

Integer Linear Programming Models for Sequential and Parallel Token Swaps in Graphs

Caio Henrique Segawa Tonetti

Universidade Federal de Minas Gerais

Av. Presidente Antônio Carlos, 6627 - Pampulha, Belo Horizonte - MG, 31270-901

caio.tonetti@gmail.com

Vinicius Fernandes dos Santos

Universidade Federal de Minas Gerais

Av. Presidente Antônio Carlos, 6627 - Pampulha, Belo Horizonte - MG, 31270-901

viniciussantos@dcc.ufmg.br

Sebastián Urrutia

Universidade Federal de Minas Gerais, Molde University College

Av. Presidente Antônio Carlos, 6627 - Pampulha, Belo Horizonte - MG, 31270-901

surrutia@dcc.ufmg.br

PALAVRAS CHAVE. Programação Linear Inteira, Teoria dos Grafos, Reconfiguração.

ABSTRACT

Reconfiguration problems focus on transforming a state of a combinatorial or geometric object into another by performing some sequence of operations. In one important class of reconfiguration problems, it is allowed to move items along the edges of a graph to achieve a desired final configuration. The Token Swapping problem is one of these problems and is known to be NP-hard. In this paper we present a novel approach to solving this problem and a parallel variant by using an integer linear programming model.

KEYWORDS. Integer Linear Programming. Graph Theory. Reconfiguration.

1. Introduction

The reconfiguration framework [Ito et al., 2011] brings the notion of *transformation* in computational problems and new questions arise from the necessity of understanding these changes under various operations and constraints. A simple version of this problem is sorting a list of numbers by adjacent swaps between two elements, in which the swap number is exactly the number of pairs that are out of order, being recognized as the number of *inversions* in a list and two elements are adjacent if they are next to each other in the list. This swap sequence can be calculated in polynomial time by a modified bubble sort algorithm [Knuth, 1998]. Under other conditions, when sorting the lists by *prefix-reversals*, an elementary operation that flips a prefix of the list, finding the minimum number of flips is **NP-complete** [Bulteau et al., 2015]. This problem is called the pancake sorting problem. Another 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 can *slide* a tile to any adjacent empty tile space to achieve a final sorted configuration state. Research of the 15-puzzle version of the n -puzzle dates back to the late 19th century [Johnson e Story, 1879] and is commonly used as an introductory problem for modelling heuristics.

Considering the reconfiguration problems, the interest lies in one of the three following parameters: connectivity, diameter or distance. These parameters translate, respectively, to the following questions: (a) can any configuration be reconfigured into another?; (b) what is the maximum number of required operation steps for any reconfiguration?; and (c) what is the minimum number of operations for any reconfiguration? In the examples aforementioned, the distance between two configurations (the initial and the sorted) was being used. For the interested, more information about most of the known reconfiguration problems can be found in the surveys by [van den Heuvel, 2013; Mouawad, 2015; Nishimura, 2018].

This paper focuses on a reconfiguration problem called the Token Swap problem (TS). Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $|E|$ edges, with distinct tokens placed on its vertices. The objective is to reconfigure this initial token placement called $f_0 : V \mapsto V$ into the identity token placement f_i that maps every node to itself with minimum distance. The reconfiguration must consist of a token swap sequence S , being each identified by a pair of adjacent graph vertices, meaning that the elementary operation will be restricted to a swap between two tokens that share an edge in the graph. For the decision version of this problem, the aim is to know if it is possible to have a swap sequence S that transforms f_0 to f_i in k or less swaps, such that $k \in \mathbb{N}$. In Figure 1, a complete example of a TS instance is shown. The formal definitions necessary to fully understand this paper will be given in Section 2.

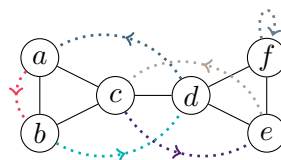


Figure 1: Example of an instance of the Token Swap problem with $V = \{a, b, c, d, e, f\}$ and tokens represented by colored dotted arrows. The arrows point from the vertex that is originally positioned in mapping f_0 to the target vertex in the identity map f_i .

Every reconfiguration problem can also be formulated as a reconfiguration graph where each node is a possible configuration of the combinatorial or geometric object and an edge exists between two vertices if and only if each of the vertices can be achieved in exactly one reconfigu-

ration step from another [Nishimura, 2018]. In the case of the TS, the reconfiguration graph will be connected as long as the original graph is also connected. This connection guarantees that any configuration can be reconfigured to any other, as it is possible to take any spanning tree of the original graph and move the desired tokens to each corresponding leaf in a way that any chosen final configuration is achieved. This process give us an upper-bound of $O(n^2)$ swaps for any instance of the TS problem [Yamanaka et al., 2015a]. The diameter and distance of this reconfiguration graph is exactly the upper-bound of the reconfiguration problem, and the distance between two configurations —i.e., two vertices in the reconfiguration graph—, respectively. 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 exponential in $|V|$.

If the TS problem is brought to the realm of permutation group theory, it is possible to model this problem by a group $(F, \circ)$, in which every element of F is a bijective function representing a possible configuration. The binary operation is function composition and every element of F can be represented by the product of finitely many elements of a subset C of F and their inverses. The elements of C are called the generators of the group $(F, \circ)$ and they represent each possible transpositions corresponding to the edges of the original graph. Thereupon, given a TS instance, the Cayley Graph $\Gamma(F, C)$ of the symmetric group will correspond (under isomorphism) exactly to the reconfiguration graph of the original problem, the distance between two configurations will be the shortest path between those two vertices and the worst case will match the diameter of this graph. The shortest path in a Cayley Graph, also known as the `Minimum Length Generator Sequence` problem, is a generalization of the TS problem, and a **PSPACE-complete** problem [Jerum, 1985]. The diameter of the Cayley Graphs has been researched in the context of transposition trees [Akers e Krishnamurthy, 1989; Cooperman e Finkelstein, 1992; Bafna e Pevzner, 1998; Ganesan, 2012a,b; Chitturi, 2013; Kraft, 2015; Chitturi e Indulekha, 2019]—i.e., transpositions generators from the edges of a tree. It resulted in many heuristic algorithms to calculate upper-bounds that do not depend on the vertex number of the Cayley Graph¹. Some application examples of TS are: computing efficient interconnection network structures where the maximum delay could be measured by calculating the diameter of the network [Annexstein et al., 1990], computational biology [Bafna e Pevzner, 1998; Heath e Vergara, 2003], model Wireless Sensor Networks (WSS) [Wang e Tang, 2007], protection routing [Pai et al., 2020] and qubit allocation for quantum computers [Siraichi et al., 2018, 2019].

The reconfiguration version of TS was first introduced in 2015 [Yamanaka et al., 2015a], and further generalizations and variations were studied in the same year [Yamanaka et al., 2015b]. The TS problem was first proved not only **NP-Complete**, and it remains even when restricted to bipartite graphs with degree bounded by 3, but also **APX-complete** [Kawahara et al., 2017; Miltzow et al., 2016] and **W[1]-hard** parameterized by number of swaps, but fixed parameter tractable (**FPT**) for the class of nowhere dense graphs. Subsequently, it was proved **NP-hard** and it remains even when both the treewidth and the diameter of the input graph are constant [Bonnet et al., 2018]. There are some special classes of graphs that can be solved through a exact polynomial time algorithm, almost all of them with good references listed in historical order by a study of TS in trees performed by [Biniaz et al., 2019]: cliques, paths, cycles, stars, brooms, lollipop [Kawahara et al., 2017], complete bipartite graphs and complete split graphs. Concerning square of paths, there is a 2-approximation algorithm [Heath e Vergara, 2003]. For trees, two 2-approximation algorithms [Yamanaka et al., 2015a; Miltzow et al., 2016] are known and proven to be tight [Biniaz et al., 2019]. Both approximation algorithms can be adapted to general graphs, providing α - and

¹In the case of the TS, the vertex number of a Cayley Graph is $O(n!)$.

4-approximation algorithms, respectively, where α is the value of an α -spanner tree of the input graph. Recently, [Bonnet et al., 2018] conjectured that TS remains **NP-Complete** even in trees and [Biniaz et al., 2019] reinforced this conjecture by showing a counterexample of the *Happy Leaf Conjecture* [Vaughan, 1991], but the complexity remains open.

The emphasis of this paper is to introduce the reader to the field of reconfiguration problems, while also focusing in the TS problem and promoting an initial discussion on modelling the problem and a parallel variant as an Integer Linear Program. In Section 2 some definitions and mathematical tools are presented. Section 3 describe the proposed model for the TS problem. Model adaptations to a parallel variant of the TS problem called `Parallel Token Swap` problem are discussed in Section 4. Section 5 gives some insights on creating formulations that uses token placement checks. And the paper is concluded in Section 6, with remarks about improvements to the models, other TS variants and what can still be done in the future.

2. Preliminaries

For an integer k , the notation $[k] := \{1, 2, \dots, k\}$ was used; and 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 $M : V \mapsto \mathbb{N}$ 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 $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E \subseteq \binom{V}{2}$. The graph is called undirected if and only if each edge $\{u, v\} \in E$ can be represented as a set, as the order of nodes do not matter. The shorthand ' uv ' is used to describe a pair $\{u, v\}$.

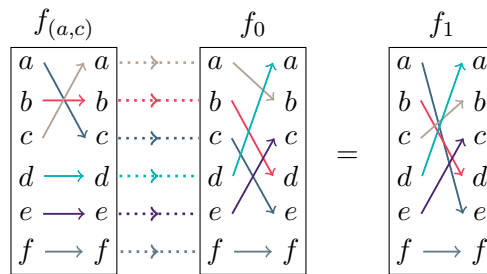


Figure 2: Let f_0 be the TS instance represented in Figure 1, $f_{(a,c)}$ be the mapping function representation of swap (a, c) and f_1 be the resulting mapping of the operation $f_0 \circ f_{(a,c)}$. Each rectangle represents a function with domain on the left, codomain on the right and color coded 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 .

An instance of the TS problem is composed of a 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 $\{u, v\} \in E$. 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 $f_s(w) = w$, $\forall w \in V \setminus \{u, v\}$, and applied to a token placement f_0 by the composition $f_0 \circ f_s = f_1$. An example on how the composition operation can swap function values is shown in Figure 2.

Let $G := (V, E)$ be a graph. It can be said that two vertices $u, v \in V$ are *connected* if there is at least one *path of vertices* $(v_1, v_2, \dots, v_k)$ in G such that $v_i v_{i+1} \in E$, $\forall i \in [k-1]$, $v_1 = u$ and $v_k = v$. A graph G is connected if all vertex pairs of V are connected. 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 graph 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 $S := (s_1, s_2, \dots, s_k)$ 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 of distance to transform one placement into another. The Token Swap problem asks for a sequence of swaps that solves a given instance with the minimal number of swaps k .

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

It is important to note that this version of the problem is equivalent to the problem of finding a swap sequence between two arbitrary mapping functions f_0, f_e by the use of the following process of token renaming from f_0 to f'_0 : if a token i has $f_0(i) = u$, the i is renamed to the token $f_e(u)$, generating an instance of the TS problem with the same graph and initial mapping $f'_0 = f_e \circ f_0$. Given a S that solves the TS instance, the vertices can just be renamed back to get a swap sequence of the initial problem.

3. The TS Problem Formulation as an Integer Programming Problem

The TS problem model given by Formulation (2)-(11) tests all possible configurations of the problem with a given upper-bound in the number of swaps T , allowing at maximum one swap per step $t \in [T]$. Each step is composed of a set of variables that describe the current configuration, which swap is being selected and Equation 9 checks if a swap sequence solves the current instance. The constant T can be calculated by using any of the best approximation algorithms, or by using the trivial upper-bound $O(n^2)$ on the size of an optimal swap sequence mentioned in Section 1.

$$\text{minimize} \quad \sum_{\forall uv \in E, u <_M v, \forall t \in [T]} y_{uvt} \quad (2)$$

$$\text{subject to} \quad \sum_{\forall u \in V} x_{iut} = 1 \quad \forall t \in [T], \forall i \in V \quad (3)$$

$$\sum_{\forall i \in V} x_{iut} = 1 \quad \forall t \in [T], \forall u \in V \quad (4)$$

$$x_{iut} + x_{iut+1} \leq y_{uvt} + 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \in E, u <_M v \quad (5)$$

$$x_{iut} + x_{iut+1} \leq y_{uvt} + 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \in E, u <_M v \quad (6)$$

$$x_{iut} + x_{iut+1} \leq 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \notin E \quad (7)$$

$$\sum_{\forall u, v \in V, u <_M v} y_{uvt} \leq 1 \quad \forall t \in [T] \quad (8)$$

$$x_{iit} = 1 \quad \forall i \in V \quad (9)$$

$$y_{uvt} \in \{0, 1\} \quad \forall t \in [T], \forall uv \in E \quad (10)$$

$$x_{iut} \in \{0, 1\} \quad \forall i \in V, \forall u \in V, \forall t \in [T] \quad (11)$$

These approximation algorithms use the same intuitive lower-bound given by

$$\sum_{v \in V} \frac{\delta(f_0(v), v)}{2}$$

where $\delta(u, v)$ is the distance from vertex u to vertex v in the graph G and the division by 2 happens since every swap can decrease the distance between tokens by at maximum two in the sum. This bound is tight for instances of TS that can be solved with only swaps that decrease the distance sum by two [Bonnet et al., 2018]. These values give us a good notion of the range of values that T can assume. More detailed explanations of each constraint are presented in the subsequent Sections 3.1 and 3.2.

3.1. Modelling the Variables

Binary variables x_{iut} determine if a token i is at node u in step t . To correct model the bijections between tokens and vertices, Equation 3 enforces that any token can be at most in one vertex and Equation 4 that a vertex can have at most one token. The binary variables y_{uv} flags if a swap happened between nodes u and v in step t . It is important to note that swaps are symmetric —i.e., a swap (u, v) for $u, v \in V$ is exactly the same as swap (v, u) —, so variables y_{uv} and y_{vu} means the same swap. On those grounds, and the fact that our graph is undirected, a technique that creates an ordering $M : V \mapsto [n]$ of the vertices of the graph and use the variables y_{uv} such that $u <_M v$ can be adopted, halving the number of variables needed to represent a swap. The ordering M is assumed to exist throughout the paper for any graph.

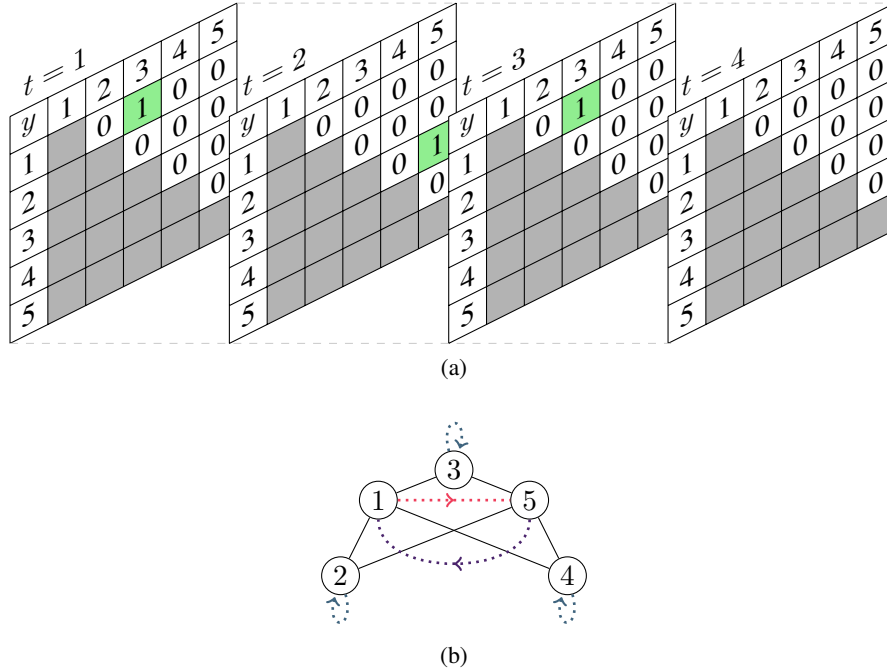


Figure 3: Here is an example of how the variable y_{uv} can model a sequence of swaps. Let $T = 4$ and the TS instance presented at Figure 3b. The minimum number of swaps needed to bring all tokens to its correct positions is three, more precisely $(1, 3), (3, 5), (3, 1)$. At Figure 3a it is shown how these swaps are represented (in green) with the fourth panel having no swaps. Note that the swap symmetry is being used to represent swaps using the upper diagonal of the matrix.

The Figure 3 is an example of a TS instance and a complete representation of an optimal swap sequence using y_{uvt} . To retrieve the final sequence, it is necessary to traverse each matrix in step order and check each y_{uvt} , appending (u, v) to the swap sequence whenever $y_{uvt} = 1$. Steps with no $y_{uvt} = 1$ can be ignored. The total number of variables is given by Equation 12.

$$\underbrace{\frac{T \times n^2 - n}{2}}_{y_{uvt}} + \underbrace{T \times n^2}_{x_{iut}} = \frac{n}{2}(3 \times T \times n - 1) \quad (12)$$

3.2. Modelling Swaps

The constraints (5)-(8) are being used to model the behavior of swaps. For a swap y_{uvt} to be possible, it is required that in step t the token on vertex u is on vertex v on step $t + 1$, given by Inequality 5. Similarly, on Inequality 6, it was guaranteed that the second token that is on vertex v on step t must be on vertex u on step $t + 1$. Note that each pair of vertices $u, v \in V$ were directly chosen from the edge set E , ensuring that an edge must exist. Inequality 7 is there to keep the consistency between each pair of vertices that do not have an edge in between, as it is impossible to have a token traverse both in one step. Inequality 8 force each step to have at maximum one swap, while also permitting steps with no swaps. In Section 4 a more in-depth discussion about a parallel variant of the TS problem that don't need to guarantee one swap per step will be presented.

4. The Parallel TS Problem Formulation as an Integer Programming Problem

The Parallel Token Swap (PTS) problem is a version of TS where swaps can be applied in parallel. For a graph $G := (V, E)$, a *parallel swap* of a set of parallel swaps $P = \{s_1, s_2, \dots, s_k\}$ is a swap s such that no other swap in P shares a vertex with s . Any swap sequence S that solves an instance of PTS or TS can be partitioned into sequences of parallel swaps $\mathcal{P}(S) = (P^0, P^1, \dots, P^b)$. The swaps of each partition can be applied to a token placement in any order without changing any intermediate step in S (hence the use of set) or it can be applied by just one representation function f_P , as all swaps in a partition have disjoint vertices. Equation 13 shows how this representation function can be built. A partitioned sequence of parallel swaps solves an instance of TS or TSP if and only if the identity function is resulted by applying each partition iteratively, while the swap application order of each partition does not matter.

$$f_P = f_{s_1} \circ f_{s_2} \circ \dots \circ f_{s_k} \quad (13)$$

It is important to note that the size of any partition has a natural upper-bound on the size of the Maximum Matching of the graph G , as it is the maximum number of edges that can be swapped in the graph and still be disjoint. Calculating the maximum matching size in general graphs can be done in polynomial time [Edmonds, 1965]. Therefore, for graphs with maximum matching size of 1, an instance of PTS has the same set of optimal swap sequences of the equivalent TS instance. For general cases, the size of a swap sequence that solves a TS instance can be used as an upper-bound for an equivalent PTS instance. This variant of the TS problem has a deeper relation to parallel sorting on SIMD machines consisting of several processors with local memory connected by a network [Yamanaka et al., 2015a; Kawahara et al., 2017] and is better suited for some reconfiguration problems, like Qubit Allocation [Siraichi et al., 2018, 2019].

$$\text{minimize} \quad \sum_{\forall t \in [T]} s_t \quad (14)$$

$$\text{subject to} \quad \sum_{\forall u \in V} x_{iut} = 1 \quad \forall t \in [T], \forall i \in V \quad (15)$$

$$\sum_{\forall i \in V} x_{iut} = 1 \quad \forall t \in [T], \forall u \in V \quad (16)$$

$$x_{iut} + x_{iut+1} \leq y_{uvt} + 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \in E, u <_M v \quad (17)$$

$$x_{iut} + x_{iut+1} \leq y_{uvt} + 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \in E, u <_M v \quad (18)$$

$$x_{iut} + x_{iut+1} \leq 1 \quad \forall i \in V, \forall t \in [T-1], \forall uv \notin E \quad (19)$$

$$\sum_{\forall u,v,w \in V, u <_M v} y_{uwt} + y_{wvt} \leq 1 \quad \forall t \in [T] \quad (20)$$

$$y_{uvt} \leq s_t \quad \forall t \in [T], \forall uv \in E, u <_M v \quad (21)$$

$$x_{iiT} = 1 \quad \forall i \in V \quad (22)$$

$$y_{uvt} \in \{0, 1\} \quad \forall t \in [T], \forall uv \in E \quad (23)$$

$$x_{iut} \in \{0, 1\} \quad \forall i \in V, \forall u \in V, \forall t \in [T] \quad (24)$$

$$s_t \in \{0, 1\} \quad \forall t \in [T] \quad (25)$$

One interesting question that naturally arises from this definition is: "what is the swap sequence that solves a given PTS instance and minimizes the number of partitions in $|\mathcal{P}(S)|$?". In the decision version, the existence of at least one swap sequence that solves a PTS instance with k or less partitions is wanted. This version of the problem has already been proven to be **NP-Complete** for general k , but can be calculated in polynomial time if $k \leq 2$ and have an approximation algorithm for paths [Kawahara et al., 2017].

The model presented by Formulation (14)-(25) is an adaptation of the TS model for the PTS problem. The variables y_{uvt} and x_{iut} are used in exactly the same way as the TS model with constraints (15)-(19) being exactly the same as constraints (3)-(7). Each new constraint will be explained in detail in Section 4.1 and Section 4.2.

4.1. Minimizing the Number of Steps

Each step is now related to a partition and a new binary variable for each step called s_t were added to keep track of which steps have *at least* one $y_{uvt} = 1$ by using Inequality 21. Note that the number of checks is improved by only verifying variables y_{uvt} such that $\{u, v\} \in E$, as swaps can only exist on the edges. These flags help counting how many steps are using at least one swap, while any step that do not need to be used to achieve a minimal swap sequence will not be counted, as $s_t = 0$. This variable and the fact that each step is now allowed to have any number of parallel swaps allows for the minimization of s_t in the minimization rule given by Equation 14, which is equivalent to minimizing the number of partitions. The total number of variables is increased by T from Equation 12.

4.2. Modelling Parallel Swaps

To allow parallel swaps in the model, an efficient way to check if two swaps are disjoint must be used. If two swaps s_1 and s_2 are not disjoint, then they have exactly one vertex in common, as it is impossible to have repeated swaps in the same step due to how the variables y_{uvt} are modeled. Thus, for any two conflicting swaps, a combination of three nodes u, v, w could be taken and assumed, without loss of generality, that w is part of these two swaps $(u, w), (w, v)$. Consequently,

the variables y_{uwt} and y_{wvt} can not be 1 at the same time in the model, as given by Inequality 20. In the case that both edges exist in the graph of an instance, these swaps would share a vertex and could not be used in the same swap. On the cases that these vertices have only one edge or no edge, there is no way two swaps can be used between these three vertices, as Inequalities (17)-(19) prohibit them.

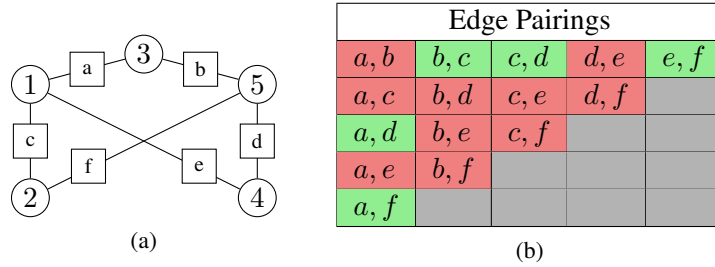


Figure 4: Let Figure 4a be a graph with vertex set $V = \{1, 2, 3, 4, 5\}$ and edge set $E = \{a, b, c, d, e, f\}$. Table 4b lists all unordered *pairs* of edges that can be swapped in a token placement at the same step with green and red otherwise. The reader can check each pairing by taking the vertex triple of any red pairing and checking in the model. All green edge pairings are vertex disjoint and there is no three edge parallel partition in this graph.

The concept was improved by the use of the inherent vertex ordering M of the graph to remove repeated vertex checks from the model. For any vertex triple u, v, w with a "center" vertex fixed as w , checking variables y_{uwt} and y_{wvt} are equivalent to checking variables y_{vwt} and y_{wvt} . Therefore, checking variables for triples u, w, v such that $u <_M v$ is enough and remove useless checkings. Figure 4 give us an example to compare swap sequence representations between TS and PTS models.

5. Observations on Realizing Token Placements Checks

Let f be a token placement map of a TS instance. The sum of the distances $\delta(f(u), u)$ 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 is 0 for the resulting placement $f_{|S|}$. As a result, 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 its 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 [Smith, 1999] tried to prove, but subsequently found an error [Smith, 2011]. Even when thinking of a TS instance in a tree, [Vaughan, 1991] 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*². This conjecture was disproved by [Biniaz et al., 2019], as they shown that there is a class of infinite trees that necessarily need (c) swaps to achieve the optimal number of swaps 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. The two

²The original paper actually claimed a stronger conjecture with the Happy Leaf Conjecture as a special case.

approximative algorithms cited in Section 1 can only generate swap sequences of category (a) and (b).

$$\sum_{\forall i \in V, \forall u \in V, \forall v \in V, u <_M v} (d_{iu} \times x_{iut} - d_{iv} \times x_{ivt+1}) \geq 0 \quad \forall t \in [T-1] \quad (26)$$

For a given step t , it was presumed that there is no guarantee that an adjacent placement will increase or decrease the sum in step $t + 1$. In such manner, constraints that keeps track of this sum in every intermediate placement do not look viable for general graphs and trees. Equation 10 and Equation 22 were chosen due to the simplicity of pushing the placement verification to the furthest step possible, when $t = T$, and allowing the existence of steps of any swap category. Let $d_{uv} \in \mathbb{N}_0$ be a constant that represents the distance between any two vertices $u, v \in V$. If there is a class of instances that can be proved to not have swaps of category (c), the Inequality 26 could be added to the model to cut off token placements that increase the sum of the distances, but the tradeoff of pre-calculating this distance would have to be considered.

6. Future Work

This paper presented an introduction to the `Token Swap` problem and one parallel variant, the `Parallel Token Swap` problem, along with an integer linear model for each problem and a detailed discussion about the constraints. The next step would be the programming and executing each of these models to check their viability and compare with other approaches. Moreover, these models could be adapted to other TS variations, such as the `Colored Token Swap` problem and the `Parallel Colored Token Swap` problem. Generalizations of the TS problem and PTS problem, respectively, where tokens can have more than one target vertex.

The presented models could still be improved with testing. More constraints would focus on the development of more valid inequalities, like checking steps with repeated configurations or constraints to eliminate the token symmetry of the model, pushing steps with no swaps to the right-most step possible. Some of those were already achieved, but omitted for the sake of conciseness. On future works, with experimental results, we will evaluate whether they provide improvements in running time. Another route would be the creation of specialty models focused in some useful and hard classes of graphs like trees, as more specific constraints could be helpful in getting faster answers.

References

- Akers, S. B. e Krishnamurthy, B. (1989). A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38:555–566.
- Annexstein, F., Baumslag, M., e Rosenberg, A. L. (1990). Group action graphs and parallel architectures. *SIAM Journal on Computing*, 19:544–569.
- Bafna, V. e Pevzner, P. A. (1998). Sorting by transpositions. *SIAM J. Discret. Math.*, 11:224–240.
- Biniaz, A., Jain, K., Lubiw, A., Masárová, Z., Miltzow, T., Mondal, D., Naredla, A. M., Tkadlec, J., e Turcotte, A. (2019). Token swapping on trees. *CoRR*, abs/1903.06981:41.
- Bonnet, É., Miltzow, T., e Rzażewski, P. (2018). Complexity of token swapping and its variants. *Algorithmica*, 80:2656–2682.

- Bulteau, L., Fertin, G., e Rusu, I. (2015). Pancake flipping is hard. *Journal of Computer and System Sciences*, 81:1556 – 1574.
- Chitturi, B. e Indulekha, T. (2019). Sorting permutations with a transposition tree. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, p. 1–4, Bahrain. IEEE.
- Chitturi, B. (2013). Upper bounds for sorting permutations with a transposition tree. *Discrete Mathematics Algorithms and Applications*, 5:24.
- Cooperman, G. e Finkelstein, L. (1992). New methods for using cayley graphs in interconnection networks. *Discrete Applied Mathematics*, 37-38:95 – 118.
- Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467.
- Ganesan, A. (2012a). Diameter of cayley graphs generated by transposition trees. *CoRR*, abs/1202.5888:11.
- Ganesan, A. (2012b). On the strictness of a bound for the diameter of cayley graphs generated y transpositions trees. In *Proceedings of the International Conference on Mathematical Modelling & Scientific Computation*, p. 54–61, Cham. Springer.
- Heath, L. e Vergara, J. (2003). Sorting by short swaps. *Journal of computational biology : a journal of computational molecular cell biology*, 10:775–89.
- Ito, T., Demaine, E. D., Harvey, N. J., Papadimitriou, C. H., Sideri, M., Uehara, R., e Uno, Y. (2011). On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412: 1054 – 1065.
- Jerrum, M. R. (1985). The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265 – 289.
- Johnson, W. W. e Story, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, 2:397–404.
- Kawahara, J., Saitoh, T., e Yoshinaka, R. (2017). The time complexity of the token swapping problem and its parallel variants. In *WALCOM: Algorithms and Computation*, p. 448–459, Cham. Springer International Publishing.
- Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., USA.
- Kraft, B. (2015). Diameters of cayley graphs generated by transposition trees. *Discrete Applied Mathematics*, 184:178 – 188.
- Miltzow, T., Narins, L., Okamoto, Y., Rote, G., Thomas, A., e Uno, T. (2016). Tight exact and approximate algorithmic results on token swapping. *CoRR*, abs/1602.05150:19.
- Mouawad, A. E. (2015). *On Reconfiguration Problems: Structure and Tractability*. PhD thesis, University of Waterloo.
- Nishimura, N. (2018). Introduction to reconfiguration. *Algorithms*, 11:52.

- Pai, K.-J., Chang, R.-S., e Chang, J.-M. (2020). Constructing dual-cists of pancake graphs and performance assessment of protection routings on some cayley networks. *The Journal of Supercomputing*, 76:124546.
- Siraichi, M. Y., Santos, V. F. d., Collange, C., e Pereira, F. M. Q. a. (2019). Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proc. ACM Program. Lang.*, 3:1–29.
- Siraichi, M. Y., Santos, V. F. d., Collange, S., e Pereira, F. M. Q. (2018). Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, p. 113–125, New York, NY, USA. ACM.
- Smith, J. H. (1999). Factoring, into edge transpositions of a tree, permutations fixing a terminal vertex. *Journal of Combinatorial Theory, Series A*, 85:92–95.
- Smith, J. H. (2011). Corrigendum to "factoring, into edge transpositions of a tree, permutations fixing a terminal vertex". *J. Comb. Theory, Ser. A*, 118:726–727.
- van den Heuvel, J. (2013). The complexity of change. In *Surveys in Combinatorics*, p. 127–160, Cambridge. Cambridge University Press.
- Vaughan, T. P. (1991). Bounds for the rank of a permutation on a tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 30:129–148.
- Wang, L. e Tang, K. W. (2007). The cayley graph implementation in tinyos for dense wireless sensor networks. In *2007 Wireless Telecommunications Symposium*, p. 1–7, Italy. 2007 Thyrranian International Workshop on Digital Communication.
- Yamanaka, K., Demaine, E. D., Ito, T., Kawahara, J., Kiyomi, M., Okamoto, Y., Saitoh, T., Suzuki, A., Uchizawa, K., e Uno, T. (2015a). Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81 – 94.
- Yamanaka, K., Horiyama, T., Neil, J. M., Kirkpatrick, D. G., Otachi, Y., Saitoh, T., Uehara, R., e Uno, Y. (2015b). Swapping colored tokens on graphs. In *Workshop on Algorithms and Data Structures*, p. 16, Victoria, BC, Canada. Springer.