Finding Optimal Solutions to Token Swapping by Conflict-based Search and Reduction to SAT

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

***Abstract*—We study practical approaches to solving the *token swapping* (TSWAP) problem optimally in this paper. In TSWAP, we are given an undirected graph with colored vertices. A colored token is placed in each vertex. A pair of tokens can be swapped between adjacent vertices. The goal is to perform a sequence of swaps so that token and vertex colors agree across the graph. The minimum number of swaps is required in the optimization variant of the problem. We observed similarities between the TSWAP problem and *multi-agent path finding* (MAPF) where instead of tokens we have multiple agents that need to be moved from their current vertices to given unique target vertices. The difference between both problems consists in local conditions that state transitions (swaps/moves) must satisfy. We developed two algorithms for solving TSWAP optimally by adapting two different approaches to MAPF - *conflict-based search* (CBS) and *SAT-based* approach that uses multi-value decision diagrams (MDD-SAT). This constitutes the first attempt to design optimal solving algorithms for TSWAP. Experimental evaluation on various types of graphs shows that the reduction to SAT scales better than CBS in optimal TSWAP solving. It has been also demonstrated that TSWAP instances are easier to solve than corresponding similar MAPF instances.**

***Index Terms*—token swapping, multi-agent path finding, conflict-based search, SAT, optimality, multi-value decision diagrams, graphs**

I. Introduction

The *token swapping problem (TSWAP)* (also known as *sorting on graphs*) [1]–[3] represents a generalization of sorting problems a fundamental task in computer science [4]. 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. Usually the minimum number of swaps is desirable both in the classical sorting and in the TSWAP problem.

Using a modified notation from [5] the TSWAP problem is given by an undirected graph $G = (V, E)$ with vertex set V and edge set E. Each vertex in G is assigned a color in $C = \{c_1, c_2, ..., c_h\}$ via $\tau_+ : V \rightarrow C$. 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 1 for an example instance of TSWAP.

Many practical problems from logistics, robotics, and manipulation can be interpreted as token swapping. Examples include discrete multi-robot navigation and coordination [6], item rearrangement in automated warehouses [7], ship collision avoidance [8], or formation maintenance and maneuvering of aerial vehicles [9].

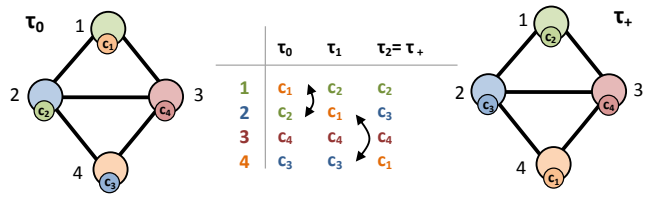

Fig. 1. A TSWAP instance. A solution consisting of two swaps is shown.

These motivating applications are closely related to another related problem studied in artificial intelligence - *multi-agent path finding* (MAPF).

The (MAPF) problem [10]–[14] is similar to TSWAP. MAPF 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 $\alpha : A \rightarrow V$. The task is to move agents in a non-conflicting way so that each agent reaches its goal vertex. An agent can move into adjacent unoccupied vertex provided no other agent enters the same target vertex. An example of MAPF instance is shown in Figure 2.

The TSWAP problem has been introduced recently. So far theoretical results concerning computational complexity [3] and approximations [15], [16] have appeared. To our best knowledge there is no study dealing with practical solving of TSWAP optimally. We would like to start to fill in this gap in this paper.

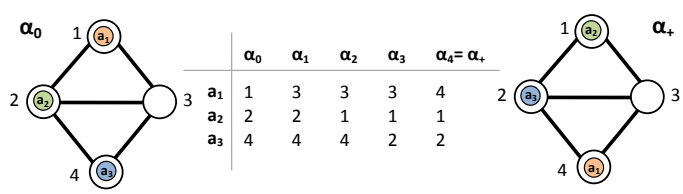

Fig. 2. A MAPF instance with three agents a_1, a_2, and a_3.

The similar MAPF problem has been studied longer and many theoretical results [17] as well as practical solving algorithms exist for MAPF [18]–[21]. Hence we would like to advance knowledge in solving both problems by initiating their cross-fertilization. Our current contribution consists in adapting optimal MAPF solvers for the TSWAP problem. We adapted *Conflict-Based Search* (CBS) [20] and MDD-SAT [22] based on reduction to *propositional satisfiability* [23] that uses *multi-value decision diagrams*.

The paper is organized as follows: We introduce TSWAP and MAPF problems formally and describe their theoretical properties first. Then we develop modifications of CBS and MDD-SAT algorithms called CBS(TSWAP) and MDD-SAT(TSWAP) for the TSWAP problem. We thoroughly discuss differences between TSWAP and MAPF and how they are reflected in the modifications of algorithms. Finally an experimental evaluation of CBS(TSWAP) and MDD-SAT(TSWAP) on a diverse set of synthetic TSWAP and MAPF benchmarks is presented.

II. Background

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 [5], [16]:

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

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 [24]. 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 swapping sequence that has as few swaps as possible is an NP-hard problem.

Similarly in MAPF we are given initial configuration of agents α_0 and goal configuration α_+. At each time step an agent can either *move* to an adjacent location or *wait* in its current location. The task is to find a sequence of move/wait actions for each agent a_i, moving it from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *conflict*, i.e., do not occupy the same location at the same time.

Definition 2: 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 $\alpha_\mu = \alpha_+$.

MAPF is usually solved aiming to minimize one of the two commonly-used global cumulative cost functions: (1) Sum-of-costs is the summation, over all agents, of the number of time steps required to reach the goal location [21]. (2) Makespan: is the time until the last agent reaches its destination (i.e., the maximum of the individual costs) [19].

III. Related Work

In contrast to the upper bound complexity result for the TSWAP problem, it can be shown that any solvable MAPF instance can be solved using $\mathcal{O}(|V|^3)$ moves [25], [26]. Moreover this a tight bound again as there as instances that need $\Omega(|V|^3)$ moves. As MAPF is of great practical interest many optimal, sub-optimal, and bounded sub-optimal solvers have been developed for MAPF. Our scope here is limited to optimal solvers. Optimal solvers for MAPF can be divided to two classes. (1) Search-based solvers. These algorithms consider MAPF as a graph search problem. Some of these algorithms are variants of the A* algorithm that search in a global *search space* – all different ways to place k agents into V vertices, one agent per vertex [18], [27]. Others algorithms such as ICTS [21] and CBS [20] search different search spaces and employ novel (non-A*) search tree. (2) Reduction-based solvers. By contrast, many recent optimal solvers reduce MAPF to known problems such as CSP [28], SAT [19], [29], [30], Inductive Logic Programming [31] and Answer Set Programming [32].

IV. Adapting Algorithms for TSWAP

We modified two optimal MAPF algorithms: CBS [20] and MDD-SAT [22] for the TSWAP problem. Both algorithms require modifications in handling of state transitions but their high level structure will be preserved.

A. Adapting the CBS Algorithm

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

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

Initially, we use shortest paths without considering any conflicts as an initial partial solution and store it into the priority queue. The top level search always takes the partial solution N with the lowest value of the objective and set of constraints forbidding conflicts $N.constraints$ associated with N. N is then checked for collisions between agents. If there are no collisions, then we can return $N.paths$ as an optimal solution. Otherwise we take the first collision; let this collision happened between agents a_i and a_j in vertex v at timestep t. Collision (a_i, a_j, v, t) will be resolved via branching at the top level search. One or the other involved agent must avoid v at t. This is carried out by adding two new states into the priority queue: partial solutions obtained by the low level search with respect to $N.constraints \cup \{(a_i, v, t)\}$ and $P.constrains \cup \{(a_j, v, t)\}$ associated with their extended conflict sets. For details on CBS we refer the reader to [20].

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

- tokens t_i and t_j both attempt to occupy vertex v at time step t (the analogical collision appears in MAPF too)
- token t_i traverses edge $\{u, v\}$ from vertex u to v at timestep t but t_j appearing in v at timestep t does not traverse $\{u, v\}$ from v to u at timestep t (this type of collision is unique to TSWAP)

Above cases represent collisions (t_i, t_j, v, t) and $(t_i, t_j, \{u, v\}, t)$ that are resolved by introducing conflicts (t_i, v, t) and (t_j, v, t) or $(t_i, \{u, v\}, t)$ and $(t_j, \{u, v\}, t)$ and by updating partial solutions accordingly. The collision of the second type as unique to TSWAP - it is a collision across edge $\{u, v\}$ that is resolved in a similar way as a standard vertex collision. That is, either t_i or t_j is forbidden to traverse $\{u, v\}$ at timestep t.

Complete CBS(TSWAP) is shown in pseudo-code as Algorithm 1. The difference from the standard CBS consists in the use of $N.constraints_E$ that forbids traversal of edges at given timesteps.

Algorithm 1: CBS(TSWAP) algorithm - a modification of CBS for the TSWAP problem

```
CBS(TSWAP) (G = (V, E), A, α_0, α_+)
    R.constraints_V ← ∅
    R.constraints_E ← ∅
    R.paths ← {shortest path from α_0(a_i) to
      α_+(a_i) | i = 1, 2, ..., k}
    R.ξ ← Σ_{i=1}^{k} ξ(N.paths(a_i))
    insert R into OPEN
    while OPEN ≠ ∅ do
        N ← min(OPEN)
        remove-Min(OPEN)
        collisions_V ← validateVertices(N.paths)
        collisions_E ← validateEdges(N.paths)
        if collisions_V = ∅ and collisions_E = ∅ then
            return N.paths
        else
            if collisions_V ≠ ∅ then
                let (a_i, a_j, v, t) ∈ collisions_V
                for each a ∈ {a_i, a_j} do
                    N'.constraints_V ←
                      N.constraints_V ∪ {(a, v, t)}
                    CBS(TSWAP)-New-Node(a, N, N')
            else
                if collisions_E ≠ ∅ then
                    let (a_i, a_j, {u, v}, t) ∈ collisions_E
                    for each a ∈ {a_i, a_j} do
                        N'.constraints_E ←
                          N.constraints_E ∪ {(a, {u, v}, t)}
                        CBS(TSWAP)-New-Node(a, N, N')

CBS(TSWAP)-New-Node(a, N, N')
    N'.paths ← N.paths
    N'.paths(a) ← update(a, N'.constraints_V, N'.cstraints_E)
    N'.ξ ← Σ_{i=1}^{k} ξ(N'.paths(a_i))
    insert N' into OPEN
```

B. Adapting the MDD-SAT Algorithm

The second algorithm we adapted is a compilation-based MDD-SAT [22]. This algorithm reduces an instance of MAPF to a series of decision Boolean satisfiability (SAT) [23] instances. Versions of MDD-SAT optimizing various objectives such as the *makespan* or the *sum-of-costs* in MAPF exist. The crucial step of the approach builds on formulating the decision of whether there is a solution to given MAPF of a specified value of the objective as an instance of SAT. Our major task when adapting MDD-SAT for TSWAP is hence to modify the SAT formulation to reflect different state transitions of TSWAP.

The SAT model of MAPF in MDD-SAT relies on a *time expansion* of G. That is we have a copy of G for every timestep [30]. Search for plans for individual agents can then be interpreted as a search for non-conflicting paths in the time expanded graph (an agent can visit a vertex multiple times hence plans for individual agents cannot be easily expressed as simple paths in unexpanded graph).

Boolean variables $\mathcal{X}(a)_v^t$ for each $v \in V$, $a \in A$ and each timestep t modeling occurrence of agent a in vertex v at timestep t are introduced. Auxiliary Boolean variables $\mathcal{E}(a)_{u,v}^t$ representing traversal of edge $\{u, v\}$ by agent a at timestep t are typically used for easier expressing of constraints. Constraints are introduced to restrict possible assignments of values to $\mathcal{X}(a)_v^t$ and $\mathcal{E}(a)_{u,v}^t$ variables so that assignments represent only valid solutions to MAPF - see for details [22].

We will follow similar scheme in the TSWAP problem. We introduce Boolean variable $\mathcal{Y}(c)_v^t$ representing occurrence of a token of color c in vertex v at time step t and analogically $\mathcal{S}(c)_{u,v}^t$ for $c \in C$ representing swap at edge $\{u, v\}$ involving color c (color c starts in u). Most of constraints such as those enforcing that only one color can assigned to each vertex can be taken directly from the MAPF encoding, since analogical properties must hold for agents. However there are some differences in TSWAP that need to be treated specifically.

Considering TSWAP Specific Constraints: For instance, swapping tokens along edge $\{u, v\}$ at time step t is slightly different as in MAPF we move an agent into a vacant vertex while in TSWAP we need to replace an outgoing token with

that from the target vertex:

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

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

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

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

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

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

The suggested encoding permits multiple non-conflicting swaps per single timestep (that is, $|F| > 1$ in the Definition 1). Observe that rotations of tokens over *non-trivial cycles* (a swap is a trivial rotation over an edge) consisting of at least three edges, that is common in some variants of MAPF, is forbidden by the encoding as it would violate constraint (1). Chain like movements from MAPF, where only the leader of a chain of agents need to enter vacant vertex while all other agents of the chain enter a vertex being simultaneously vacated by the preceding agent, are also forbidden.

Considering TSWAP Specific Objective: In TSWAP, we minimize the number of swaps which is an objective different from both the makespan and the sum-of-costs being implemented in MDD-SAT. As the SAT-based approach always answers a given formula in a yes/no manner we need to translate the minimization of the number of swaps in TSWAP to a series of queries to the SAT-solver. We always build a formula using above constructs that encodes a question whether there is a solution to TSWAP using specified number of swaps σ. By posting queries for $\sigma = 1, 2, 3...$ one can obtain the minimum number of swaps as σ corresponding to the first satisfiable formula.

The high-level optimization of the number of swaps within the MDD-SAT framework is shown in pseudo-code as Algorithm 2.

In our practical implementation we do not start with $\sigma = 1$ but with a lower bound estimation corresponding to the sum of lengths of shortest paths connecting the starting and target token positions divided by 2 (a single swap can move 2 tokens towards their target vertices each by one step forward).

Encoding the σ bound can be done through various cardinality constraints in the formula [33]–[35] on top of edge variables $\mathcal{S}(c)^t_{u,v}$. Once an edge variable is set to $True$ a corresponding swap is to be made. The following constraint

Algorithm 2: Main loop of MDD-SAT(TSWAP) - a modification of the MDD-SAT algorithm for TSWAP

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

hence need to be encoded in the formula through cardinality constraints.

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

As multiple token swaps can occur within the suggested encoding per single timestep, it is not necessary to make expansions for all steps up to σ. However careful setting of the number of time expansions with respect to given σ need to be done. We need to use sufficient number of expansions to ensure that if there is a swapping sequence of length σ we can find it within the given number of expansions. Precise calculation of the number of expansions depending on the cost parameter in MAPF has been done in [22]. We omit the analogical calculation for TSWAP here for the sake of brevity.

Reducing the Number of Variables: A common approach to reduce the number of decision variables in solving approaches to MAPF is the use of *multi-value decision diagrams* (MDDs) [21]. The basic observation that holds both for MAPF and TSWAP 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 token, we do not need to consider all vertices at time step t but only those that are reachable by the token in t timesteps from its initial position and that ensure that token's goal can be reached in the remaining $\sigma - t$ timesteps. This idea can reduce the size the expanded graph significantly and consequently can reduce the size of the Boolean formula by eliminating $\mathcal{Y}(c)^t_v$ and $\mathcal{S}(c,d)^t_{u,v}$ variables correspoding to unreachable vertices.

An example of MDD expansion for orange token (c_1) from Figure 1 in shown in Figure 3. The standard expansion taking copies of entire G for each timestep is shown for comparison too. As each vertex and edge in the time expansion correspond to $\mathcal{Y}(c)^t_v$ or $\mathcal{S}(d)^t_{v,u}$ propositional variable respectively, MDD expansion can reduce the size of the Boolean formula significantly.

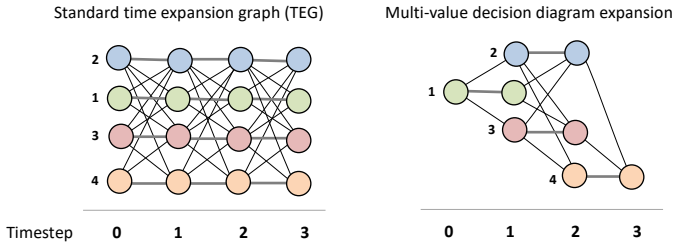

Fig. 3. A comparison of the standard time expansion of G and MDD-based expansion for token of color c_1 from the instance in Figure 1.

V. Experimental Evaluation

Results of our evaluation of the performance of presented solving approaches to TSWAP are presented in this section. To provide broader picture of difficulty of solving of related problems we also include tests with MAPF instances. We implemented suggested modifications of CBS and MDD-SAT for the TSWAP problem - we call our modification CBS(TSWAP) and MDD-SAT(TSWAP) respectively. The CBS algorithm for TSWAP has been implemented from scratch in C++ since the original implementation written in Java does support only grids but not general graphs [20]. To obtain MDD-SAT applicable on TSWAP we modified the existing C++ implementation [22], [36]. The underlying SAT solver in MDD-SAT is Glucose 3 [37], [38] [2]

Our implementations of both CBS(TSWAP) and MDD-SAT(TSWAP) support fully occupied graphs with tokens of different colors. To obtain more relevant comparison with MAPF solving where it is common that the graph is not fully occupied with agents we also added handling of empty vertices in both CBS(TSWAP) and MDD-SAT(TSWAP). Empty vertex is a vertex not containing any agent/token. [3]

A. Experimental Setup

All experiments were run on an i7 CPU 2.6 Ghz under Kubuntu linux 16 with 8GB RAM. [4] Tests were focused on collecting runtimes and other characteristics on various synthetic benchmarks. The timeout of 1 minute (60 seconds) was used in all tests.

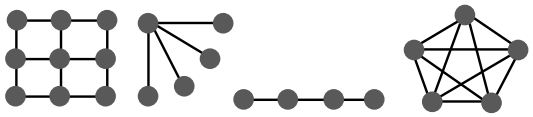

Fig. 4. Example of regular 4-connected *grid*, *star*, *path/bubblesort*, and *clique*.

[2]Technically Glucose and the MDD-SAT high-level control loop are compiled together as a single executable.

[3]In case of the TSWAP problem, empty vertices may be understood as those containing a special color representing the empty space. The implementations of CBS(TSWAP) and MDD-SAT(TSWAP) do not support multiple tokens of the same color, hence empty vertices are treated explicitly not using the special color. However, it is important to note that presented algorithms themselves can be implemented in a way to support multiple tokens of the same color.

[4]To enable reproducibility of presented results we provide complete source code of our TSWAP solvers on: `http://users.fit.cvut.cz/~surynpav/research/ictai2018`.

The experimental evaluation has been done on diverse instances consisting of grid graphs of sizes 8×8 and 16×16 which is a common benchmark for evaluation of MAPF solving [39], [40]. In addition to this we used *random graphs* containing 50% of random edges, *star* graphs, *path* graphs and *cliques* (see Figure 4) - these types of graphs have been already addressed by special rule-based algorithms in the literature [2]. Initial and goal configurations of tokens/agents have been generated randomly in all tests. To ensure solvability in MAPF, goal configurations were obtained by long random walks from initial configurations where random steps correspond to valid agent movements. Solvability was does not need to be treated in TSWAP tests as any TSWAP instance on a connected graph is solvable [1].

B. Comparison TSWAP Algorithms

Our first test has been focused on comparison of CBS(TSWAP) and MDD-SAT(TSWAP) on TSWAP instances. We used random graphs, stars, paths, and cliques in two orthogonal series of tests: **(1)** we varied number of vertices of the graph from 4 to 16 vertices while the graph was fully occupied and **(2)** the size of the graph was fixed to 16 vertices and we varied number of tokens from 4 up to 16. For each graph size and number of tokens we generated 10 random instances and measured runtime of CBS(TSWAP) and MDD-SAT(TSWAP). The number of swaps has been collected too.

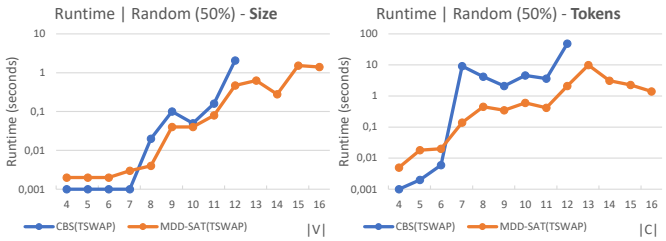

Fig. 5. Comparison of CBS(TSWAP) and MDD-SAT(TSWAP) algorithms on *random* TSWAP instances having 50% of random edges (varying size of G - left part; varying the number of tokens in graph with $|V| = 16$ - right part).

Mean values out of the 10 instances for runtime are presented in Figures 5, 6, 7, and 8.

The general observation is that for smaller instances (that is, those with small graph or containing few tokens) the performance comparison of CBS(TSWAP) and MDD-SAT(TSWAP) has no clear winner. On larger instances however MDD-SAT(TSWAP) tends to dominate. In all types types of tested graphs the time consumption grows quicker in CBS(TSWAP) as the size of TSWAP instance increases. We attribute this to worse performance of CBS when dealing with too many conflicts in densely occupied instances. TSWAP over tested graphs often represent an extreme case in this regards as its graph is close to fully occupied with tokens.

A close look at the individual types of graphs revels that on *random* graphs the performance of both tested algorithms is better than on *stars*, *paths*, and *cliques*. That is, larger

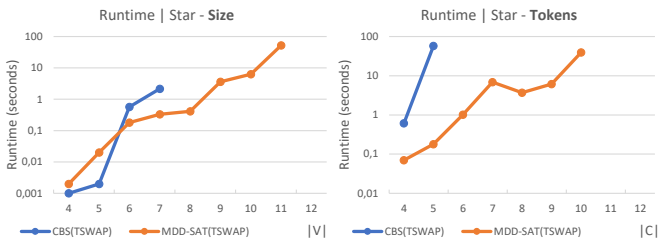

Fig. 6. Comparison of CBS(TSWAP) and MDD-SAT(TSWAP) algorithms on *star* TSWAP instances (varying size of G - left part; varying the number of tokens in graph with $|V| = 16$ - right part).

instances and more tokens can be solver in random graphs than in other three types. The difference in performance between CBS(TSWAP) and MDD-SAT(TSWAP) is smaller in random graphs too. We attribute this to a high connectivity in *random* graphs compared to *stars* and *paths* and to a limited parallelism of swaps compared to *cliques*. While the high connectivity helps tokens to reach their goals, high parallelism increases difficulty of solving because CBS(TSWAP) need to handle many conflicts and MDD-SAT(TSWAP) need to satisfy many constraints.

TABLE I
MEAN NUMBER OF SWAPS ON RANDOM GRAPHS (50%) AND CLIQUES

$\|V\|$	4	6	8	10	12	14	16
Random(50%)	9	11	14	27	26	33	39
Clique	9	11	14	24	-	-	-

The mean number of swaps is shown in Tables I and II. The number of swaps in optimal solutions is relatively high if we consider the small size of input instances. This is especially true for *star* and *path* graphs. The size of solutions is one of the factors causing difficulty of solving TSWAP on *star* and *path* graphs - the larger the optimal solution is the more difficult it is to find it.

Fig. 7. Comparison of CBS(TSWAP) and MDD-SAT(TSWAP) algorithms on *path* TSWAP instances.

TABLE II
MEAN NUMBER OF SWAPS ON STAR AND PATHS GRAPHS

$\|V\|$	4	5	6	7	8	9	10	11	12	13
Star	9	34	32	34	38	54	66	60	-	-
Path	20	14	26	35	33	64	47	94	97	101

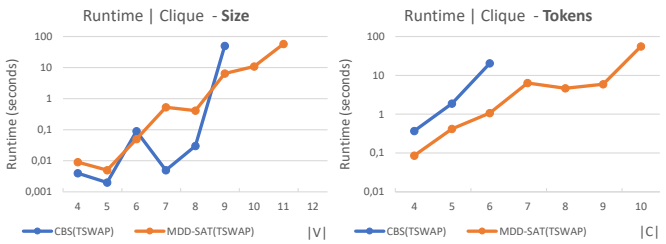

Fig. 8. Comparison of CBS(TSWAP) and MDD-SAT(TSWAP) algorithms on *clique* TSWAP instances.

C. Comparison of Difficulty of TSWAP and MAPF

We also compared difficulty of TSWAP and MAPF solving. This has been done with both CBS(TSWAP) and MDD-SAT(TSWAP). We used grids 8×8 and 16×16 in this test. For a fixed graph we gradually increased the number of tokens/agents and measured runtimes of CBS(TSWAP) and MDD-SAT(TSWAP). Again, for each number of agents/tokens 10 random instances were generated. Mean values out of these 10 instances are reported in Figures 9 and 10.

The evaluation on grids shows almost the order of magnitute speedup for the TSWAP problem compared to MAPF. This is a result in line with theoretical results for TSWAP, namely with the fact that TSWAP requires fewer swaps than MAPF moves ($\mathcal{O}(|V|^2)$ in TSWAP against $\mathcal{O}(|V|^3)$ in MAPF).

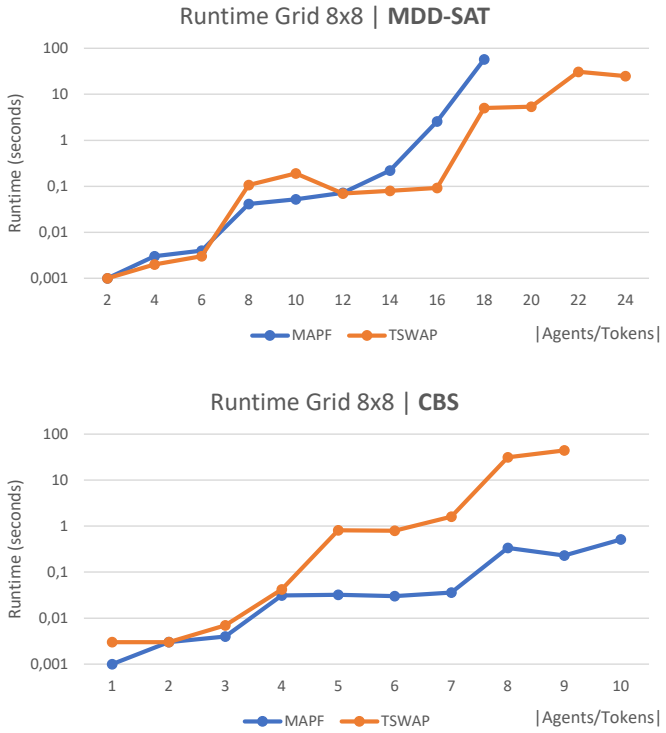

Fig. 9. Comparison of MAPF and TSWAP solving on a grid of size 8×8 by the MDD-SAT and CBS algorithms.

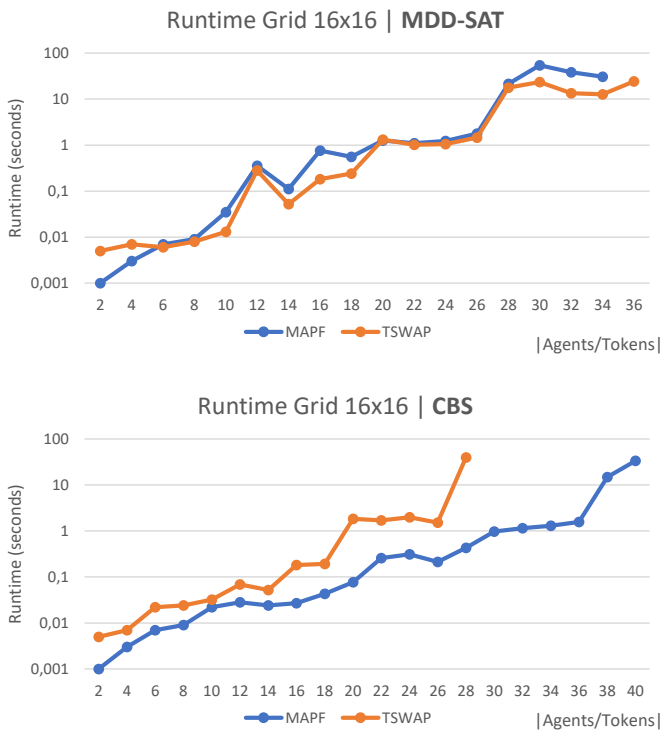

Fig. 10. Comparison of MAPF and TSWAP solving on a grid of size 16×16 by the MDD-SAT and CBS algorithms.

VI. Conclusion

We observed similarities between two important problems being addressed in computer science - the *multi-agent path finding* (MAPF) and *token swapping problem* (TSWAP) in graphs. We focused on optimal TSWAP solving in terms of the number of swaps. We demonstrated how to modify existing MAPF algorithms for TSWAP: search-based CBS and SAT-based MDD-SAT. The literature so far focused on theoretical properties of TSWAP but not on its optimal solving. To our best knowledge this is the first attempt to solve TSWAP optimally from the practical solving point of view.

We performed series of tests with our modified algorithms CBS(TSWAP) and MDD-SAT(TSWAP) on a diverse set of synthetic benchmarks. Our experimental results indicate that modifications of CBS and MDD-SAT represent a viable options for TSWAP solving while MDD-SAT(TSWAP) exhibits better performance. Better performance of the MDD-SAT(TSWAP) algorithm on TSWAP instances can be explained by the fact that highly constrained small instances suit better to the SAT solver than to CBS that was primarily designed for large instances with limited interaction among agents.

Results also indicate that TSWAP instances are easier than corresponding MAPF instances. This is in line with theoretical results for TSWAP and MAPF saying that solutions of TSWAP are shorter.

The obvious limitation of presented approaches is that they are not applicable on larger structured instances containing patterns like *paths* or *stars* that may appear in practice. It is also rarely the case in practice that we need all tokens to have distinct colors. Hence a viable step towards practically interesting large structured instances may be consideration of multiple tokens sharing a color. We plan to investigate this in our future work. Treating of *meta-tokens* - a group consisting of tokens of the same color analogically to how single tokens are treated in current algorithms may lead to new algorithmic concepts. For example shortest paths for meta-tokens may be searched as network flows [41] which eventually has the potential to stop the combinatorial explosion at the level of meta-tokens.

References

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

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

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

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

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

[6] R. Luna and K. E. Bekris, "Network-guided multi-robot path planning in discrete representations," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, 2010, pp. 4596–4602.

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

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

[9] D. Zhou and M. Schwager, "Virtual rigid bodies for coordinated agile maneuvering of teams of micro aerial vehicles," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 1737–1742.

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

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

[12] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, 2009, pp. 3613–3619.

[13] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, 2013, pp. 3612–3617.

[14] ——, "Optimal multi-robot path planning on graphs: Structure and computational complexity," *CoRR*, vol. abs/1507.03289, 2015.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[36] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "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.

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

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

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

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

[41] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, 2016*. ACM, 2016, pp. 1144–1152.