RAIRO-Oper. Res. 58 (2024) 441–455
https://doi.org/10.1051/ro/2023134

RAIRO Operations Research
www.rairo-ro.org

POLYNOMIAL TIME ALGORITHMS FOR THE TOKEN SWAPPING PROBLEM ON COGRAPHS

Caio Henrique Segawa Tonetti[1,*], Vinicius Fernandes dos Santos[1] and Sebastián Urrutia[2]

Abstract. The reconfiguration framework models the concept of transformation of combinatorial objects under a variety of operations and constraints. When it comes to reconfiguration challenges, the questions of importance are connectivity, diameter and distance, which can be considered and restrained in many ways. This work focuses on the Token Swap problem, a reconfiguration problem with variations that even precede the systematic study of the reconfiguration framework. In this problem, the goal is to convert an initial token placement on the vertices of a graph into a target token placement with the minimum number of swap operations. The main result of this paper is the construction of a polynomial algorithm for threshold graphs and subsequently cographs.

Mathematics Subject Classification. 05C85, 05C75.

Received February 22, 2023. Accepted September 1, 2023.

1. Introduction

The reconfiguration framework [12] formalizes the notion of *transformation* of abstract objects and questions arises from the necessity of understanding these changes under various operations and constraints. A simple problem in this class is sorting a list of numbers by adjacent swaps between two elements. A swap sequence can be calculated in polynomial time by a modified bubble sort algorithm [15]. The minimum number of swaps is called the *swap number* and in this case, is exactly the number of pairs of elements that are out of order.

Under other conditions, in a related problem, the list is to be sorted by *prefix-reversals*, an elementary operation that flips a prefix of the list. In this case, finding the minimum number of flips is NP-Hard [8]. This problem is called the pancake sorting problem.

Another well-known reconfiguration problem is the n-puzzle, a sliding puzzle on a grid of n numbered square tiles with exactly one tile missing. In this problem, each step slides a tile to an adjacent empty tile space to achieve a final sorted configuration state. Research of the 15-puzzle dates back to the late 19th century [13] and is commonly used as an introductory problem for modelling heuristics. Figure 1 is an example of a 15-puzzle instance.

Keywords. Graph theory, reconfiguration problems.

[1] Deparment of Computer Science, Federal University of Minas Gerais, Av. Antônio Carlos, 6627, Belo Horizonte, Minas Gerais, Brazil.

[2] Faculty of Logistics, Molde University College, Molde Campus, Britvegen 2, NO-6410, Molde, Norway.

*Corresponding author: caio.tonetti@gmail.com

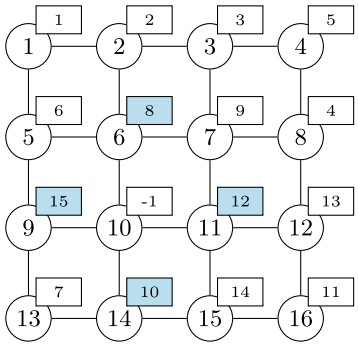


FIGURE 1. Instance of the 15-puzzle problem in a grid graph. Each vertex is represented by a circle and the rectangle is the tile currently positioned in the vertex. The -1 rectangle represents the empty tile. Each light blue colored rectangle is a possible swap operation with the empty tile in the current configuration.

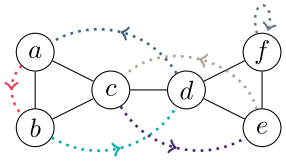


FIGURE 2. Example of an instance of the Token Swap problem with $V = \{a, b, c, d, e, f\}$ and tokens represented by distinct colored dotted arrows. The arrows point from the vertex that is originally positioned in mapping f_0 to the target vertex in the identity map f_i.

Considering the reconfiguration framework, the interest usually lies in one of the three following problems: connectivity, diameter or distance. These problems translate, respectively, into the following questions: (a) Can any configuration be reconfigured into another? (b) What is the maximum number of required steps between any pair of configurations? and (c) Given two configurations, what is the minimum number of operations to move from one to the other? In the examples aforementioned, the distance between two configurations (the initial and the target) was used. More information on reconfiguration problems can be found in the surveys available in the literature [17, 18, 25].

This work focuses on a reconfiguration problem called the Token Swapping Problem (TS). Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $|E|$ edges. Let $f_0 : V \mapsto V$ be an initial bijective token placement. The objective is to reconfigure this initial token placement into the identity token placement f_i that maps every node to itself with minimum distance. The reconfiguration must consist of a token swapping sequence S, in which each element of S is a pair of adjacent graph vertices, meaning that the elementary operation is restricted to a swap between two tokens placed on vertices that share an edge in the graph.

The decision version of this problem aims to determine if it is possible to have a swap sequence S that transforms f_0 to f_i in k or less swaps, for a given $k \in \mathbb{N}$. In Figure 2, a complete example of a TS instance is shown. The formal mathematical definitions needed to fully understand this work will be given in Section 3.

As we will review in Section 2, TS is known to be hard, in many senses, but solvable in polynomial-time for a few special cases. In this work we generalize some results in the literature for a much larger class of graphs, namely, cographs. Our main contribution is the following theorem: The Token Swapping Problem can be solved in polynomial time in cographs.

2. The token Swapping Problem: related literature

Every reconfiguration problem may be defined using a reconfiguration graph where each node is a possible state of the combinatorial or geometric object and an edge exists between two vertices if and only if each of the vertices can be reached in exactly one reconfiguration step from the other [18]. In the case of the TS, the vertices of the reconfiguration graph correspond to the possible token placements and the reconfiguration graph will be connected as long as the original graph is also connected. The connectivity guarantees that any configuration can be reconfigured into any other, as it is possible to take any spanning tree of the original graph and greedily move any token corresponding to a leaf to its correct position. Then, the leaf can be disregarded and the procedure continues inductively until the final configuration is achieved. This process gives us an upper-bound of $O(n^2)$ swaps for any instance of the TS problem [28].

Although the reconfiguration graph of a given graph G could be theoretically built from G, in practice this is not viable, since, in general, the number of configurations is factorial in $|V|$.

Applications of the TS problem encompass a wide range of fields and some examples are: computing efficient interconnection network structures [4], computational biology [5, 11], model Wireless Sensor Networks (WSS) [27], protection routing [19] and qubit allocation for quantum computers [21, 22]. More applications can be found in [2].

Although variants and particular cases of TS have been studied of decades the reconfiguration version of TS considered was first formalized in 2015 [28], and further generalizations and variations were studied in the same year [29]. The TS problem was proved to be NP-Complete, even when restricted to bipartite graphs with degree bounded by 3. It is also APX-complete and W[1]-hard parameterized by the number of swaps, but fixed parameter tractable (FPT) for the class of nowhere dense graphs [14, 16]. Subsequently, it was proved that the problem remains hard even when both the treewidth and the diameter of the input graph are constant [7]. These are some special classes of graphs in which the problem can be solved through exact polynomial time algorithms: cliques, paths, cycles, stars, brooms, lollipop [14], complete bipartite graphs and complete split graphs [6], where the last two are subclasses of cographs. We remark that all these graph classes are extremely restricted, in the sense that, for each n, there is a constant number of graphs on n vertices in any of them.

Approximation algorithms have also been studied for the problem. In particular, there are 2-approximation algorithms for square of paths [11] and for trees [16, 28], the latter being known to be tight [6]. It is established that the known techniques for approximating TS on trees cannot achieve better approximation factors [2]. Both approximation algorithms can be adapted to general graphs, providing α- and 4-approximation algorithms, respectively, where α is a value for which the input graph admits an α-spanner tree. In [7] it was conjectured that TS remains NP-Complete even in trees and [6] reinforced this conjecture by showing a counterexample of the *Happy Leaf Conjecture* [26]. And finally, it was shown that TS and other two variations (Weighted Token Swaping and Parallel Token Swaping) are NP-Complete on trees [2].

3. Problem definition and preliminaries

For an integer k, the notation $[k] := \{1, 2, \ldots, k\}$ is used. For a set V, a mapping function $f : V \mapsto V$ is a bijective function that internally maps elements of the set. An identity map is a special mapping function f_i that maps every element to itself.

For a set V, an ordering of the elements of the set is a bijective function $M : V \mapsto [|V|]$ and the shorthand $u <_M v$ for the comparison $M(u) < M(v)$ of the order of two elements $u, v \in V$ is adopted.

A graph $G := (V, E)$ is a pair of a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set E, and the elements of E are pairs of elements of V. The graph is called *undirected* when edges E are unordered tuples, while in *directed* graphs edges are ordered tuples.

The shorthand "uv" is used to describe a pair $\{u, v\}$ (or (u, v) in case directed graphs) and the functions $V(G)$ and $E(G)$ are used to respectively retrieve the sets V and E when they are omitted in the graph definition.

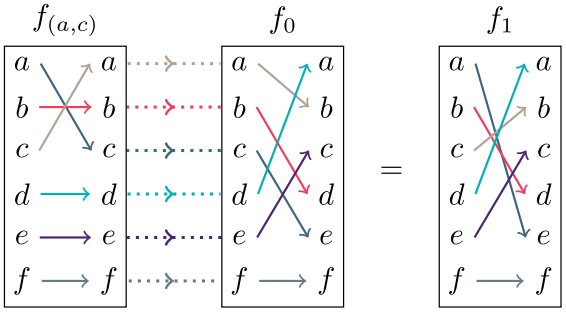


FIGURE 3. Let f_0 be the TS instance represented in Figure 2, $f_{(a,c)}$ be the mapping function representation of swap (a, c) and f_1 be the resulting mapping of the operation $f_0 \circ f_{(a,c)}$. Each rectangle represents a function with domain on the left, codomain on the right and color coded arrow mappings. It is possible to observe the resulting operation by following each arrow from left to right between $f_{(a,c)}$ and f_0.

A *subgraph* $\dot{G} := (\dot{V}, \dot{E})$ of the graph G, denoted as $\dot{G} \subseteq G$, is a graph such that $\dot{V} \subseteq V(G)$ and $\dot{E} \subseteq E(G) \cap \dot{V}^2$ and is called an *induced* subgraph when $\dot{E} = E(G) \cap \dot{V}^2$.

The outdegree $\delta_{\text{out}}(v)$ and indegree $\delta_{\text{in}}(v)$ of a vertex $v \in V(G)$ is the number of edges (v, w), for all $w \in V(G)$ and (w, v), for all $w \in V(G)$ that belong to $E(G)$, respectively. The degree $\delta(v)$ of a vertex is defined as the sum of $\delta_{\text{in}}(v)$ and $\delta_{\text{out}}(v)$ for directed graphs and as the number of edges having v as endpoint for undirected graphs.

The open neighborhood $N_G(u)$ of a vertex is defined as a set of all vertices v such that $uv \in E(G)$ and u itself is included. If u is not included, the set is called a closed neighborhood as is denoted as $N_G[u]$. The complement of a graph G is a graph $\dot{G}$ with the same vertex set $V(G)$ such that an edge exists between two distinct vertices if and only if it does not exist on G.

An instance of the TS problem is composed of an undirected graph $G := (V, E)$ and a mapping function f_0 that denotes an initial token placement on vertices of G. A *swap* is defined by a vertex pair $s := (u, v)$ and it can be *applied* to exchange two tokens in the current mapping of an instance provided that $uv \in E$. More formally, any swap $s = (u, v)$ can be represented as a mapping function f_s, such that $f_s(u) = v$, $f_s(v) = u$ and $f_s(w) = w$, for all $w \in V \setminus \{u, v\}$. f_s can be applied to any token placement f by the composition $f \circ f_s = f_1$. An example on how the composition operation can swap function values is shown in Figure 3.

Let $G := (V, E)$ be a graph. Two vertices $u, v \in V$ are *connected* if there is at least one *path of vertices* $(v_1, v_2, \ldots, v_k)$ in G such that for all $i \in [k-1]$, $v_i v_{i+1} \in E$, $v_1 = u$ and $v_k = v$. A graph G is connected if all vertex pairs of V are connected and disconnected otherwise. The function $dist_G(u, v)$ denotes the minimum number of edges in any path of vertices between nodes u and v for a graph G. Note that this distance is considered infinite in the case there is no path between the two vertices. A token placement f is considered *valid* if all vertex pairs $(f(u), u)$ are connected. As all token placements in this paper are assumed to be valid, every disconnected subgraph of a TS instance can be separated into smaller connected instances with restrictions to the domain of the initial placement. Hence, all graphs can be assumed connected without loss of generality.

Let $S := (s_1, s_2, \ldots, s_k)$ be a sequence of swaps. A swap sequence S *solves* an instance of the TS problem if and only if the identity function is resulted by applying each swap iteratively, as shown in equation (1). Every intermediate mapping function created is represented by a f_p, where p is the number of swaps applied from the initial configuration. Moreover, two placement mappings are said to be *adjacent* if there is exactly one swap to transform one placement into another. The Token Swapping problem asks for a sequence of swaps that solves a given instance with the minimum number of swaps k. The function $\text{OPT}_G(f)$ is a special function to symbolize

an optimal swap sequence size for a token placement f.

$$\underbrace{\left(\left(\underbrace{\left(\underbrace{(f_0 \circ f_{s_1})}_{f_1} \circ f_{s_2}\right)}_{f_2} \circ \dots\right) \circ f_{s_k}\right)}_{f_k} = f_i. \tag{1}$$

Another important definition needed throughout this work is the notion of the *Lowest Common Ancestor* (LCA), sometimes called nearest common ancestor, for trees. A *tree* G is an undirected and connected graph that has no cycles and is called *rooted*, denoted as G^r, if there exists a special node $r \in V(G)$ called *root* that functions as a reference node for heights in the graph. Any vertex u of a tree with $\delta(u) = 1$ is called a leaf, with the possible exception of the root of a rooted tree. A *subtree* is a subgraph of a tree that is still a tree. A subtree $\dot{G}$ of a rooted tree G^r can also be rooted in relation to the original root, as the nearest vertex from the subtree to r in G^r will be the subtree's root. For a given rooted tree G^r, $u, v \in V(G^r)$, the lowest common ancestor between u and v, denoted as $\mathrm{LCA}_{G^r}(u, v)$, is the lowest node such that both nodes are descendants of (possibly one of) them. In other words, it is the lowest shared ancestor of both u and v.

This notion is extended to calculate the LCA for any vertex subset of the tree. Let G^r be a rooted tree and $\dot{V} \subseteq V(G^r)$ a vertex subset. The lowest common ancestor $\mathrm{LCA}_{G^r}(\dot{V})$ is the nearest node from the root in the set of nodes built from the lowest common ancestors between every pair of nodes in $\dot{V}$. Given the set of every pairwise LCA of the subset, the LCA of the entire subset $\dot{V}$ is the nearest node from the root in the set.

The problem of calculating $\mathrm{LCA}_{G^r}(u, v)$ is called *offline* when the graph and queries are being given as input and has been first proposed in [1] with an optimally efficient algorithm. Other versions were studied later [10], with many different algorithms and general improvements over the years [3]. Some of these algorithms present a linear time pre-process stage that creates a data structure that can be dynamically queried in asymptotically constant time and others offers efficient algorithms that queries on trees that can be changed dynamically. The exact improvements and methods used to calculate the lowest common ancestor go out of the scope of this work. From the above, the calculation of the $\mathrm{LCA}_{G^r}(\dot{V})$ can also be derived in optimally efficient time, as the number of pairs of vertex is bounded by $O(|\dot{V}|^2)$.

A Conflict Graph $\mathrm{CG}_f := (V(G), E_{\mathrm{CG}})$ is a directed graph that, for a token placement f of a graph G, an edge $(u, v) \in E_{\mathrm{CG}}$ if and only if $f(u) = v$. Note that each node has outdegree 1, as there can be only one token per vertex, and indegree 1, as there is exactly one vertex for each token. Then, the graph is composed by a set of vertex disjoint directed cycles. Observe that the directed graph may contain self-loops whenever a token is already in the correct vertex. The number of cycles in the Conflict Graph is bounded by the number of vertices in the graph and it matches this bound when the current token placement matches the identity map f_i.

In Figure 2 the Conflict Graph is shown with dotted lines. The *joint representation* is an easy way to visually represent a Conflict Graph and the original graph as one, as seen in Figure 2, and will be used throughout this work.

Let f_x be a token placement and CG_{f_x} the related Conflict Graph with permutation cycle set $\mathrm{CS}(\mathrm{CG}_{f_x})$. Every swap that can be applied to f_x transforms CS in some way. These transformation can be classified into two types: *Merge* and *Split* as they are described in the following paragraph.

Take two distinct permutation cycles $C_i, C_j \in \mathrm{CS}(\mathrm{CG}_{f_x})$ and assume, without loss of generality, that $C_i = (u, u_1, \dots, u_k, u)$ and $C_j = (v, v_1, \dots, v_p, v)$, for $k, p \in \mathbb{N}_0$. A Merge is a swap (u, v) that is applied between the two cycles C_i and C_j resulting in a configuration $\dot{f}_x$ such that $\mathrm{CS}(\mathrm{CG}_{\dot{f}_x}) = (\mathrm{CS}(\mathrm{CG}_{f_x}) \setminus \{C_i, C_j\}) \cup \{C_{ij}\}$, where $C_{ij} = (v, u_1, \dots, u_k, u, v_1, \dots, v_p, v)$. Now, take a permutation cycle $C \in \mathrm{CS}(\mathrm{CG}_{f_x})$ and assume, without loss of generality, that $C = (u, u_1, \dots, u_k, v, v_{k+1}, \dots, v_{k+p}, u)$, for $k, p \in \mathbb{N}_0$. A Split is a swap (u, v) that is applied on the cycle C resulting in a configuration $\dot{f}_x$ such that $\mathrm{CS}(\mathrm{CG}_{\dot{f}_x}) = (\mathrm{CS}(\mathrm{CG}_{f_x}) \setminus \{C\}) \cup \{C_i, C_j\}$, where $C_i = (v, u_1, \dots, u_k, v)$ and $C_j = (u, v_{k+1}, \dots, v_{k+p}, u)$.

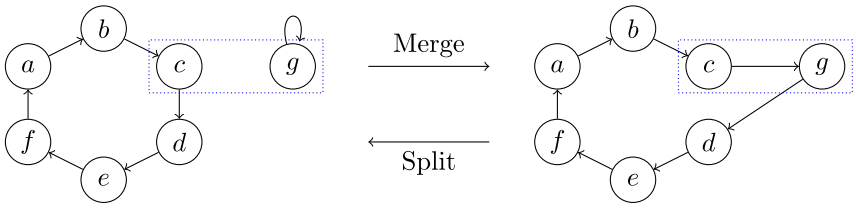


FIGURE 4. Let G be a connected graph with seven vertices where edge (c, g) must exist in $E(G)$. The left image denotes the current configuration as directed cycles and the blue rectangle shows which swap will be applied to achieve the configuration on the right image. This operation *merges* the two cycles and reapplying the swap returns the configuration to its original configuration, effectively *splitting* the two cycles.

Lemma 3.1. *Let G be a graph and f be a configuration of a* TS *problem instance. Any possible swap $(u, v) \in E(G)$ is either a merge or a split transformation in* $\mathrm{CS}(\mathrm{CG}_f)$.

Proof. By the Conflict Graph definition, every vertex must have outdegree one and indegree one and they are in cycles and self-loops. As every vertex is on a cycle and vertices between cycles are disjoint, every swap must be applied either internally on a cycle or between two cycles. A swap applied internally on a cycle is called a split swap and a swap applied between two cycles is called a merge swap by the above definition. □

From these transformations, there are two special cases that are worth mentioning: Swap between a cycle of size one and any other cycle and swap on an edge of the cycle. Figure 4 is an example to help visualize split and merge swaps. These two cases can be used as tools to add or remove, respectively, a node from a cycle and will be useful in this work. For a Conflict Graph CG_f, any cycle $C \in \mathrm{CS}(\mathrm{CG}_f)$ and vertex $v \in V(G)$, it is said that v *dominates* the cycle C if and only if the vertices $V(C)$ are a subset of the open neighborhood of v, $V(C) \subseteq N_G(v)$.

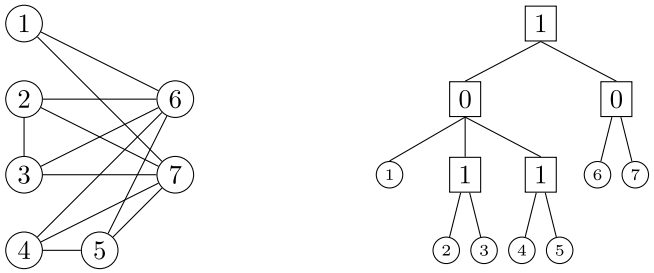
Let f be a token placement of a TS instance. The sum of the distances $\beta(f(u), u)$ is the sum of the distances between each token to its target vertex. With the sum, one could test if a swap sequence S solves the instance by checking if it is 0 for the resulting placement $f_{|S|}$. For trees, every swap can be classified in one of three categories related to the sum of distances of an instance: (a) The swap decreases the sum by two through moving two tokens closer to their target vertices, also called a *happy swap*; (b) the swap does not change the total sum by moving one token closer and one token further from their target vertices and (c) the swap increases the sum by two, as it moves two tokens further from its target vertices.

Intuitively, one could think that any swap sequence that solves a TS instance with swaps restricted to categories (a) and (b) will have less swaps than a swap sequence that solves the same instance and uses swaps of category (c) – as [23] tried to prove, but subsequently found an error [24]. Then, Vaughan [26] conjectured that any optimal swap sequence would not swap already correct tokens on leafs and called them *happy leafs*, resulting in the so-called *Happy Leaf Conjecture*. This conjecture was disproved by Biniaz *et al.* [6], as they showed that there is a class of infinite trees that need (c) swaps to achieve an optimal solution and that any algorithm that do not consider swaps on happy leafs has an approximation factor of *at least* $\frac{4}{3}$ for general graphs and trees.

4. Solving Token Swapping on cographs

A cograph is defined recursively as follows:

- A graph with a single vertex is a cograph;
- If $G_1, G_2, \ldots, G_k$ are cographs, then so is their disjoint union;
- If G is a cograph, then so is its complement $\overline{G}$.



(A) (B)

FIGURE 5. (a) Example of a cograph with labeled nodes; (b) Cotree that represents the structure of the cograph 5a.

Some authors also describe another rule called the *join* operation to build cographs. Given two cographs G_i and G_j, a join $join(G_i, G_j)$ results in a graph $\dot{G}$ such that $V(\dot{G}) = V(G_i) \cup V(G_j)$ and $E(\dot{G}) = E(G_i) \cup E(G_j) \cup \{uv \mid u \in V(G_i), v \in V(G_j)\}$.

This operation can be described by the basic cograph operations $join(G_i, G_j) = \overline{\overline{G_i} \cup \overline{G_j}}$, resulting in a valid operation to build cographs.

A cotree $CT(G)$ of a cograph $G = (V, E)$ is a rooted tree representing its structure. The leaves of $CT(G)$ are exactly V and each internal node is labelled 0 or 1 and will be called 0-node and 1-node, respectively. Each 1-node represents a join and each 0-node represents a disjoint union between the graphs represented by its subtrees.

The children of a 1-node are 0-nodes or leaves and the children of a 0-node are 1-nodes or leaves. Two vertices are adjacent in a cograph if and only if their lowest common ancestor (LCA) in the cotree is a 1-node. The cotree of any particular cograph is unique. Figure 5 shows a simple example of a cograph and its cotree representation.

Let G be a cograph and $CT(G)$ the respective cotree. Let f_0 be a token placement on G and CG_{f_0} the related Conflict Graph with set of permutation cycles $\mathrm{CS}(\mathrm{CG}_{f_0})$. Let CS^0 be the set of all cycles $C \in \mathrm{CS}$ such that the lowest common ancestor in the cotree of G of the vertices of the cycle is a 0-node of the cotree. Let CS^1 be the set of all cycles $C \in \mathrm{CS}$ such that the lowest common ancestor in the cotree of G of the vertices of the cycle is a 1-node of the cotree, or the cycle is a self-loop. These two sets form a partition of the set of cycles of the Conflict Graph and will be called the *zero-cycles* set and the *one-cycles* set respectively. Figure 6 shows the cotree in Figure 5 with a zero-cycle and a one-cycle.

4.1. Solving individual permutation cycles

The following two lemmas show how an individual permutation cycle C can be solved with $|C| - 1$ if it is a one-cycle and with $|C| + 1$ if it is a zero-cycle without affecting the rest of the configuration.

Lemma 4.1. *Any cycle C of CS^1 can be solved in $|C| - 1$ swaps.*

Proof. We proceed by induction on $|C|$. As base case consider $|C| < 3$. If $|C| = 1$ the cycle is already solved and $1 - 1 = 0$ swaps were used. If $|C| = 2$ the cycle can be solved with $2 - 1 = 1$ swap by swapping the two nodes of the cycle as they are adjacent in G.

Now we consider an one-cycle C of size at least 3. Let T be the subtree of $CT(G)$ rooted at $\mathrm{LCA}_{G^r}(C)$. Notice that there are at least two subtrees, each rooted on a child of the root of T that have at least one vertex of C, in which there is a vertex $v \in C$ whose predecessor u in C belongs to a subtree rooted in a different child

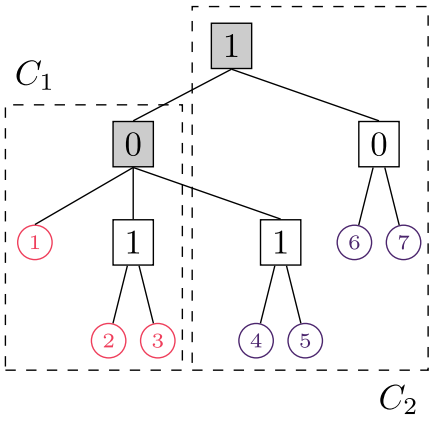


FIGURE 6. The cotree of the graph showed in Figure 5 is being used. Let f be a configuration where nodes $1, 2, 3$ are part of a permutation cycle C_1 and nodes $4, 5, 6, 7$ are part of another permutation cycle C_2. The lowest common ancestor of each cycle is denoted as a gray rectangle inside each corresponding labeled rectangle. C_1 belongs to CS^0 and C_2 belongs to CS^1.

of T. Then, a swap between u and v (we know they are adjacent in G because their LCA is a 1-node) splits C into one self-loop on v and a cycle $\dot{C}$ with size $|C| - 1$.

To ensure that the obtained $\dot{C}$ is a one-cycle we can choose v in such a way that $\mathrm{LCA}_{G^r}(\dot{C}) = \mathrm{LCA}_{G^r}(C)$. With that aim, we chose v such that its removal would keep vertices of $\dot{C}$ in at least two subtrees rooted in different children of T. This is always possible as it has at least 3 vertices and, in consequence, we have vertices of C in more than 2 children of T or at least one child of T has more than one vertex of C when there are exactly 2 children of T.

Then, the inductive hypothesis is applied on $\dot{C}$. Since $\dot{C}$ can be solved in $|C| - 2$ swaps, C is solved in $C - 1$ swaps. □

Lemma 4.2. *Any cycle C of CS^0 can be solved in $|C| + 1$ swaps.*

Proof. As G is connected, there must exist at least one ancestor of $\mathrm{LCA}_{G^r}(C)$ in $CT(G)$ that is a 1-node and connects the graph (or more locally, the nodes of C). From those 1-nodes, let w be the lowest among them and let T be the subtree of $CT(G)$ rooted at w. All the vertices in C are contained in a subtree rooted on a child of T and there must exist a vertex u that belongs to a subtree rooted in a differnet child of T. Then, since T is rooted on a 1-node, u must dominate $V(C)$.

Assume that u already has a correct token in the current configuration. We now merge it into C by using any swap (u, v), $v \in V(C)$, and call this new cycle $\dot{C}$. In this new cycle, the 1-node w is the lowest common ancestor and the technique presented at Lemma 4.1 can be used to solve $\dot{C}$ in $|\dot{C}| - 1$ swaps, which is equivalent to $|C|$ swaps, as there is one vertex more. In total, the number of swaps is $|C| + 1$ because of the first swap needed to add u to the cycle.

Notice that as the token of u is assumed to be correct, its token went back to the same place as before when the sequence of swaps is applied. This means that in fact, the token currently in u does not need to be the correct token. We can just assume it is correct for applying the swaps indicated in this lemma. □

With both of the methods presented, all permutation cycles of any TS instance on a cograph G can be solved with $|V(G)| - |\mathrm{CS}^1| + |\mathrm{CS}^0|$ swaps. Now, the next lemmas will carry on with the analysis of the optimality of each method. First, it is possible to show that the methods derived form the lemmas achieve the minimum number of swaps for individual cycles by Lemmas 4.3 and 4.6.

Lemma 4.3. *Individually, any cycle C of CS^1 cannot be solved in less than $|C|-1$ swaps.*

Proof. A solved instance (an identity mapping) has $|C|$ self-loop cycles on the vertices of C. Since each split swap increases the number of cycles by 1, at least $|C|-1$ split swaps are needed to transform the single cycle C into $|C|$ cycles. □

Lemma 4.4. *Let G be a cograph with a token placement f. Let C_1 and C_2 be the two cycles resulting from any split swap on a zero-cycle C. Either C_1 or C_2 must be a cycle with exactly the same lowest common ancestor as the original cycle C.*

Proof. Let T be the subtree of CG rooted on $\mathrm{LCA}_{G^r}(C)$. By definition of the LCA, the vertices in C are placed in at least two subtrees rooted on different children of $\mathrm{LCA}_{G^r}(C)$ in T. If they belong to more than two of those subtrees, then when C is split into C_1 or C_2, by the pigeon-hole principle [9,20], either C_1 or C_2 (or both) have vertices in more than one of those subtrees and it has the same lowest common ancestor as C.

Assume now that the vertices in C are placed in exactly two subtrees rooted on children of $\mathrm{LCA}_{G^r}(C)$ and that a single inner-swap divides C in such a way that C_1 is fully contained in the subtree rooted on one child of $\mathrm{LCA}_{G^r}(C)$ and C_2 is fully contained in another subtree rooted on another child of $\mathrm{LCA}_{G^r}(C)$. Notice that this is the only way in which the two new created cycles have lowest common ancestors that differ from the one of C. Such a swap must necessarily involve vertices from both subtress of $\mathrm{LCA}_{G^r}(C)$ as swaps between two vertices on the same subtree do not change the number of arcs of the cycle that cross from one child to the other. Now, notice that such a swap is impossible because $\mathrm{LCA}_{G^r}(C)$ is a 0-node. □

Lemma 4.5. *Let G be a cograph with a token placement f and let C be a zero-cycle. No split-swap can transform C into two one-cycles.*

Proof. By Lemma 4.4 at least one of the two cycles obtained after the split-swap maintains the same least common ancestor of the original cycle. Then it is still a zero-cycle. □

Lemma 4.6. *Individually, any cycle $C \in \mathrm{CS}^0$ cannot be solved in less than $|C|+1$ swaps.*

Proof. By Lemma 4.5, applying just split swaps to C there will always remain at least one zero-cycle. Then in order to solve the cycle at least one merge-swap is needed. Each merge swap decreases the number of cycles in CG by one. Then, at least one merge swap and $|C|$ splits swaps are needed for the single cycle C to be transformed into $|C|$ self-loops. Therefore, at least $C+1$ swaps are needed. □

This model of solving cycles can be used to prove the optimality of some subclasses of the cograph class. In complete graphs, every permutation cycle belongs to CS^1, as every vertex is connected to every other, resulting in an empty CS^0 set. In star graphs, the number of cycles $C \in \mathrm{CS}^1$ with $|C| > 1$ cannot be more than 1, as all nodes are only neighbors of the center node and the center node can be only part of one-cycle, resulting in every other non-trivial cycle belonging to CS^0. These two classes can be generalized to the class of threshold graphs, that will be considered in Lemma 4.9. There are many equivalent definitions for this class, but, for the sake of conciseness, we define a *threshold* graph as a graph that can be constructed from a single vertex graph by repeatedly adding either an isolated vertex or a dominating vertex. Also, a threshold graph is a graph free of induced cycles of size four (C_4), induced paths of size four (P_4) and induced two disjoint edges ($2K_2$).

Lemma 4.7. *Let G be a threshold graph with an initial token placement f_0. Then,* $\mathrm{OPT}_G(f_0) \leq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})|$.

Proof. Directly from Lemmas 4.1 and 4.2. □

Lemma 4.8. *Let G be a threshold graph with an initial token placement f_0. Then,* $\mathrm{OPT}_G(f_0) \geq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})|$.

Proof. Let $p(G,f) = |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_f)| + |\mathrm{CS}^0(\mathrm{CG}_f)|$. Notice that $p(G,f_i) = 0$ as $|\mathrm{CS}^1(\mathrm{CG}_{f_i})| = |V(G)|$ and $|\mathrm{CS}^0(\mathrm{CG}_{f_i})| = 0$. In the following, we show that any possible swap transforming f into an adjacent configuration $\dot{f}$ cannot decrease the value of $p(G,f)$ in more than 1. That is, $p(G,\dot{f}) \geq p(G,f) - 1$.

Recall that every swap, either split a cycle C or merge two cycles C_u and C_v in CG_f. Then, there are two possibilities to decrease $p(G,\dot{f})$ by more than one: (1) a zero-cycle is split into two one-cycles or (2) two zero-cycles are merged into one one-cycle.

The first case (1) is not possible since contradicts Lemma 4.5.

For the second case (2), observe that the fact that the merged cycle is an one-cycle implies that $\mathrm{LCA}_{G^r}(C_u) \neq \mathrm{LCA}_{G^r}(C_v)$, $\mathrm{LCA}_{G^r}(C_u)$ is not an ancestor of $\mathrm{LCA}_{G^r}(C_v)$ and $\mathrm{LCA}_{G^r}(C_v)$ is not an ancestor of $\mathrm{LCA}_{G^r}(C_u)$. If (at least) one of these cases does not hold, the merged cycle would remain a zero-cycle. Let T be the subtree of CG rooted on $\mathrm{LCA}_{G^r}(C_u \cup C_v)$.

The vertices of C_u and those of C_v lie in subtrees rooted in different children of T and since the root of T is a 1-node, all vertices in C_u are adjacent to all vertices in C_v in G. Now, take any two non-adjacent vertices x, y from $V(C_u)$ and two non-adjacent vertices w, z from $V(C_v)$ (they must exist on both cycles as they are zero-cycles).

Then, the induced subgraph $G[\{x,y,w,z\}]$ is a cycle of size 4. This is a contradiction on the definition of threshold graphs.

By the above analysis, it is possible to conclude that any token swap decreases $p(G,f)$ by at most one unit for any token placement f and obtain the Inequation (2).

$$p(G,\dot{f}) \geq p(G,f) - 1. \tag{2}$$

Thus, for any swapping sequence $S = (s_1, s_2, \ldots, s_k)$ that transforms the initial configuration f_0 to the identity configuration f_i through adjacent configurations $f_1, f_2, \ldots, f_k = f_i$, each pair of configurations $p(G,f_{j+1}) \geq p(G,f_j) - 1$ holds from Inequation (2) for $j = 1, 2, \ldots, k-1$. Take the sum of these inequations $\sum_j p(G,f_{j+1}) \geq p(G,f_j) - 1$ shown in Inequation (3).

$$\begin{aligned} p(G,f_1) &\geq p(G,f_0) - 1 \\ p(G,f_2) &\geq p(G,f_1) - 1 \\ &\cdots \\ p(G,f_k) &\geq p(G,f_{k-1}) - 1 \\ \hline p(G,f_k) &\geq p(G,f_0) - |S|. \end{aligned} \tag{3}$$

From Inequation (3) and substituting $p(G,f_k)$ by 0 we get:

$$\begin{aligned} p(G,f_k) &\geq p(G,f_0) - |S| \\ |S| &\geq p(G,f_0) - p(G,f_k) \\ |S| &\geq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_f)| + |\mathrm{CS}^0(\mathrm{CG}_f)|. \end{aligned} \tag{4}$$

□

Theorem 4.9. *Let G be a threshold graph with an initial token placement f_0. The minimum number of required swaps is given by $|V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})|$.*

Proof. Directly from Lemmas 4.7 and 4.8. □

4.2. Dependencies betweeen permutation cycles

Section 4.1 introduced techniques for solving cycles of the two types individually without changing the configurations of other cycles and proved the optimality of this method for threshold graphs. The proof for general cographs is constructed from the one for threshold graphs, observing that general cographs allow induced cycles of size 4. A merge of two zero-cycles into a one-cycle can save exactly two swaps in the final configuration. Let this type of swap be called a *cutback* swap.

A *Cycle Matching Graph* $H = (\mathrm{CS}^0, E_H)$ of a cograph G and initial token configuration f_0 is a graph where each node represents an individual zero-cycle of the current configuration. Let $u, v \in V(H)$ and C_u, C_v be the two related cycles from CS^0. An edge exists between nodes u and v if and only if the lowest common ancestor of the union of the vertices $V(C_u) \cup V(C_v)$ is a one-node. This graph will be used to identify the cases where two swaps can be saved.

Lemma 4.10. *Let G be a graph and $\mu(G)$ a maximum edge matching of G.*

(1) *Adding a vertex connected by edges to the graph G can increase the size of the matching $|\mu(G)|$ in at most one;*
(2) *Adding edges to a single vertex can increase the size of the matching $|\mu(G)|$ in at most one;*
(3) *Let $\dot{G}$ and $\ddot{G}$ be graphs such that $G \subseteq \dot{G} \subseteq \ddot{G}$, $|V(\dot{G}) \setminus V(G)| = 1$ and $|V(\ddot{G}) \setminus V(\dot{G})| = 1$. Then, $|\mu(\ddot{G})| \leq |\mu(G)| + 2$.*

Proof. The proofs are respectively enumerated below.

(1) Let $\dot{G}$ be the graph obtained by adding to G a vertex x connected to any subset of V and let $\mu(\dot{G})$ be a maximum edge matching of $\dot{G}$. Suppose x is saturated in $\mu(\dot{G})$ and $|\mu(\dot{G})| > |\mu(G)|$. $|\mu(\dot{G})|$ must be $|\mu(G)|+1$, otherwise the size of the maximum matching $|\mu(G)|$ results in a contradiction, as it is possible to use the matching $|\mu(\dot{G})|$ minus the edge involving x, obtaining a larger matching of G. If x is not saturated, then $|\mu(\dot{G})| = |\mu(G)|$.
(2) Let $\dot{G}$ be the graph obtained by adding to G any amount of edges involving a vertex $v \in V(G)$ and let $\mu(\dot{G})$ be a maximum edge matching of $\dot{G}$. Assume $|\mu(\dot{G})| > |\mu(G)|+1$. The matching obtained by removing the edge related to v in $\mu(\dot{G})$ (if it exists) is a valid matching on the original graph G and has size at least $|\mu(G)| + 1$, which contradicts the maximality of the matching of $\mu(G)$.
(3) By part (1) of this lemma, the size of the matching $\mu(\dot{G})$ can be at most 1 more than the size of $\mu(G)$, and the size of the matching $\mu(\ddot{G})$ can be at most 1 more than the size of $\mu(\dot{G})$. Combining both inequalities we get:

$$\mu(\dot{G}) \leq \mu(G) + 1 \tag{5}$$
$$\mu(\ddot{G}) \leq \mu(\dot{G}) + 1 \tag{6}$$
$$\mu(\ddot{G}) \leq \mu(G) + 2. \tag{7}$$

□

Let $\mu(H)$ be a maximum matching on the graph H. Each element of this matching represents a possible merge swap that can save two swaps in the final swap sequence (by the cost of one swap, two zero-cycles are transformed into one one-cycle), saving in total $2 \times |\mu(H)|$ swaps. Note that after applying these swaps, the resulting configuration will have a totally disconnected Cycle Matching Graph, as only zero-cycles unsaturated by $\mu(H)$ will remain.

Lemma 4.11. *Let G be a cograph with an initial token placement f_0. Then,* $\mathrm{OPT}_G(f_0) \leq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})| - 2 \times |\mu(H)|$.

Proof. Directly from Lemmas 4.3, 4.6 and the previous discussion. □

Lemma 4.12. *Let G be a cograph with an initial token placement f_0. Then,* $\mathrm{OPT}_G(f_0) \geq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})| - 2 \times |\mu(H)|$.

Proof. Let $p(G,f) = |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_f)| + |\mathrm{CS}^0(\mathrm{CG}_f)| - 2 \times |\mu(H)|$. Note that $p(G,f_i) = 0$ holds. In the following we show that any possible swap transforming f into an adjacent configuration $\dot{f}$ cannot decrease the value of $p(G,f)$ in more than 1. That is, $p(G,\dot{f}) \geq p(G,f) - 1$.

The following cases encompass every possible split swap of vertices u and v on the given configuration.

- **Case SPLIT-1.** Let u and v belong to the same cycle $C \in \mathrm{CS}^1$. This token swap breaks the original cycle into two vertex disjoint cycles C_u and C_v. Assume that $u \in V(C_u)$ and $v \in V(C_v)$.
  Case A. If $C_u \in \mathrm{CS}^1$ and $C_v \in \mathrm{CS}^1$, the value of $|\mathrm{CS}^1|$ is increased in 1 and the size of the matching $\mu(H)$ does not change, as graph H is not modified, resulting in $p(G,\dot{f}) = p(G,f) - 1$;
  Case B. If $C_u \in \mathrm{CS}^0$ and $C_v \in \mathrm{CS}^1$ or if $C_u \in \mathrm{CS}^1$ and $C_v \in \mathrm{CS}^0$, the value of $|\mathrm{CS}^0|$ is increased by 1, increasing the number of vertices in H by one. By Lemma 4.10, the matching can increase in at most one unit, resulting in $p(G,f) - 1 \leq p(G,\dot{f}) \leq p(G,f) + 1$;
  Case C. If $C_u \in \mathrm{CS}^0$ and $C_v \in \mathrm{CS}^0$, the value of $|\mathrm{CS}^0|$ is increased by 2 and $|\mathrm{CS}^1|$ is decreased by 1, increasing the number of vertices in H in two with a guaranteed additional edge between C_u and C_v as $\mathrm{LCA}_{\mathrm{CG}(G)}(V(C_u \cup C_v))$ is a 1-node. Then, $|\mu(H)|$ increases at least 1 and by Lemma 4.10, the increase is at most 2, resulting in $p(G,f) - 1 \leq p(G,\dot{f}) \leq p(G,f) + 1$.
- **Case SPLIT-0.** Let u and v belong to the same cycle $C \in \mathrm{CS}^0$. This token swap breaks the original cycle into two vertex disjoint cycles C_u and C_v. Assume that $u \in V(C_u)$ and $v \in V(C_v)$.
  Case A. Let $C_u, C_v \in \mathrm{CS}^1$. This case is impossible as stated by Lemma 4.5.
  Case B. If $C_u \in \mathrm{CS}^0$ and $C_v \in \mathrm{CS}^1$ or if $C_u \in \mathrm{CS}^1$ and $C_v \in \mathrm{CS}^0$, the value of $|\mathrm{CS}^1|$ is increased by one and $|\mathrm{CS}^0|$ does not change. Without loss of generality, suppose that $C_u \in \mathrm{CS}^0$ and $C_v \in \mathrm{CS}^1$. By Lemma 4.4, $\mathrm{LCA}_{\mathrm{CG}(G)}(V(C_u)) = \mathrm{LCA}_{\mathrm{CG}(G)}(V(C))$, not changing the graph H, resulting in $p(G,\dot{f}) = p(G,f) - 1$;
  Case C. If $C_u \in \mathrm{CS}^0$ and $C_v \in \mathrm{CS}^0$, the value of $|\mathrm{CS}^0|$ is increased by 1 and $|\mathrm{CS}^1|$ does not change. By Lemma 4.4, either C_u or C_v has the same lowest common ancestor as the cycle C. That cycle introduces in H a vertex with the same neighborhood as the vertex corresponding to C that is being removed. Then, in practice, exactly one new vertex is added to H. Lemma 4.10 shows that the maximum matching can increase in at most one in this case, resulting in $p(G,f) - 1 \leq p(G,\dot{f}) \leq p(G,f) + 1$.

It is no needed to check the merge swaps in this case, as each of them is just the inverse of a split swap. This means that the previous equations and inequations have inverted plus and minus signals. No equation or inequation show and increase larger than 1, so no merge swap can decrease $p(G,f)$ in more than 1.

By the above analysis, it is possible to conclude that any token swap decreases $p(G,f)$ by at most one for any token placement f and obtain:

$$p(G,\dot{f}) \geq p(G,f) - 1. \tag{8}$$

Thus, for any swapping sequence $S = (s_1, s_2, \ldots, s_k)$ that transforms the initial configuration f_0 to the identity configuration f_i through adjacent configurations $f_1, f_2, \ldots, f_k = f_i$, for each pair of configurations $p(G,f_{j+1}) \geq p(G,f_j) - 1$ holds. Taking the sum of these inequations $\sum_j p(G,f_{j+1}) \geq p(G,f_j) - 1$ we get:

$$\begin{gathered} p(G,f_1) \geq p(G,f_0) - 1 \\ p(G,f_2) \geq p(G,f_1) - 1 \\ \cdots \\ p(G,f_{k-1}) \geq p(G,f_k) - 1 \\ \hline p(G,f_k) \geq p(G,f_0) - |S|. \end{gathered} \tag{9}$$

From Inequation (9) and substituting $p(G, f_k)$ for 0, we get:

$$\begin{aligned} p(G, f_k) &\geq p(G, f_0) - |S| \\ |S| &\geq p(G, f_0) - p(G, f_k) \\ |S| &\geq |V(G)| - |\mathrm{CS}^1(\mathrm{CG}_f)| + |\mathrm{CS}^0(\mathrm{CG}_f)| - 2 \times |\mu(H)|. \end{aligned} \tag{10}$$

□

Theorem 4.13. *Let G be a cograph with an initial token placement f_0. The minimum number of required swaps is given by $|V(G)| - |\mathrm{CS}^1(\mathrm{CG}_{f_0})| + |\mathrm{CS}^0(\mathrm{CG}_{f_0})| - 2 \times |\mu(H)|$.*

Proof. Directly from Lemmas 4.11 and 4.12. □

Theorem 4.14. *Token Swapping Problem can be solved in polynomial time in cographs.*

Proof. Let G be a cograph and f be a token configuration. By the previous lemmas, the following algorithm solves the Token Swapping Problem in G with initial configuration f.

(1) Compute the cotree of G.
(2) Compute the Conflict Graph CG_f.
(3) For each cycle in CG_f determine if it is a zero-cycle or a one-cycle.
(4) Compute H using the zero-cycles from CG_f.
(5) Compute a maximum matching in H.
(6) For each pair of zero-cycles matched in the computed maximum matching, perform a cutback swap using any edge between vertices of both cycles.
(7) Solve the remaining zero-cycles with the procedure implied by the proof of Lemma 4.2 and the remaining one-cycles with the procedure implied by the proof of Lemma 4.1.

Since each step of the algorithm can be performed in polynomial time, the whole algorithm can be executed in polynomial time. □

5. Conclusions

In this article we considered the Token Swapping Problem in Graphs. For the particular cases of threshold graphs and cographs, it is shown how the value of the optimal solution can be calculated. It is also shown that it is possible to compute a solution with a cost matching this value, thus showing that this NP-Hard problem is polynomial-time solvable for the cograph class of graphs.

In future works we aim to tackle the problem for other classes of graphs as well as considering the problem on general graphs and restricting the initial token configuration. Another approach can be understanding the relation between classes of graphs and variations of the *distance to cographs* problem, which is the minimum number of vertices that have to be deleted from a graph in order to obtain a cograph.

Acknowledgements. The first author would like to first thank his mother, Liria Yuriko, that always supported my decisions. He would also like to thank his advisor, Vinicius Fernandes dos Santos, and co-advisor, Sebastián Urrutia, for the encouragement, patience and vast knowledge in their respective fields. Finally, we would like to thank CAPES, CNPq and

FAPEMIG for partially funding this research. The first author was funded in part by the grant Programa de Excelência Acadêmica (PROEX), file number 88882.348155/2019-01, from CAPES.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, On finding lowest common ancestors in trees, in Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC'73. Association for Computing Machinery. New York, NY, USA (1973) 253–265.

[2] O. Aichholzer, E.D. Demaine, M. Korman, A. Lubiw, J. Lynch, Z. Masárová, M. Rudoy, V. Vassilevska Williams and N. Wein, Hardness of Token Swapping on Trees, in 30th Annual European Symposium on Algorithms (ESA 2022). Vol. 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, edited by S. Chechik, G. Navarro, E. Rotenberg and G. Herman. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022) 3:1–3:15.

[3] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe, Nearest common ancestors: a survey and a new algorithm for a distributed environment. *Theory Comput. Syst.* **37** (2004) 441–456.

[4] F. Annexstein, M. Baumslag and A.L. Rosenberg, Group action graphs and parallel architectures. *SIAM J. Comput.* **19** (1990) 544–569.

[5] V. Bafna and P.A. Pevzner, Sorting by transpositions. *SIAM J. Discret. Math.* **11** (1998) 224–240.

[6] A. Biniaz, K. Jain, A. Lubiw, Z. Masárová, T. Miltzow, D. Mondal, A.M. Naredla, J. Tkadlec and A. Turcotte, Token swapping on trees. *Discrete Math. Theor. Comput. Sci.* **24** (2023) 1–37.

[7] É. Bonnet, T. Miltzow and P. Rzążewski, Complexity of token swapping and its variants. *Algorithmica* **80** (2018) 2656–2682.

[8] L. Bulteau, G. Fertin and I. Rusu, Pancake flipping is hard. *J. Comput. Syst. Sci.* **81** (2015) 1556–1574.

[9] P. Erdös and R. Rado, A Partition Calculus in Set Theory. Birkhäuser Boston, Boston, MA (1987) 179–241.

[10] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* **13** (1984) 338–355.

[11] L. Heath and J. Vergara, Sorting by short swaps. *J. Comput. Biol. J. Comput. Mol. Cell Biol.* **10** (2003) 775–89.

[12] T. Ito, E.D. Demaine, N.J. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara and Y. Uno, On the complexity of reconfiguration problems. *Theor. Comput. Sci.* **412** (2011) 1054–1065.

[13] W.W. Johnson and W.E. Story, Notes on the "15" puzzle. *Am. J. Math.* **2** (1879) 397–404.

[14] J. Kawahara, T. Saitoh and R. Yoshinaka, The time complexity of the token swapping problem and its parallel variants, in WALCOM: Algorithms and Computation. Springer International Publishing, Cham (2017) 448–459.

[15] D.E. Knuth, The art of computer programming, in Sorting and Searching, 2nd edition. Vol. 3. Addison Wesley Longman Publishing Co., Inc., USA (1998).

[16] T. Miltzow, L. Narins, Y. Okamoto, G. Rote, A. Thomas and T. Uno, Tight exact and approximate algorithmic results on token swapping. Preprint arXiv:1602.05150 (2016).

[17] A.E. Mouawad, *On reconfiguration problems: structure and tractability*. Ph.D. thesis, University of Waterloo (2015).

[18] N. Nishimura, Introduction to reconfiguration. *Algorithms* **11** (2018) 52.

[19] K.-J. Pai, R.-S. Chang and J.-M. Chang, Constructing dual-cists of pancake graphs and performance assessment of protection routings on some cayley networks. *J. Supercomput.* **76** (2020) 124546.

[20] A. Razborov, Proof complexity of pigeonhole principles, in Developments in Language Theory. Springer Berlin Heidelberg, Berlin, Heidelberg (2002) 100–116.

[21] M.Y. Siraichi, V.F.D. Santos, S. Collange and F.M.Q. Pereira, Qubit allocation, in Proceedings of the 2018 International Symposium on Code Generation and Optimization. ACM, New York, NY, USA (2018) 113–125.

[22] M.Y. Siraichi, V.F.D. Santos, C. Collange and F.M.Q.A. Pereira, Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proc. ACM Program. Lang.* **3** (2019) 1–29.

[23] J.H. Smith, Factoring, into edge transpositions of a tree, permutations fixing a terminal vertex. *J. Comb. Theory Ser. A* **85** (1999) 92–95.

[24] J.H. Smith, Corrigendum to "factoring, into edge transpositions of a tree, permutations fixing a terminal vertex". *J. Comb. Theory Ser. A* **118** (2011) 726–727.

[25] J. van den Heuvel, The complexity of change, in Surveys in Combinatorics. Cambridge University Press, Cambridge (2013) 127–160.

[26] T.P. Vaughan, Bounds for the rank of a permutation on a tree. *J. Comb. Math. Comb. Comput.* **30** (1991) 129–148.

[27] L. Wang and K.W. Tang, The cayley graph implementation in tinyos for dense wireless sensor networks, in 2007 Wireless Telecommunications Symposium. 2007 Thyrrenian International Workshop on Digital Communication, Italy (2007) 1–7.

[28] 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.* **586** (2015) 81–94.

[29] K. Yamanaka, T. Horiyama, J.M. Neil, D.G. Kirkpatrick, Y. Otachi, T. Saitoh, R. Uehara and Y. Uno, Swapping colored tokens on graphs, in Workshop on Algorithms and Data Structures. Springer, Victoria, BC, Canada (2015) 16.