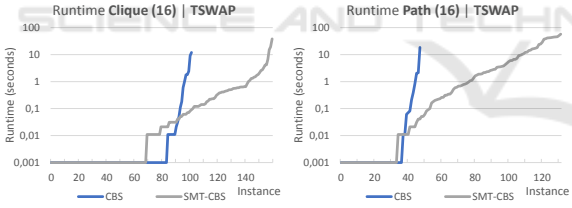

Figure 11: Sorted runtimes of CBS and SMT-CBS solving TSWAP on *clique* and *path* graphs consisting of 16 vertices.

SMT-CBS turned out to be fastest in performed tests on small graphs. SMT-CBS reduces the runtime by about 30% to 50% relatively to MDD-SAT. More significant benefit of SMT-CBS was observed in MAPF and TSWAP while in TROT and TPERM the improvement was less significant. Both MAPF and TSWAP have more clauses in their eagerly-generated encodings by MDD-SAT than TROT and TPERM hence SMT-CBS has greater room for reducing the size of encoding by constructing it lazily in these types of relocation problems. This claim has been experimentally verified (Figure 9); the SMT-CBS reduces the number of clauses to less than half of the number generated by MDD-SAT.

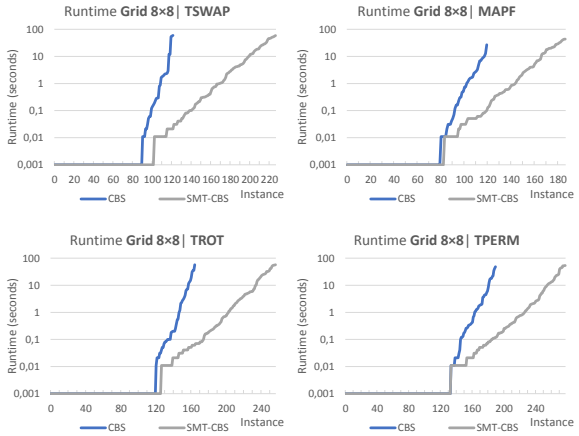

Figure 12: Sorted runtimes of CBS and SMT-CBS solving MAPF, TSWAP, TPERM, and TROT on 8×8 *grid*.

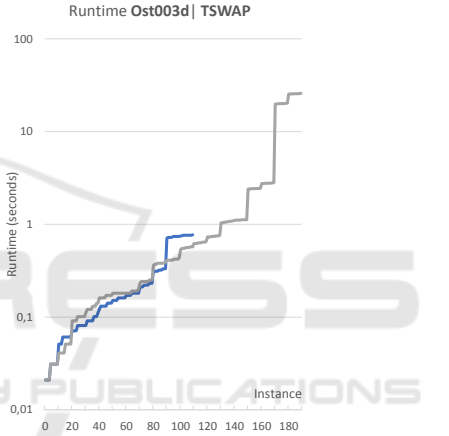

Figure 13: Sorted runtimes of CBS and SMT-CBS solving TSWAP on `ost003d`.

6.3 Evaluation on Large Maps

The second category of tests was focused on the performance of CBS and SMT-CBS on large maps (experimenting with MDD-SAT was omitted). In the three structurally different maps, up to 32 items were placed randomly. Again we had 10 random instances per each number of items.

Sorted runtimes are reported in Figures 13 and 14. Completely different picture can be seen here. CBS is faster than SMT-CBS across all difficulties of instances over `brc202d` and `den520d`. In the case of `ost003d` we can see CBS and SMT-CBS performing similarly in easier instances but eventually CBS won in hard instances containing more items.

A deeper analysis of runtimes revealed that whenever CBS has a chance to search for a long conflict free path it can outperform SMT-CBS. On the other hand if conflict handling due to intensive interaction

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

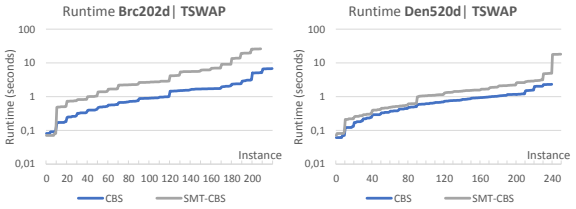

Figure 14: Sorted runtimes of CBS and SMT-CBS solving TSWAP on `brc202d` and `den520d`.

among items prevails then SMT-CBS tends to dominate which usually takes place in small graphs.

7 CONCLUSIONS

We studied a general framework for reasoning about conflicts in item relocation problems in graphs based on concepts from the CBS algorithm. In addition to two known problems MAPF and TSWAP, we studied two derived variants TROT and TPERM. The experimental evaluation of CBS, MDD-SAT, and SMT-CBS showed that SMT-CBS outperforms both CBS and MDD-SAT on instances in small graphs. But we have also shown that there is no universal winner as CBS turned out to be faster on large maps.

The most significant benefit of SMT-CBS can be observed on highly constrained MAPF and TSWAP instances. The search for long paths with few conflicts is, on the other hand, the performance bottleneck of SMT-CBS. For future work we plan to further reduce the size of SAT encodings in SMT-CBS by eliminating unnecessary time expansions in MDDs and improve the search for long paths. We also would like improve the performance of all implemented algorithms to be able to observe their behavior on large more densely occupied instances so far a missing case in our experiments.

ACKNOWLEDGEMENTS

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

REFERENCES

Audemard, G., Lagniez, J., and Simon, L. (2013). Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pages 309–317.

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

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

Basile, F., Chiacchio, P., and Coppola, J. (2012). A hybrid model of complex automated warehouse systems - part I: modeling and simulation. *IEEE Trans. Automation Science and Engineering*, 9(4):640–653.

Biere, A., Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of Satisfiability*. IOS Press.

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

Bonnet, É., Miltzow, T., and Rzazewski, P. (2017). Complexity of token swapping and its variants. In *STACS 2017*, volume 66 of *LIPIcs*, pages 16:1–16:14.

Botea, A., Kishimoto, A., and Marinescu, R. (2018). On the complexity of quantum circuit compilation. In *Proceedings of SOCS 2018*, pages 138–142.

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

de Wilde, B., ter Mors, A., and Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *JAIR*, 51:443–492.

Dresner, K. and Stone, P. (2008). A multiagent approach to autonomous intersection management. *JAIR*, 31:591–656.

Kapadia, M., Ninomiya, K., Shoulson, A., Garcia, F. M., and Badler, N. I. (2013). Constraint-aware navigation in dynamic environments. In *Motion in Games, MIG '13*, pages 111–120. ACM.

Kawahara, J., Saitoh, T., and Yoshinaka, R. (2017). The time complexity of the token swapping problem and its parallel variants. In *WALCOM 2017*, volume 10167 of *LNCS*, pages 448–459. Springer.

Kim, D.-G., Hirayama, K., and Park, G.-K. (2014). Collision avoidance in multiple-ship situations by distributed local search. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 18:839–848.

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

Luna, R. and Bekris, K. (2011a). Efficient and complete centralized multi-robot path planning. In *IROS*, pages 3268–3275.

Luna, R. and Bekris, K. E. (2010). Network-guided multi-robot path planning in discrete representations. In *IROS*, pages 4596–4602.

Luna, R. and Bekris, K. E. (2011b). Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300.

Miltzow, T., Narins, L., Okamoto, Y., Rote, G., Thomas, A., and Uno, T. (2016). Approximation and hardness of

token swapping. In *ESA 2016*, volume 57 of *LIPIcs*, pages 66:1–66:15.

Ratner, D. and Warmuth, M. K. (1986). Finding a shortest solution for the N × N extension of the 15-puzzle is intractable. In *AAAI*, pages 168–172.

Ryan, M. R. K. (2007). Graph decomposition for efficient multi-robot path planning. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2003–2008.

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

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

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

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

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

Standley, T. (2010). Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, pages 173–178.

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

Surynek, P. (2009a). An application of pebble motion on graphs to abstract multi-robot path planning. In *ICTAI 2009*, pages 151–158.

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

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

Surynek, P. (2012a). On propositional encodings of cooperative path-finding. In *ICTAI 2012*, pages 524–531. IEEE Computer Society.

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

Surynek, P. (2014a). Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *ICTAI*, pages 875–882.

Surynek, P. (2014b). Solving abstract cooperative path-finding in densely populated environments. *Computational Intelligence*, 30(2):402–450.

Surynek, P. (2017). Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. *Ann. Math. Artif. Intell.*, 81(3-4):329–375.

Surynek, P. (2018a). Finding optimal solutions to token swapping by conflict-based search and reduction to SAT. In *Proceedings of ICTAI 2018*, pages 592–599.

Surynek, P. (2018b). Lazy modeling of variants of token swapping problem and multi-agent path finding through combination of satisfiability modulo theories and conflict-based search. *CoRR*, abs/1809.05959.

Surynek, P., Felner, A., Stern, R., and Boyarski, E. (2016a). Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, pages 810–818.

Surynek, P., Felner, A., Stern, R., and Boyarski, E. (2016b). An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives. In *SoCS 2016*.

Thorup, M. (2002). Randomized sorting in o(n log log n) time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42(2):205–230.

Wender, S. and Watson, I. D. (2014). Combining case-based reasoning and reinforcement learning for unit navigation in real-time strategy game AI. In *ICCBR*, volume 8765 of *LNCS*, pages 511–525. Springer.

Wilson, R. M. (1974). Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86 – 96.

Yamanaka, K., Demaine, E. D., Horiyama, T., Kawamura, A., Nakano, S., Okamoto, Y., Saitoh, T., Suzuki, A., Uehara, R., and Uno, T. (2017). Sequentially swapping colored tokens on graphs. In *WALCOM 2017*, volume 10167 of *LNCS*, pages 435–447. Springer.

Yamanaka, K., Demaine, E. D., Ito, T., Kawahara, J., Kiyomi, M., Okamoto, Y., Saitoh, T., Suzuki, A., Uchizawa, K., and Uno, T. (2014). Swapping labeled tokens on graphs. In *FUN 2014 Proceedings*, volume 8496 of *LNCS*, pages 364–375. Springer.

Yamanaka, K., Demaine, E. D., Ito, T., Kawahara, J., Kiyomi, M., Okamoto, Y., Saitoh, T., Suzuki, A., Uchizawa, K., and Uno, T. (2015). Swapping labeled tokens on graphs. *Theor. Comput. Sci.*, 586:81–94.

Yamanaka, K., Horiyama, T., Kirkpatrick, D., Otachi, Y., Saitoh, T., Uehara, R., and Uno, Y. (2016). Computational complexity of colored token swapping problem. In *IPSJ SIG Technical Report*, volume 156.

Yu, J. and LaValle, S. M. (2013). Planning optimal paths for multiple robots on graphs. In *ICRA 2013*, pages 3612–3617.

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

Yu, J. and LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Trans. Robotics*, 32(5):1163–1177.

Yu, J. and Rus, D. (2014). Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *WAFR 2014*, pages 729–746.

Zhou, D. and Schwager, M. (2015). Virtual rigid bodies for coordinated agile maneuvering of teams of micro aerial vehicles. In *ICRA 2015*, pages 1737–1742.