Multi-Agent Path Finding with Generalized Conflicts: an experimental study*

Pavel Surynek[0000−0001−7200−0542]

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

Abstract. This paper gives an overview of conflict reasoning in generalizations of multi-agent path finding (MAPF). MAPF and derived variants 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 relocation problem must be satisfied. We recall a general problem formulation that encompasses known types of item relocation problems such as multi-agent path finding (MAPF), token swapping (TSWAP), token rotation (TROT), and token permutation (TPERM). We then focused on three existing optimal algorithms for MAPF: search-based CBS, and propositional satisfiability (SAT) -based MDD-SAT and SMT-CBS. These algorithms were modified to tackle various types of conflicts. The major contribution of this paper is a thorough experimental evaluation of CBS, MDD-SAT, and SMT-CBS on various types of relocation problems.

Keywords: conflicts, MAPF, token swapping, token rotation, token permutation, SMT, SAT

1 Introduction

Item relocation problems in graphs such as *token swapping* (TSWAP) [13,7], *multi-agent path finding* (MAPF) [22,28,53], or pebble motion on graphs (PMG) [47,15] represent important combinatorial problems in artificial intelligence with specific applications in coordination of multiple robots and other areas such as quantum circuit compilation [8]. Graphs in item relocation problems may be directly derived from the physical environment where items move but can be also represented by abstract spaces like *configuration spaces* in robotics [52].

Distinguishable items placed in vertices of an undirected graph such that at most one item is placed in each vertex. Items can be moved 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. TSWAP on the other hand permits only swaps of pairs of tokens along edges

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

while more complex movements involving more than two tokens are forbidden. The task in item relocation problems is to reach a given goal configuration from a given starting configuration.

We focus here on the optimal solving of item relocation problems with respect to common objectives. Two cumulative objective functions are used in MAPF and TSWAP - *sum-of-costs* [25,19] and *makespan* [35,55]. The sum-of-costs corresponds to the total cost of all movements performed. The makespan corresponds to the total number of time-steps until the goal is reached. We trying to minimize the objective in both cases.

Many practical problems from robotics involving multiple robots can be interpreted as an item relocation problems. Examples include discrete multi-robot navigation and coordination [18], item rearrangement in automated warehouses [4], ship collision avoidance [14], or formation maintenance and maneuvering of aerial vehicles [57]. Examples not only include problems concerning physical items but problems occurring in virtual spaces of simulations [12], computer games [45], or quantum systems [8].

The contribution of this paper consists in an experimental evaluation of a general framework for defining and solving item relocation problems based on *satisfiability modulo theories* (SMT) [6,40] and *conflict-based search* (CBS) [24].

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 modify existing algorithms for various variants of item relocation problems. We will adapt the standard *conflict-based search* (CBS) but also propositional satisfiability (SAT) - based MDD-SAT [42] and recent SMT-based SMT-CBS [40].

This work originally appeared as a conference paper [41]. In this revised version we provide more thorough experimental study of concepts presented in the original conference paper. We first introduce TSWAP and MAPF. Then prerequisites for conflict handling formulated in the SMT framework are recalled. On top of this, the combination of CBS and MDD-SAT is developed - the SMT-CBS algorithm. Finally, a thorough experimental evaluation of CBS, MDD-SAT, and SMT-CBS on various benchmarks including both small and large instances is presented.

2 Background

Multi-agent path finding (MAPF) problem [27,23] 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 $\alpha : A \to V$. 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.

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*) [49] is similar to MAPF. It represents a generalization of sorting problems [44]. 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 are allowed to swap elements at selected pairs of positions only.

Using a modified notation from [50] 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.

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 [50,48]:

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 $\tau(w) = \tau'(w)$.* [1]

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 $\mathcal{O}(|V|^2)$ swaps [51]. 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.

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

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.

3 Related Work

Although many works sudying TSWAP from the theoretical point of view exist [51,19,7] practical solving of the problem started only lately. In [39] optimal solving of TSWAP by adapted algorithms from MAPF has been suggested. Namely *conflict-based search* (CBS) [26,24] and *propositional satisfiability-based* (SAT) [5] MDD-SAT [42,43] 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.

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 [10,28,25,24]. 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.* [2]

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 yields the cost of 2 as two tokens traverses single edge.

A feasible solution of a solvable MAPF instance can be found in polynomial time [47,15]; precisely the worst case time complexity of most practical algorithms for finding feasible solutions is $\mathcal{O}(|V|^3)$ (asymptotic size of the solution is also $\mathcal{O}(|V|^3)$) [32,31,37,17,16,46]. 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 [21,33,54].

3.2 Conflict-based Search

CBS uses the idea of resolving conflicts lazily; that is, a solution is searched against an incomplete set of movement constraints hoping a valid solution can be found 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

[2] 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.

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 [24].

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.

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.

3.3 SAT-based Approach

An alternative approach to optimal MAPF solving is represented by the reduction of MAPF to propositional satisfiability (SAT) [30,34]. 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.

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 common cummulative objectives such as the 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 $\mathcal{F}(\xi)$ [1].

Construction of $\mathcal{F}(\xi)$ relies on time expansion of underlying graph G [38]. 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

Algorithm 1: Basic CBS algorithm for MAPF solving [41]

```
1  CBS (G = (V, E), A, α_0, α_+)
2      R.constraints ← ∅
3      R.paths ← {shortest path from α_0(a_i) to α_+(a_i)|i = 1, 2, ..., k}
4      R.ξ ← Σ_{i=1}^{k} ξ(N.paths(a_i))
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 (a_i, a_j, v, t) ∈ collisions
13         for each a ∈ {a_i, a_j} 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(a_i))
18             insert N' into OPEN
```

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 $\alpha_+^\mu(a_i)$.

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 [42]. 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 [41]:

Algorithm 2: Framework of SAT-based MAPF solving [41]

```
SAT-Based (G = (V, E), A, α_0, α_+)
    paths ← {shortest path from α_0(a_i) to α_+(a_i) | i = 1, 2, ..., k}
    ξ ← Σ_{i=1}^{k} ξ(N.paths(a_i))
    while True do
        F(ξ) ← encode(ξ, G, A, α_0, α_+)
        assignment ← consult-SAT-Solver(F(ξ))
        if assignment ≠ UNSAT then
            paths ← extract-Solution(assignment)
            return paths
        ξ ← ξ + 1
```

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

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

Similarly, the target vertex of any movement except wait action must be empty. This is ensured by the following constraint for every $(u^t, v^{t+1}) \in E_i$ [41]:

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

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 [41]:

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

Agents do not collide with each other; the following constraint is introduced for every $v \in V$ and timestep t [41]:

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

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) [25]. 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.

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.

The combination of SAT-based approach and MDD time expansion led to the MDD-SAT algorithm described in [42] that currently represent state-of-the-art in SAT-based MAPF solving.

4 Generalizations of Item Relocation

We define two problems derived from MAPF and TSWAP: *token rotation* (TROT) and *token permutation* (TPERM) [3].

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 analogous 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.

[3] These problems have been considered in the literature in different contexts already (for example in [56]). But not from the practical solving perspective focused on finding optimal solutions.

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.

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.

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 $t + 1$.

Conflict reasoning in individual item relocation problems follows.

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. [4]

TSWAP: This problem takes conflicts from TPERM but adds new conflicts that arise from doing something else than swapping [39]. 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 $(c_i, \tau_t(v),\{\mathrm{u,v}\}, t)$ which is treated in the same way as above.

[4] 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.

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) [41]:

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

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 [41]:

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

5 Combining the SAT-based Approach and CBS

A close look at CBS reveals that it works similarly as problem solving in *satisfiability modulo theories* (SMT) [6,20]. 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 only conjunctive formulae over T. A general T-formula is transformed to *propositional skeleton* by replacing atoms with propositional variables. The 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. $DECIDE_T$ checks if the conjunction of selected (satisfied) atoms is satisfiable. If so then solution is returned. Otherwise a conflict from $DECIDE_T$ is reported back and the skeleton is extended with a constraint that eliminates the conflict.

Following the above observation we rephrased CBS in the SMT manner. The abstract propositional part working with the skeleton was taken from MDD-SAT except that only constraints ensuring that assignments form valid paths interconnecting starting positions with goals will be preserved. Other constraints, namely those ensuring collision avoidance between items 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 (here is a difference from the SMT-style solving that reports only one conflict while here we take all conflicts). We call the new 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).

Algorithm 3: SMT-CBS algorithm for solving MAPF [41]

```
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_v^t(ai) ∨ ¬X_v^t(aj)
21                 conflicts ← conflicts ∪ {[(ai, v, t), (aj, v, t)]}
22         return (UNSAT,conflicts)
```

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 roughly corresponds 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 the standard CBS implemented as high-level branching is here represented by refinement of $\mathcal{F}(\xi)$ with a disjunction (line 20). Branching is thus deferred into the SAT solver. The advantage of SMT-CBS is that it builds the formula lazily; that is, it adds constraints on demand after a conflict occurs. Such approach may save resources as solution may be found earlier than all constraints are added. In contrast to this, the eager approach of MDD-SAT first adds all constraints and then solves the complete formula.

6 Experimental Evaluation

To evaluate how different conflicts affect the performance of solving algorithms we performed an extensive evaluation of all presented algorithms on both standard synthetic benchmarks [9,25] and large maps from games [29]. A representative part of results is presented in this section.

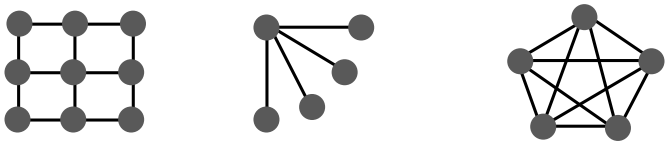

Fig. 1. Example of *4-connected grid*, *star*, *path*, and *clique* [41].

6.1 Benchmarks and Setup

We implemented the SMT-CBS algorithm in C++ on top of the Glucose 4 SAT solver [1,2]. The choice of Glucose 4 is given by the fact that it ranks among the best SAT solvers according to recent SAT solver competitions [3]. The standard CBS has been re-implemented from scratch since the original implementation written in Java does support only grids but not general graphs [24] that we need in our tests.

Regarding MDD-SAT we used an existing implementation in C++ [42]. The original MDD-SAT has been developed for MAPF but versions applicable on TSWAP, TROT, and TPERM are implemented in the existing package as well. All experiments were run on a Ryzen 7 CPU 3.0 Ghz under Kubuntu linux 16 with 16GB RAM[5].

Our experimental evaluation is divided in three parts. The first part of experimental evaluation has been done on diverse instances consisting of **small** graphs: *random graphs* containing 20% of random edges, *star* graphs, *paths*, and *cliques* (see Figure 1). The initial and the goal configuration of tokens/agents was set at random in all tests. The size of the set of vertices of clique, random graph, path, and star graphs was 16 . Small graphs were densely populated with tokes/agents. Instances containing up to 40 tokens/agents in grids of size 8×8 and up to 64 tokens/agents in grids of size 16×16 were generated. All generated instances were solvable but not all of them could be solved under the given timeout.

[5] To enable reproducibility of results presented in this paper we provide complete source codes and experimental data on author's web page: http://users.fit.cvut.cz/surynpav/research/icaart2019revised.

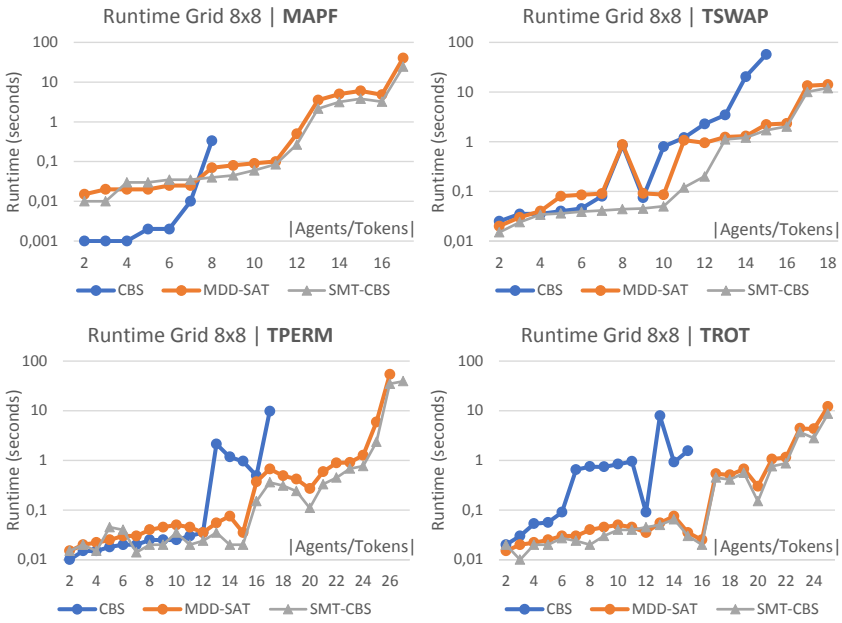

Fig. 2. Runtime comparison of CBS, MDD-SAT, and SMT-CBS algorithms solving MAPF, TSWAP, TPERM, and TROT on 8×8 *grid* [41].

The second part of experiments has been done on medium-sized graphs - *4-connected open grids* of size 8×8 and 16×16. This is the standard benchmark being used for evaluation of MAPF algorithms [11].

And finally the third part of experimental evaluation took place on large 4-connected maps taken from *Dragon Age* [24,29] - three maps we used in our experiments are shown in Figure 3. These are structurally different maps focusing on various aspects such as narrow corridors, large almost isolated rooms, or topologically complex open space. 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 was set to 60 seconds in the series of tests comparing the performance of CBS, MDD-SAT, and, SMT-CBS with respect to the growing number of items. The next series of large scale tests comparing the performance of CBS

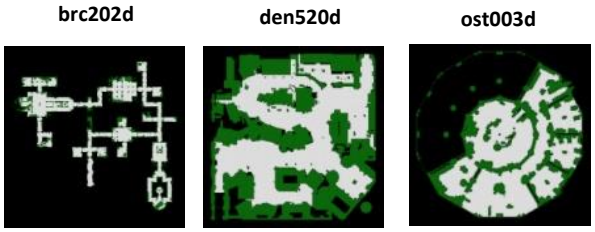

Fig. 3. Three structurally diverse `Dragon-Age` maps used in the experimental evaluation. This selection includes: narrow corridors in `brc202d`, large topologically complex space in `den520d`, and open space with large almost isolated rooms in `ost003d` [41].

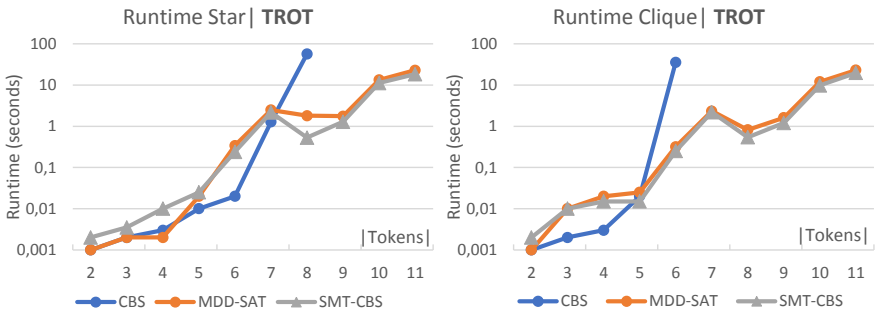

Fig. 4. Comparison of TROT solving by CBS, MDD-SAT, and SMT-CBS on a *star* and *clique* graphs consisting of 16 vertices [41].

and SMT-CBS with respect to the growing difficulty of instances used the timeout of 1000 seconds (sorted runtimes are compared). All presented results were obtained from instances finished under the given timeout.

6.2 Comparison on Small Graphs

Tests on small graphs were focused on the runtime comparison and the evaluation of the size of encodings in case of MDD-SAT and SMT-CBS. Part of results we obtained is presented in Figures 2, 4, and 5. The mean runtime out of 10 random instances is reported per each number of items. Surprisingly we can see in 4 that instances are relatively hard even for small graphs. For CBS the runtime quickly grows with the increasing number of items. The runtime growth is slower in case of MDD-SAT and SMT-CBS but even these algorithms are not fast enough to solve all instances under the given timeout (only instances with up to 11 items were solved).

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 quickly on densely occupied instances [43].

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

Fig. 5. Comparison of the size of encodings generated by MDD-SAT and SMT-CBS (number of clauses is shown) on MAPF instances [41].

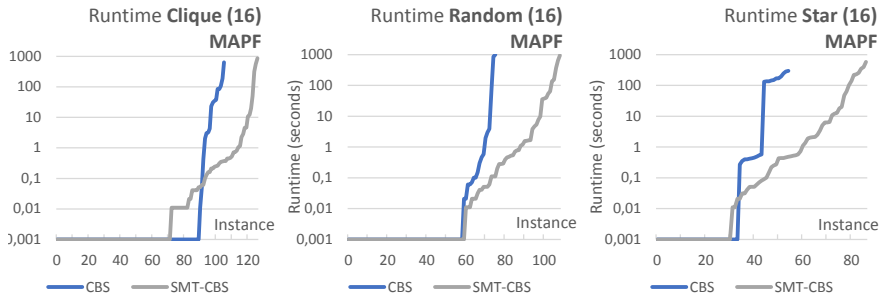

Fig. 6. Sorted runtimes of CBS and SMT-CBS solving MAPF on *clique*, *random*, and *star* graphs consisting of 16 vertices.

If we focus on the number of clauses generated by SAT-based solvers MDD-SAT and SMT-CBS we can see that MAPF, TSWAP and TROT have more clauses in their eagerly-generated encodings by MDD-SAT than TPERM hence SMT-CBS has greater room for reducing the size of the encoding by constructing it lazily in these types of relocation problems.

Experiments indicate that using SMT-CBS generally leads to reduction of the size of encoding to less than half of the original size generated by MDD-SAT in case of MAPF, TSWAP, and TROT. Results concerning this claim for MAPF are shown in Figure 5. The number of clauses for 4-connected grids are analyzed in the next section.

In the rest of runtime experiments that are focused on large scale evaluation we omitted MDD-SAT.

Figures 6, 7, 8, and 9 show sorted runtimes of CBS and SMT-CBS solving MAPF, TSWAP, TROT, and TPERM on all types of small graphs: *clique*, *path*, *random graph*, a *star* all consisting of 16 vertices (some combinations were not applicable: MAPF is typically unsolvable on *path*; TROT is trivial on *clique* but unsolvable on *path*; and TPERM is also trivial on *clique*).

The general trend is that 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 1000 seconds.

Some cases are particularly interesting as they point to the role of the structure of the underlying graph in the difficulty of instance. CBS significantly outperforms SMT-CBS on easy instances over *cliques* (and generally highly con-

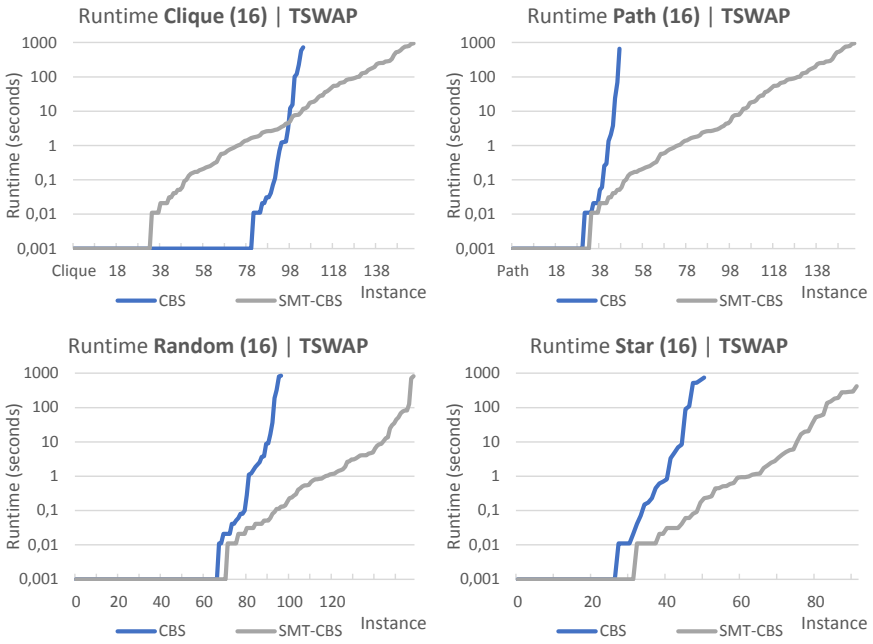

Fig. 7. Sorted runtimes of CBS and SMT-CBS solving TSWAP on *clique*, *path*, *random*, and *star* graphs consisting of 16 vertices.

nected graphs). Instances over *paths* seem to yield biggest difference in performance of CBS and SMT-CBS, CBS looses very quickly here.

Interesting results can be seen on *star* graphs. The growth of the runtime across sorted instances looks step-wise here (especially in TPERM - Figure 9). The interpretation is that adding an item into the graph causes sharp increase in the runtime (a step consists of 10 instances of roughly similar difficulty).

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.

6.3 Comparison on 4-connected Grids

Solving of all types of relocation problems on girds - the 8×8 grid and the 16×16 grid - is shown in Figure 10 and 11. A different pattern can be observed in these results, SMT-CBS dominates across all difficulties of instances over CBS.

The 8×8 grid contained up to 40 items, so having 10 random instances per number of agents, we had 400 instances in total, but only about 300 were solvable under 1000 seconds in the case of TPERM problem using SMT-CBS. In

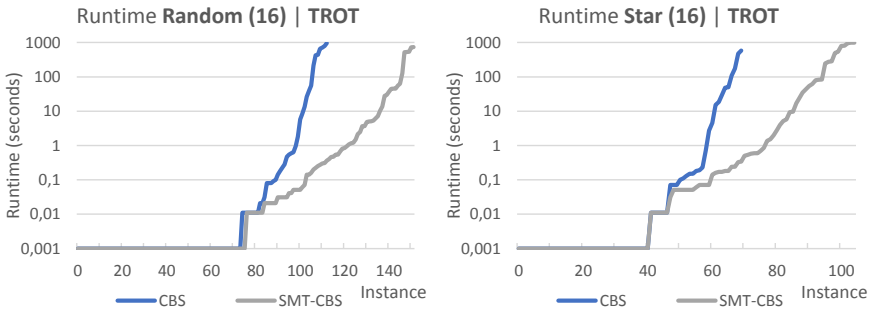

Fig. 8. Sorted runtimes of CBS and SMT-CBS solving TROT on *random* and *star* graphs consisting of 16 vertices.

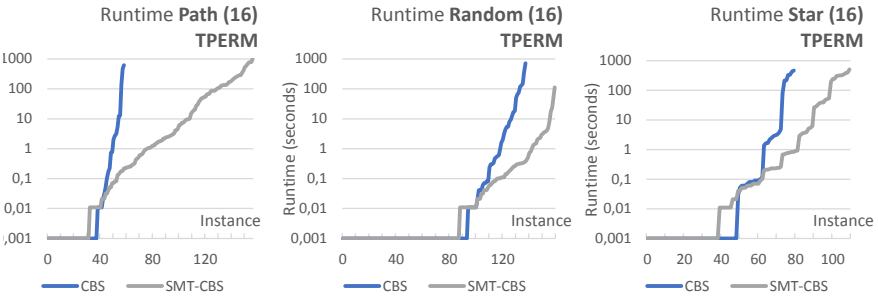

Fig. 9. Sorted runtimes of CBS and SMT-CBS solving TPERM on *random* and *star* graphs consisting of 16 vertices.

the 16×16 grid we had up to 64 items yielding to 640 instances in total. Almost all were solvable in the case of TPERM by SMT-CBS.

On the 16×16 grid the dominance of SMT-CBS seems to be pronounced. The general observation from this trend is that the difficulty on instances for CBS grows faster with every new item on larger maps than in smaller maps (disregarding the region where the performance of CBS and SMT-CBS is roughly the same). This observation complements results for large maps where we will see even bigger difference in difficulty growth.

In addition to runtime experiments we measured the number of generated clauses for instances on the 8×8 grid and the 16×16 grid. Results are presented in Figures 12 and 13.

We can observe that there is a small difference in the number of generated clauses between MDD-SAT and SMT-CBS on the TPERM problem. This can be attributed to the fact that TPERM is the least constrained version of relocation problems hence conflicts here arise less frequently and are simpler to express than in other more constrained versions. In other words, an encoding forbidding no conflict is of similar size as that eagerly forbidding all conflicts.

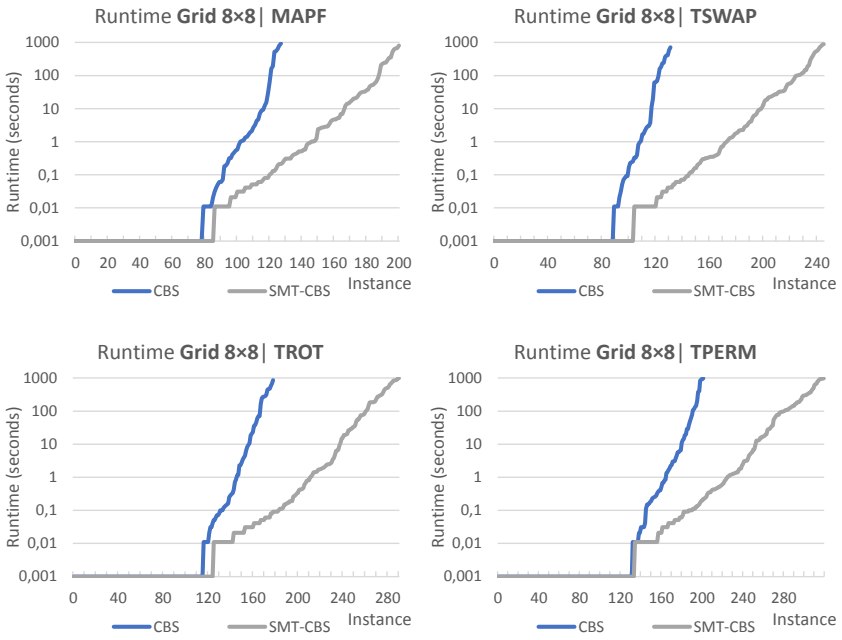

Fig. 10. Sorted runtimes of CBS and SMT-CBS solving MAPF, TSWAP, TPERM, and TROT on the 8×8 *grid* [41].

In MAPF, TSWAP, and TROT we can observe on both the 8×8 grid and the 16×16 grid that the difference in the size of eagerly generated encoding and lazily generated encoding grows in instances containing more agents. The reduction to about half of the size of the encoding generated by MDD-SAT can be achieved by SMT-CBS in sparsely occupied instances. But the difference is up to the factor of 10 in densely occupied instances.

6.4 Evaluation on Large Maps

The final category of tests was focused on the performance of CBS and SMT-CBS on large maps (experimenting with MDD-SAT was omitted here). In the three structurally different maps, up to 64 items were placed randomly. Again we had 10 random instances per each number of items.

Sorted runtimes are reported for each individual map and each version of relocation problem in Figures 14, 15, 16, and 17. Somewhat different picture can be seen here in comparison with experiments on small graphs. We attribute the different picture to the fact that we observe the problem in a different scale.

CBS shows its advantage over SMT-CBS across large set of easier instances where these correspond to instances containing fewer items. Eventually however SMT-CBS wins since the runtime of SMT-CBS goes quickly up when instances get more difficult. This is quite expectable from the theoretical properties of

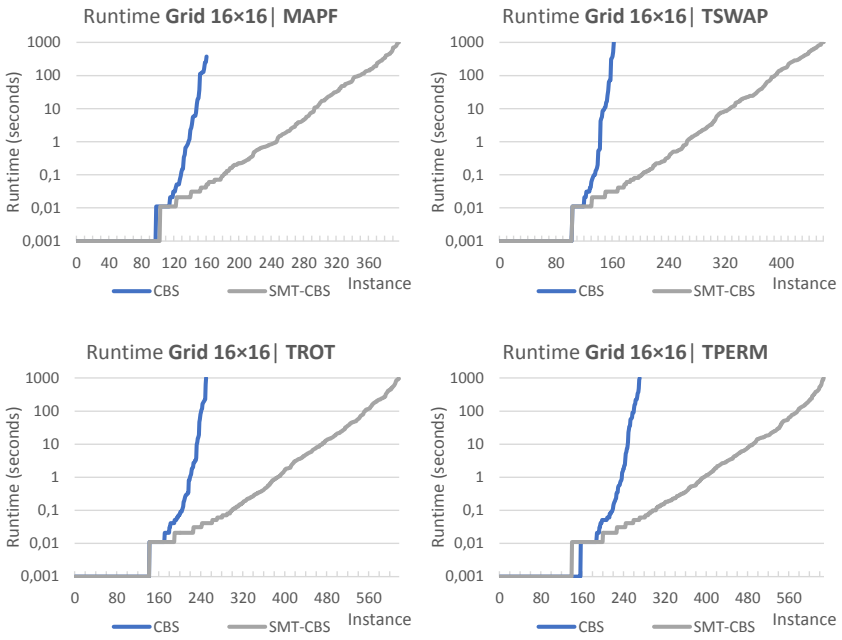

Fig. 11. Sorted runtimes of CBS and SMT-CBS solving MAPF, TSWAP, TPERM, and TROT on the 16×16 *grid*.

CBS and SMT-CBS. In instances with few items, CBS mostly searches for single source shortest paths while not needing to handle conflicts frequently. This is easier than building a SAT instance for the same problem. The situation changes when CBS must handle frequent conflicts between items in more densely occupied instances. Here viewing the problem as SAT and handling many conflicts in SAT as done by SMT-CBS seems to be more efficient than handling conflicts via branching the search at the high level in CBS.

MAPF and TSWAP are relatively more constrained than TROT and TPERM while TPERM is the least constrained version of item relocation. This property is clearly reflected in the line with the above observation in experiments. We can see that in less constrained cases CBS performs better than SMT-CBS for larger set of instances. Especially it is observable in TROT and TPERM solving on the `brc202d` map.

The overall analysis of runtimes can be summarized into the observation 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 among items prevails then SMT-CBS tends to dominate.

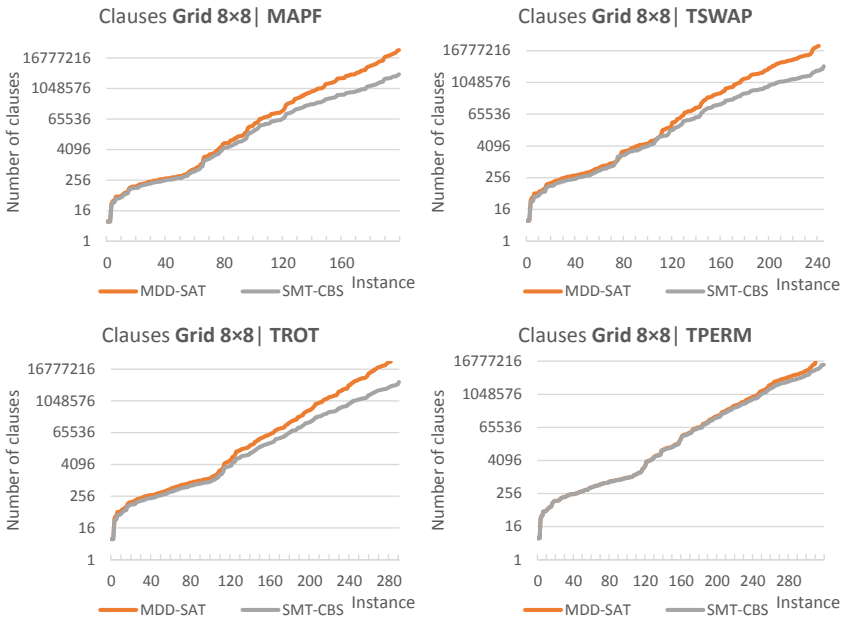

Fig. 12. The number of clauses generated by MDD-SAT and SMT-CBS when solving MAPF, TSWAP, TPERM, and TROT on the 8×8 *grid*.

7 CONCLUSIONS

This paper summarizes a general framework for reasoning about conflicts in item relocation problems in graphs based on concepts from the CBS algorithm. Different types of conflicts in four versions of relocation problems derived from multi-agent path finding (MAPF) are studied. In addition to two well studied problems MAPF and TSWAP, we also cover two derived variants TROT and TPERM. We presented thorough experimental evaluation of conflict handling in CBS, MDD-SAT and novel algorithm SMT-CBS that combines CBS and SAT-based reasoning from MDD-SAT. The experimental evaluation has been focused on runtime comparison as well as on the size of generated SAT encodings.

Experiments with CBS, MDD-SAT, and SMT-CBS showed that SMT-CBS outperforms both CBS and MDD-SAT on harder instances in all types of graphs. The most significant benefit of SMT-CBS can be observed on highly constrained MAPF and TSWAP instances where disjunctive conflict elimination is intensively used. The CBS algorithm on the other hand suffers from steep growth of the runtime in instances containing more items because it has to eliminate many conflicts through branching at the high level. This observation can be made across all individual types of relocation problem. The search for long paths with few conflicts is, on the other hand, the performance bottleneck of SMT-CBS. Hence in easier instances CBS is usually the fastest option.

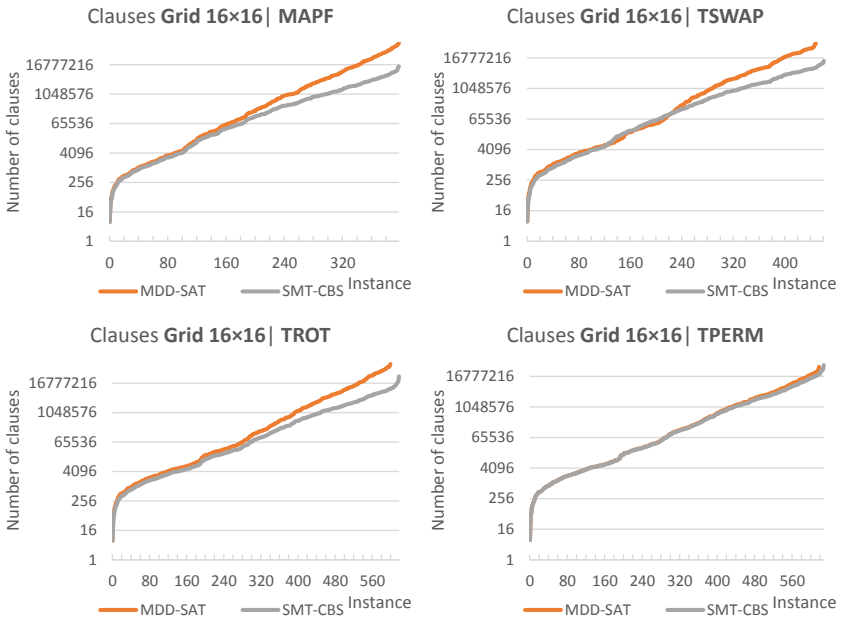

Fig. 13. The number of clauses generated by MDD-SAT and SMT-CBS when solving MAPF, TSWAP, TPERM, and TROT on the 16×16 *grid*.

MDD-SAT placed in the middle between CBS and SMT-CBS. The performance of MDD-SAT almost copies that of SMT-CBS though it is worse approximately by a factor of 2.0.

For the future work we plan to revise SAT encodings used in SMT-CBS and perform relevant experiments. Variables $\mathcal{E}_u, v^t(a_i)$ are auxiliary in fact as they can be derived from $\mathcal{X}_v^t(a_i)$. Hence we plan to make experiments with modified encodings where $\mathcal{E}_u, v^t(a_i)$ variables will not be used. This attempt is inspired by the DIRECT encoding [36] that was the first MAPF encoding relying on only $\mathcal{X}_v^t(a_i)$ variables.

References

1. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: SAT. pp. 309–317 (2013)
2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. pp. 399–404 (2009)
3. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: Recent developments. In: AAAI. pp. 5061–5063 (2017)
4. Basile, F., Chiacchio, P., Coppola, J.: A hybrid model of complex automated warehouse systems - part I: modeling and simulation. IEEE Trans. Automation Science and Engineering **9**(4), 640–653 (2012)

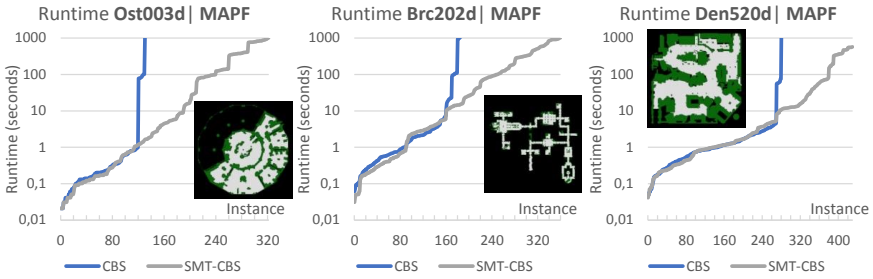

Fig. 14. Sorted runtimes of CBS and SMT-CBS solving MAPF on `ost003d`, `brc202d`, and `den520d` maps.

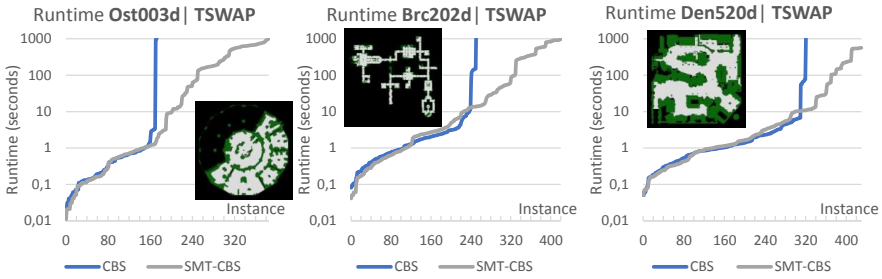

Fig. 15. Sorted runtimes of CBS and SMT-CBS solving TSWAP on `ost003d`, `brc202d`, and `den520d` maps.

5. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
6. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3), 273–303 (2012)
7. Bonnet, É., Miltzow, T., Rzazewski, P.: Complexity of token swapping and its variants. In: STACS 2017. LIPIcs, vol. 66, pp. 16:1–16:14. Schloss Dagstuhl (2017)
8. Botea, A., Kishimoto, A., Marinescu, R.: On the complexity of quantum circuit compilation. In: Bulitko, V., Storandt, S. (eds.) Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS 2018, Stockholm, Sweden - 14-15 July 2018. pp. 138–142. AAAI Press (2018)
9. Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., Shimony, S.: ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In: IJCAI. pp. 740–746 (2015)
10. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. JAIR **31**, 591–656 (2008)
11. Felner, A., Stern, R., Shimony, S.E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N.R., Wagner, G., Surynek, P.: Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In: Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017. pp. 29–37. AAAI Press (2017)

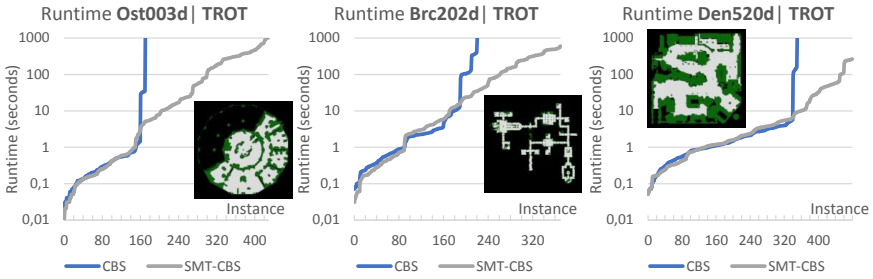

Fig. 16. Sorted runtimes of CBS and SMT-CBS solving TROT on `ost003d`, `brc202d`, and `2den520d` maps.

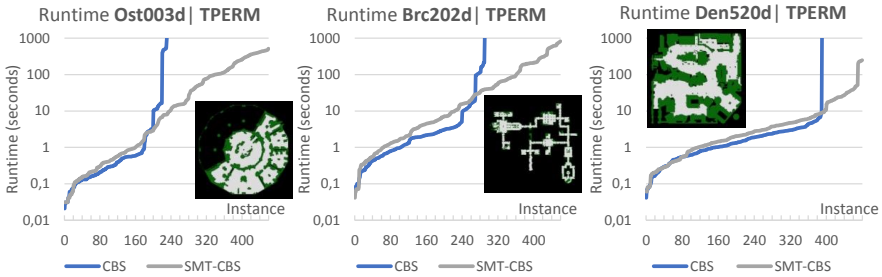

Fig. 17. Sorted runtimes of CBS and SMT-CBS solving TPERM on `ost003d`, `brc202d`, and `den520d` maps.

12. Kapadia, M., Ninomiya, K., Shoulson, A., Garcia, F.M., Badler, N.I.: Constraint-aware navigation in dynamic environments. In: Motion in Games, MIG '13, Dublin, Ireland, November 6-8, 2013. pp. 111–120. ACM (2013)
13. Kawahara, J., Saitoh, T., Yoshinaka, R.: The time complexity of the token swapping problem and its parallel variants. In: WALCOM 2017 Proceedings. LNCS, vol. 10167, pp. 448–459. Springer (2017)
14. Kim, D.G., Hirayama, K., Park, G.K.: Collision avoidance in multiple-ship situations by distributed local search. Journal of Advanced Computational Intelligence and Intelligent Informatics **18**, 839–848 (09 2014)
15. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, 1984. pp. 241–250 (1984)
16. Luna, R., Bekris, K.E.: Push and swap: Fast cooperative path-finding with completeness guarantees. In: IJCAI. pp. 294–300 (2011)
17. Luna, R., Bekris, K.: Efficient and complete centralized multi-robot path planning. In: IROS. pp. 3268–3275 (2011)
18. Luna, R., Bekris, K.E.: Network-guided multi-robot path planning in discrete representations. In: IROS. pp. 4596–4602 (2010)
19. Miltzow, T., Narins, L., Okamoto, Y., Rote, G., Thomas, A., Uno, T.: Approximation and hardness of token swapping. In: ESA 2016. LIPIcs, vol. 57, pp. 66:1–66:15. Schloss Dagstuhl (2016)

20. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). J. ACM **53**(6), 937–977 (2006)
21. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N x N extension of the 15-puzzle is intractable. In: AAAI. pp. 168–172 (1986)
22. Ryan, M.R.K.: Graph decomposition for efficient multi-robot path planning. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2003–2008 (2007)
23. Ryan, M.R.K.: Exploiting subgraph structure in multi-robot path planning. J. Artif. Intell. Res. (JAIR) **31**, 497–542 (2008)
24. Sharon, G., Stern, R., Felner, A., Sturtevant, N.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
25. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
26. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI (2012)
27. Silver, D.: Cooperative pathfinding. In: AIIDE. pp. 117–122 (2005)
28. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: AAAI. pp. 173–178 (2010)
29. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. Computational Intelligence and AI in Games **4**(2), 144–148 (2012)
30. Surynek, P.: Towards optimal cooperative path planning in hard setups through satisfiability solving. In: PRICAI 2012. pp. 564–576. Springer (2012)
31. Surynek, P.: An application of pebble motion on graphs to abstract multi-robot path planning. In: ICTAI 2009. pp. 151–158 (2009)
32. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009. pp. 3613–3619 (2009)
33. Surynek, P.: An optimization variant of multi-robot path planning is intractable. In: AAAI 2010. AAAI Press (2010)
34. Surynek, P.: On propositional encodings of cooperative path-finding. In: ICTAI 2012. pp. 524–531. IEEE Computer Society (2012)
35. Surynek, P.: Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In: ICTAI. pp. 875–882 (2014)
36. Surynek, P.: Simple direct propositional encoding of cooperative path finding simplified yet more. In: Nature-Inspired Computation and Machine Learning - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014. Lecture Notes in Computer Science, vol. 8857, pp. 410–425. Springer (2014)
37. Surynek, P.: Solving abstract cooperative path-finding in densely populated environments. Computational Intelligence **30**(2), 402–450 (2014)
38. Surynek, P.: Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. Ann. Math. Artif. Intell. **81**(3-4), 329–375 (2017)
39. Surynek, P.: Finding optimal solutions to token swapping by conflict-based search and reduction to SAT. In: IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018. pp. 592–599. IEEE (2018)
40. Surynek, P.: 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** (2018), `http://arxiv.org/abs/1809.05959`

41. Surynek, P.: Conflict handling framework in generalized multi-agent path finding: Advantages and shortcomings of satisfiability modulo approach. In: Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019, Volume 2. pp. 192–203. SciTePress (2019)
42. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: ECAI. pp. 810–818 (2016)
43. Surynek, P., Felner, A., Stern, R., Boyarski, E.: An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives. In: Symposium on Combinatorial Search (SoCS) (2016)
44. Thorup, M.: 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 (2002)
45. Wender, S., Watson, I.D.: Combining case-based reasoning and reinforcement learning for unit navigation in real-time strategy game AI. In: ICCBR. LNCS, vol. 8765, pp. 511–525. Springer (2014)
46. de Wilde, B., ter Mors, A., Witteveen, C.: Push and rotate: a complete multi-agent pathfinding algorithm. JAIR **51**, 443–492 (2014)
47. Wilson, R.M.: Graph puzzles, homotopy, and the alternating group. Journal of Combinatorial Theory, Series B **16**(1), 86 – 96 (1974)
48. Yamanaka, K., Demaine, E.D., Horiyama, T., Kawamura, A., Nakano, S., Okamoto, Y., Saitoh, T., Suzuki, A., Uehara, R., Uno, T.: Sequentially swapping colored tokens on graphs. In: WALCOM 2017 Proceedings. LNCS, vol. 10167, pp. 435–447. Springer (2017)
49. Yamanaka, K., Demaine, E.D., Ito, T., Kawahara, J., Kiyomi, M., Okamoto, Y., Saitoh, T., Suzuki, A., Uchizawa, K., Uno, T.: Swapping labeled tokens on graphs. In: FUN 2014 Proceedings. LNCS, vol. 8496, pp. 364–375. Springer (2014)
50. Yamanaka, K., Demaine, E.D., Ito, T., Kawahara, J., Kiyomi, M., Okamoto, Y., Saitoh, T., Suzuki, A., Uchizawa, K., Uno, T.: Swapping labeled tokens on graphs. Theor. Comput. Sci. **586**, 81–94 (2015)
51. Yamanaka, K., Horiyama, T., Kirkpatrick, D., Otachi, Y., Saitoh, T., Uehara, R., Uno, Y.: Computational complexity of colored token swapping problem. In: IPSJ SIG Technical Report. vol. 156 (2016)
52. Yershova, A., LaValle, S.M.: Improving motion-planning algorithms by efficient nearest-neighbor searching. IEEE Trans. Robotics **23**(1), 151–157 (2007)
53. Yu, J., LaValle, S.M.: Planning optimal paths for multiple robots on graphs. In: ICRA 2013. pp. 3612–3617 (2013)
54. Yu, J., LaValle, S.M.: Optimal multi-robot path planning on graphs: Structure and computational complexity. CoRR **abs/1507.03289** (2015)
55. Yu, J., LaValle, S.M.: Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. IEEE Trans. Robotics **32**(5), 1163–1177 (2016)
56. Yu, J., Rus, D.: Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In: Eleventh Workshop on the Algorithmic Fundations of Robotics (2014)
57. Zhou, D., Schwager, M.: Virtual rigid bodies for coordinated agile maneuvering of teams of micro aerial vehicles. In: ICRA 2015. pp. 1737–1742 (2015)