Hardness of Token Swapping on Trees

Oswin Aichholzer
Technische Universität Graz, Austria

Erik D. Demaine
CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Matias Korman
Siemens Electronic Design Automation, Wilsonville, OR, USA

Anna Lubiw
Cheriton School of Computer Science, University of Waterloo, Canada

Jayson Lynch
Cheriton School of Computer Science, University of Waterloo, Canada

Zuzana Masárová
IST Austria, Klosterneuburg, Austria

Mikhail Rudoy
LeapYear Technologies, San Francisco, CA, USA

Virginia Vassilevska Williams
CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Nicole Wein
DIMACS, Rutgers University, Piscataway, NJ, USA

Abstract

Given a graph where every vertex has exactly one labeled token, how can we most quickly execute a given permutation on the tokens? In ***(sequential) token swapping***, the goal is to use the shortest possible sequence of ***swaps***, each of which exchanges the tokens at the two endpoints of an edge of the graph. In ***parallel token swapping***, the goal is to use the fewest ***rounds***, each of which consists of one or more swaps on the edges of a matching. We prove that both of these problems remain NP-hard when the graph is restricted to be a tree.

These token swapping problems have been studied by disparate groups of researchers in discrete mathematics, theoretical computer science, robot motion planning, game theory, and engineering. Previous work establishes NP-completeness on general graphs (for both problems), constant-factor approximation algorithms, and some poly-time exact algorithms for simple graph classes such as cliques, stars, paths, and cycles. Sequential and parallel token swapping on *trees* were first studied over thirty years ago (as "sorting with a transposition tree") and over twenty-five years ago (as "routing permutations via matchings"), yet their complexities were previously unknown.

We also show limitations on approximation of sequential token swapping on trees: we identify a broad class of algorithms that encompass all three known polynomial-time algorithms that achieve the best known approximation factor (which is 2) and show that no such algorithm can achieve an approximation factor less than 2.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Sorting, Token swapping, Trees, NP-hard, Approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.3

Related Version *Full Version*: `https://arxiv.org/abs/2103.06707`

Funding *Anna Lubiw*: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).
Jayson Lynch: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

© Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein;
licensed under Creative Commons License CC-BY 4.0

Zuzana Masárová: Supported by Wittgenstein Prize, Austrian Science Fund (FWF), grant no. Z 342-N31.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.
Nicole Wein: Supported by a grant to DIMACS from the Simons Foundation (820931). This work was done while the author was at MIT.

Acknowledgements This research was initiated at the 34th Bellairs Winter Workshop on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint, held on March 22–29, 2019 in Holetown, Barbados. We thank the other participants of that workshop for providing a stimulating research environment.

1 Introduction

Imagine n distinctly labeled tokens placed without collisions on the n vertices of a graph G. For example, these n tokens might represent (densely packed) movable ***agents*** – robots, people, packages, shipping containers, data packets, etc. – while the n vertices represent possible agent locations. Now suppose we want to move the tokens/agents around, for example, to bring certain shipping containers to the loading side of a cargo ship. In particular, we can suppose every token has a given start vertex and destination vertex, and the goal is to move every token to its desired destination. Because every vertex has a token (agents are densely packed), a natural reconfiguration operation is to ***swap*** two adjacent tokens/agents, that is, to exchange the tokens on the two endpoints of a given edge in G. In this paper, we study token reconfiguration by swaps from a given start configuration to a given destination configuration with two natural objective functions:

1. **(Sequential) Token Swapping** (a.k.a. "sorting with a transposition graph" [2]): Minimize the number of swaps, i.e., the total work required to reconfigure.
2. **Parallel Token Swapping** (a.k.a. "routing permutations via matchings" [3]): Minimize the number of rounds of simultaneous swaps (where the edges defining the swaps form a matching, so avoid conflicting shared endpoints), i.e., the total execution time or makespan required to reconfigure.

These reconfiguration problems can be cast in terms of the symmetric group. Each possible reconfiguration step – swapping along one edge in the sequential problem, or swapping along every edge of a matching in the parallel problem – is a particular permutation on the n tokens (an element of the symmetric group S_n). Assuming the graph is connected, these permutations generate S_n, defining a ***Cayley graph*** C [9] where each node π in C corresponds to a permutation π of the tokens (a collision-free placement of the tokens) and an undirected edge connects two nodes π_1, π_2 in C if there is a reconfiguration step (swapping an edge or matching in G) that transforms between the two corresponding permutations π_1, π_2. Minimizing the number of reconfiguration steps between two configurations of the tokens (sequential/parallel token swapping) is equivalent to finding the shortest path in the Cayley graph between two given nodes corresponding to two given permutations in S_n. In fact, sequential token swapping was first studied by Cayley in 1849 [8] who (before inventing the Cayley graph) solved the problem on a clique, i.e., without any constraint on which tokens can be swapped.

Since its introduction, token swapping has been studied by many researchers in many disparate fields, from discrete mathematics [8, 25, 27, 29, 24, 28, 21] and theoretical computer science [17, 20, 13, 30, 32, 4, 23, 7, 31, 11, 18, 10, 6] to more applied fields including network engineering as mentioned earlier [2], robot motion planning [12, 26], and game theory [16].

What is the complexity of token swapping? In general, it is PSPACE-complete to find a shortest path between two given nodes in a Cayley graph defined by given generators [17]. But when the generators include transpositions (single-swap permutations) as in both sequential and parallel token swapping, $O(n^2)$ swaps always suffice [30], so the token-swapping problems are in NP. Both sequential token swapping [23] and parallel token swapping [5, 18] are known to be NP-complete on a general graph.

Sequential token swapping on general graphs is also known to be APX-hard [23], and even W[1]-hard with respect to the number of swaps [7]. For the special case of graphs with constant treewidth and constant diameter, sequential token swapping is also known to be NP-hard [7]. Additionally, for the special case of trees, but for the variant where the tokens have "weights" and "colors" sequential token swapping is known to be NP-hard [6]. From the algorithms side, there is a 4-approximation for sequential token swapping in general graphs [23]. Polynomial-time exact algorithms are known for a number of special classes of graphs including cliques [8], paths [20], cycles [17], stars [25, 24], brooms [28, 18, 6], complete bipartite graphs [30], and complete split graphs [32]. The problem is also known to be fixed parameter tractable (where the parameter is the number of swaps) on nowhere dense graphs, which includes planar graphs and graphs of bounded treewidth [7]. See also the surveys by Kim [19] and Biniaz et al. [6].

In this paper, we study the special case when the underlying graph is a tree. Sequential token swapping on a tree was first studied over thirty years ago, even before the problem was studied on general graphs. Akers and Krishnamurthy [2] studied the problem in the context of interconnection networks. Specifically, they proposed connecting processors together in a network defined by a Cayley graph, in particular a Cayley graph of transpositions corresponding to edges of a tree (what they call a ***transposition tree***), so the shortest-path problem naturally arises when routing network messages. They gave an algorithm for finding short (but not necessarily shortest) paths in the resulting Cayley graphs, and characterized the diameter of the Cayley graph (and thus found optimal paths *in the worst case* over possible start/destination pairs of vertices) when the tree is a star. Follow-up work along this line attains tighter upper bounds on the diameter of the Cayley graph in this situation when the graph is a tree [27, 13, 21, 11] and develops exponential algorithms to compute the exact diameter of the Cayley graph of a transposition tree [10], though the complexity of the latter problem remains open.

Sequential token swapping on a tree is the related problem of computing the shortest-path distance between two given nodes in the Cayley graph of a transposition tree. For sequential token swapping on a tree, the literature exhibits a curious phenomenon whereby there are three 2-approximation algorithms that were all developed independently and all use completely different techniques. These algorithms are by Akers and Krishnamurthy [2] in 1989, Vaughan and Portier [29] in 1995, and Yamanaka et al. [30] in 2015. No better approximation factor than 2 is known.

Parallel token swapping was also introduced in the context of network routing: in 1994, Alon, Chung, and Graham [3] called the problem "routing permutations via matchings". They focused on worst-case bounds for a given graph (the diameter of the Cayley graph); in particular, they proved that any n-vertex tree (and thus any n-vertex connected graph) admits a solution with less than $3n$ rounds, a bound later improved to $\frac{3}{2}n + O(\log n)$ [33]. Like sequential token swapping, computing the exact diameter of the Cayley graph of a given tree remains open.

Parallel token swapping on a tree is the related problem of computing the shortest-path distance between two given nodes in such a Cayley graph. Parallel token swapping is known to be NP-complete in bipartite maximum-degree-3 graphs, NP-complete even when restricted

to just three rounds, but polynomial-time when restricted to one or two rounds, but NP-complete again for "colored" tokens restricted to two rounds [5, 18]. Two approximation results are known: an additive approximation for paths which uses only one extra round [18], and a multiplicative $O(1)$-approximation for the $n \times n$ grid graph [12].[1] For other special graph classes, there are tighter worst-case bounds on the diameter of the Cayley graph [3, 22, 5].

1.1 Our Results

There have been many attempts to understand token swapping on a tree, but all have fallen short of determining its actual complexity. To summarize the previously stated results for sequential token swapping on a tree, there are three known 2-approximation algorithms, and no better approximation known. There are also exact algorithms for several special cases of trees, with the most general case being a broom (a path attached to a star). From the hardness side, attempts to prove that the problem is NP-complete have led to NP-completeness proofs for more general cases. In particular, token swapping on graphs of constant treewidth and diameter is NP-hard [7], and the "weighted, colored" variant of token swapping on trees is NP-hard [6]. This leads to our first main question:

Question: Is sequential token swapping on a tree NP-complete?

This question has been implicit since sequential token swapping on a tree was first studied over 30 years ago, and the question has been explicitly stated by Biniaz et al. [6] and by Bonnet et al. [7] who conjectured that the answer is yes.

We resolve this question in the affirmative by providing a proof that sequential token swapping on a tree is NP-complete.

Next, we turn to the approximability of token swapping on a tree. The fact that there were three independently discovered 2-approximation algorithms, and nothing better is known, suggests that perhaps there is some barrier at approximation factor 2. This leads to our second main question:

Question: Is there an inherent barrier to obtaining a $(2-\varepsilon)$-approximation for sequential token swapping on trees?

We address this question by showing that there is indeed a restriction on the *types* of algorithms that can achieve approximation factor better than 2. To motivate the class of algorithms we rule out, it helps to examine known algorithms. Specifically, it was previously known that neither Akers and Krishnamurthy's "happy swap" algorithm [2] nor Yamanaka et al.'s cycle algorithm [30] can possibly achieve an approximation ratio better than 2 [6]. These two algorithms share a natural property: every token t always remains within distance 1 of the shortest path from t's start vertex to t's destination vertex. A natural question is, can a better-than-2 approximation be achieved if one allows tokens to deviate from their shortest paths more, say to distance 10 or 100?

Motivated by this question, we define an ***ℓ-straying*** algorithm as an algorithm that never moves a token a distance more than ℓ from its shortest path. We prove a surprisingly strong limitation on ℓ-straying algorithms: any less-than-2-approximation algorithm for sequential

[1] The results of [12] are phrased in terms of motion planning for robots, and in terms of a model where an arbitrary disjoint collection of cycles can rotate one step in a round. However, the techniques quickly reduce to the model of swapping disjoint pairs of robots, so they apply to parallel motion planning as well. They show that there is always a solution within a constant factor of the obvious lower bound on the number of rounds: the maximum distance between any token's start and destination.

token swapping on trees must in general bring a token *arbitrarily far* – an $\Omega(n^{1-\varepsilon})$ distance away – from its shortest path. That is, no ℓ-straying algorithm for $\ell = o(n^{1-\varepsilon})$ can achieve better than a 2-approximation.

The other known 2-approximation algorithm (besides [2] and [30]), is the Vaughan-Portier algorithm [29], which in fact *does* move tokens arbitrarily far from their shortest paths. That is, our result on ℓ-straying algorithms does not imply a limitation on the Vaughan-Portier algorithm. To address this, we also obtain the first proof that the Vaughan-Portier algorithm [29] is no better than a 2-approximation; the best previous lower bound for its approximation factor was $\frac{4}{3}$ [6]. Thus, none of the known algorithms or even their generalizations can improve upon the approximation factor of 2.

For parallel token swapping on a tree, less is known than for the sequential version. In particular, there is no known approximation algorithm nor is there any known hardness for tree-like graphs. Thus, the complexity of this problem is completely unclear. This leads to our third main question:

Question: What is the complexity of parallel token swapping on a tree?

We address this question by showing that parallel token swapping on a tree is NP-hard. In summary, our results are as follows:

1. Sequential token swapping is NP-complete on trees.
2. Parallel token swapping is NP-complete on trees, even on subdivided stars.
3. Limitations on known techniques for approximating sequential token swapping on trees:
   a) No ℓ-straying algorithm for any $\ell = O(n^{1-\varepsilon})$ can achieve better than a 2-approximation.
   b) The Vaughan-Portier algorithm does not achieve better than a 2-approximation.

1.2 Our Techniques

NP-hardness of sequential token swapping on trees

Our NP-hardness proof for sequential token swapping on trees is our most technical and conceptually difficult result. Prior work has built towards this result by providing NP-hardness for generalizations of the problem, but there appear to be barriers against extending these techniques. In the following, we briefly review this prior work and compare it to our own.

Token swapping on trees is known to be NP-hard for the variant where tokens have weights as well as "colors" [6]. However, the use of weights and colors appears to be crucial to the reduction. Token swapping is also known to be NP-hard on graphs with treewidth 2 and diameter 6 [7]. In particular, the graph in this construction is almost a tree in the sense that if you remove a single vertex the remaining graph is a forest. However, this single vertex has very high degree and is crucial to the construction. Given the apparent barriers against extending these known approaches to token swapping on trees, we take a completely different approach.

We reduce from the ***permutation generation*** problem in Garey and Johnson [14, MS6] (also called the "word problem for products of symmetric groups" (WPPSG) in [15]). In comparison, the above prior work [6, 7] reduces from the vertex cover problem, and the 3-dimensional matching problem, respectively. We observe that the permutation generation problem has a similar feel to token swapping, as it can be recast in terms of a token-swapping reachability problem as follows:

> **Star Subsequence Token-Swapping Reachability (Star STS):** Given a star graph with center vertex 0 and leaves $1, 2, \ldots, m$, where vertex i initially has a token i; given a target permutation π of the tokens; and given a sequence of swaps $s_1, s_2, \ldots, s_n$, where $s_j \in \{1, \ldots, m\}$ indicates a swap on edge $(0, s_j)$, is there a subsequence of the given swaps that realizes π?

As a first step towards reducing from Star STS to token swapping on trees, we reduce to *weighted* token swapping on trees, where each token has a non-negative integer *weight*, and the cost of a swap is the sum of the weights of the two tokens being swapped. Our reduction contains only tokens of weight 0 or 1. That is, the tokens of weight 0 are free to move, while the tokens of weight 1 cost to move. Our reduction from Star STS to 0/1-weighted token swapping on trees is quite simple. It is presented in Section 2.

The situation becomes much more complicated when we extend this result from the 0/1-weighted setting to the unweighted setting. Now, we need to *simulate* the weight-0 tokens using unweighted tokens. This introduces several complications.

First, we will describe why weight-0 tokens are integral to our reduction from Star STS to 0/1-weighted token swapping on trees. The Star STS problem asks whether there *exists* a subsequence of swaps that realizes the target permutation π. This subsequence could contain any number of swaps. In our reduction to 0/1-weighted token swapping, the swaps from this subsequence are represented using tokens of weight 0. This way, if there is a solution to the Star STS instance, then the cost of the 0/1-weighted token swapping is the same *regardless* of how many swaps occurred in the solution to the Star STS instance. This introduces a challenge for unweighted token swapping for the following reason. For 0/1 weighted token swapping, we prove a statement of the form "if the token swapping cost is exactly K then there is a solution to the Star STS instance", while for unweighted token swapping, we prove a statement of the form "if the token swapping cost is within a particular *range* then there is a solution to the Star STS instance". The second statement is much more difficult to prove because we need to argue that the additional swaps in this range do not allow the tokens to move around in a clever way to admit a solution even when there is no Star STS solution. In fact, as we discuss next, natural modifications of the weighted construction *do* admit such clever ways to create counterexamples.

The most basic first attempt to remove the weights from the weighted construction is simply to replace all weight-0 tokens with unweighted tokens. This construction admits a straightforward counterexample due to the increased cost of swapping these formerly weight-0 tokens. Thus, we would like to make the contribution of the formerly weight-0 tokens negligible in comparison to the weight-1 tokens. A natural attempt is to replace each weight-1 token with a *long path* of tokens. However, as it turns out, there is a surprising and subtle counterexample to this strategy. To overcome this counterexample, we introduce a set of "padding tokens" throughout the graph whose role is to block any deviant movement of the original tokens.

The resulting proof is very involved. To give a sense of the complexity, our 0/1-weighted hardness proof fits in just a couple of pages, while our unweighted proof spans around thirty pages. Our unweighted proof is presented in Section 3.

NP-hardness of parallel token swapping on trees

We prove that parallel token swapping on trees is NP-hard, even when restricted to ***subdivided stars***. This result is presented in the extended version of this paper [1]. As for sequential token swapping, we reduce from the Star STS problem.

Our construction is reminiscent of our construction for sequential token swapping, although the details differ significantly. In particular, we use the single high-degree vertex in the subdivided star as a bottleneck to limit the available parallelism. We develop "enforcement" tokens that need to swap through the high-degree vertex to force congestion at specific times. This proof's complexity is between the weighted and unweighted sequential hardness proofs.

Limitations on known techniques for approximation algorithms

To prove that neither an ℓ-straying algorithm nor the Vaughan-Portier algorithm can achieve better than a 2-approximation, we use a problem instance that has been previously used to prove that Akers and Krishnamurthy's and Yamanaka et al.'s algorithms cannot achieve an approximation ratio better than 2 [6]. To prove our results, we show that while there exists a solution to the instance with K swaps (for some K), (1) every ℓ-straying algorithm performs $2K$ swaps, and (2) the Vaughan-Portier algorithm performs $2K$ swaps. The existence of a solution with K swaps was already shown by [6], so it remains to show that the above algorithms require $2K$ swaps.

We emphasize that the proofs in [6] for Akers and Krishnamurthy's and Yamanaka et al.'s algorithms, as well as our proof for the Vaughan-Portier algorithm, are for *specific* algorithms, while our proof for ℓ-straying algorithms shows limitations against a very wide class of possible algorithms. Thus, our proof for ℓ-straying algorithms requires much more general reasoning about how tokens can possibly move around the graph.

2 Weighted sequential token swapping on trees is NP-hard

In this section, we prove that weighted token swapping on a tree is NP-hard, even when the token weights are in $\{0, 1\}$. Our purpose is to introduce the general idea that is used in our main NP-completeness proof for the unweighted case given in Section 3.

We first precisely define the decision problem ***weighted sequential token swapping on trees*** [6], abbreviated in this section to ***weighted token swapping***. The input consists of: a tree on n vertices with distinct initial positions (vertices) and distinct target positions for the n tokens; non-negative integer weights on the tokens; and a maximum cost K. The cost of a swap is the sum of the weights of the two tokens involved. The decision problem asks whether there is a sequence of swaps that moves all the tokens to their target positions and such that the sum of the costs of the swaps is at most K.

It is not clear whether the weighted token swapping problem lies in NP; however, it is in NP if K is given in unary.[2]

We prove that weighted token swapping is NP-hard when K is given in unary. The reduction uses Star STS. In order to distinguish tokens and vertices in the original star from those in the tree that we construct, we will call the tokens of the Star STS instance ***items*** and we will call the leaves ***slots***. Then the instance of Star STS consists of: a star with center 0 and slots $1, \ldots, m$, each of which initially has an item of its same label; a permutation π of the items; and a sequence $s_1, \ldots, s_n$ of slots that specify the allowed swaps. Figure 1(a) shows an example input for $m = 4$ slots, 5 items and a sequence of length $n = 7$.

▶ **Theorem 1.** *Weighted token swapping on trees is NP-hard.*

[2] Consider a minimum length swap sequence of weight at most K. The number of swaps involving a nonzero weight token is at most K. Because the sequence has minimum length, it can be shown that no two zero-weight tokens swap more than once. Thus the sequence has length at most $K + n^2$ and provides a polynomial-size certificate, showing that the problem is in NP.

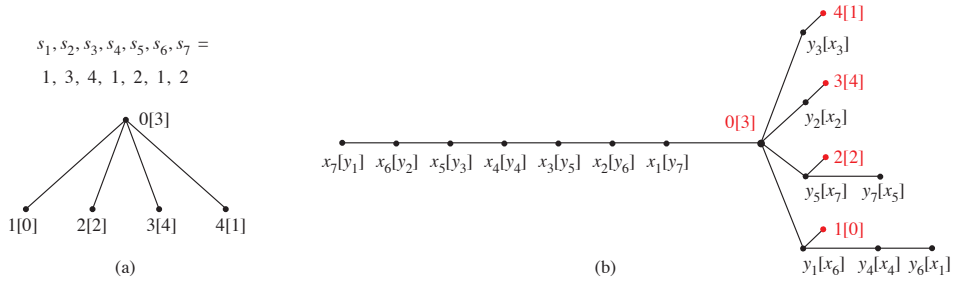

Figure 1 (a) An instance of Star STS with $m = 4$ slots and a sequence of length $n = 7$; notation $a[b]$ indicates that token a is initially at this vertex and token b should move to this vertex. This instance has no solution because item 4 should move to slot 3, which is possible only if slot 3 appears after slot 4 in the sequence. (b) The corresponding instance of weighted token swapping with the ordering gadget on the left and $m = 4$ slot gadgets attached to the root, each with a nook vertex shown in red. After swapping item token 0 at the root with token y_1 in the first (bottom) slot gadget there is an opportunity to swap tokens 0 and 1 for free along the nook edge of the first slot gadget, before moving y_1 to its target position at the end of the ordering gadget and moving x_1 to its target position at the end of the first slot gadget.

Proof. Suppose we are given an instance of Star STS as described above. We may assume without loss of generality that every slot appears in the sequence (otherwise remove that slot from the problem) and that no slot appears twice in a row in the sequence.

Construct a tree with a root, an ***ordering gadget*** which is a path of length n attached to the root, and m ***slot gadgets*** attached to the root. Slot gadgets are defined below. See Figure 1(b) where the ordering gadget of length $n = 7$ appears on the left and there are $m = 4$ slot gadgets attached to the root. We picture the tree with the root in the middle, and use directions left/right as in the figure.

Let n_i be the number of occurrences of slot i in the input sequence. Observe that $\sum_{i=1}^{m} n_i = n$. ***Slot gadget*** i consists of a path of n_i vertices, plus an extra leaf attached to the leftmost vertex of the path. This extra leaf is called the ***nook*** of the slot gadget. The m nooks and the root are in one-to-one correspondence with the slots and the center of the original star (respectively), and we place ***item tokens*** at these vertices whose names, initial positions, and final positions correspond exactly to those of the items of the input star. These item tokens are given a weight of 0.

There are $2n$ additional ***non-item tokens*** $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. These all have weight 1. The x_j's are initially placed along the ordering gadget path, in order, with x_1 at the right and x_n at the left. The ordering path is also the target position of the y_j's in reverse order with y_1 at the left and y_n at the right.

Suppose slot i appears in the sequence as $s_{j_1}, s_{j_2}, \ldots, s_{j_{n_i}}$ with indices in order $j_1 < j_2 < \cdots < j_{n_i}$. Then tokens $y_{j_1}, y_{j_2}, \ldots, y_{j_{n_i}}$ are initially placed along the path of slot gadget i, in order with smallest index at the left. The path of slot gadget i is also the target position of the tokens $x_{j_1}, x_{j_2}, \ldots, x_{j_{n_i}}$ in reverse order with smallest index at the right.

Consider, for each non-item token x_j or y_j, the distance from its initial location to its target location. This is a lower bound on the cost of moving that token. We set the max cost K to the sum of these lower bounds. This guarantees that every x_j or y_j only travels along its shortest path. Observe that this reduction takes polynomial time. We now prove that a YES instance of Star STS maps to a YES instance of token swapping and vice versa.

YES instance of Star STS. Suppose the Star STS instance has a solution. The "intended" solution to the token swapping instance implements each s_j for $j = 1, \ldots, n$ as follows. Suppose $s_j = i$. By induction on j, we claim that token y_j will be in the leftmost vertex of slot gadget i when it is time to implement s_j. Swap token y_j with the item token t currently at the root. Then item token t has the opportunity to swap for free with the item token in nook i. We perform this free swap if and only if swap s_j was performed in the solution to Star STS. Next, swap tokens y_j and x_j – we claim by induction that x_j will be in the rightmost vertex of the ordering gadget. Finally, move token y_j to its target position in the ordering gadget, and move x_j to its target position in slot gadget i. It is straightforward to verify the induction assumptions. Every x_j and y_j moves along its shortest path so the cost of the solution is equal to the specified bound K.

YES instance of weighted token swapping. Suppose the weighted token swapping instance has a solution with at most K swaps. Because K is the sum of the distances of the non-item tokens from their target positions, each x_j and y_j can only move along the shortest path to its target position. Token x_j must move into the root before x_{j+1}, otherwise they would need to swap before that, which means moving x_j the wrong way along the ordering gadget path. Similarly, y_j must move into the root before y_{j+1}, otherwise they need to swap after that, which means moving y_{j+1} the wrong way.

Furthermore, x_j must move into the root before y_{j+1} otherwise the ordering gadget would contain $x_j, \ldots, x_n$, and $y_1, \ldots, y_j$, a total of $n + 1$ tokens, which is more tokens than there are vertices in the ordering gadget.

We also claim y_j must move into the root before x_{j+1}. The nooks can only contain item tokens, since no x_j or y_j can move into a nook. This accounts for every item token except for one "free" item token. Now suppose x_{j+1} moves into the root before y_j. Then the ordering gadget contains $x_{j+2}, \ldots, x_n$, and $y_1, \ldots, y_{j-1}$, a total of $n - 2$ tokens. Even if the free item token is in the ordering gadget, there are not enough tokens to fill the ordering gadget.

Thus the x_j's and y_j's must use the root in order, first x_1 and y_1 in some order, then x_2 and y_2 in some order, etc. Finally, we examine the swaps of item tokens. A swap between two item tokens can only occur when the free item token is at the parent of a nook. Suppose this happens in slot gadget i. Then some token y_j must have left the slot gadget, and the corresponding token x_j has not yet entered the slot gadget, which means that neither of the tokens y_{j+1} or x_{j+1} has moved into the root. This implies that any swap of item tokens is associated with a unique $s_j = i$, and such swaps must occur in order of j, $1 \leq j \leq n$. Thus the swaps of item tokens can be mimicked by swaps in the original Star STS sequence, and the Star STS instance has a solution. ◀

3 Sequential token swapping on trees is NP-complete

Recall that the token swapping problem is a decision problem: given a tree with initial and target positions of the tokens, and given a non-negative integer K, can the tokens be moved from their initial to their target positions with at most K swaps. Membership in NP is easy to show: any problem instance can always be solved with a quadratic number of swaps by repeatedly choosing a leaf and swapping its target token to it. Thus, a certificate can simply be the list of swaps to execute. Here we give an overview of the construction for the unweighted case; however, most of the proof is left to the full version [1].

3.1 Overview

We reduce from Star STS and follow the same reduction plan as for the weighted case, building a tree with an ordering gadget and m slot gadgets, each with a nook for an item token. However, there are several challenges. First of all, the swaps with item tokens are no longer free, so we need to take their number into account. Secondly, we cannot know the number of item token swaps precisely since it will depend on the number of swaps (between 0 and n) required by the original Star STS instance. Our plan is to make this "slack" n very small compared to the total number of swaps needed for the constructed token swapping instance. To do this, we will make the total number of swaps very big by replacing each of the non-item tokens x_j and y_j by a long path of non-item tokens called a "segment." This raises further difficulties, because nothing forces the tokens in one segment to stay together, which means that they can sneak around and occupy nooks, freeing item tokens to swap amongst themselves in unanticipated ways. To remedy this we add further "padding segments" to the construction. The construction details are given in Section 3.2.

Proving that our unweighted construction is correct is much more involved than in the weighted case. There are two parts to the proof.

Part 1: YES instance of Star STS → YES instance of token swapping. In the full version [1] we show that if there is a solution to an instance of Star STS then the constructed token swapping instance can be solved with at most K swaps (where K will be specified in the construction). This solution uses exactly $H = K - n$ swaps to get the non-item tokens to their target positions, and uses between 0 and n additional swaps to get the item tokens to their target positions. Note that H counts swaps of both item and non-item tokens and is more than just the cost of moving each non-item token along its shortest path.

Part 2: YES instance of token swapping → YES instance of Star STS. In the full version [1] we show that if the constructed token swapping instance can be solved with at most K swaps then the original Star STS instance has a solution. We show that H swaps are needed to get the non-item tokens to their destinations. We then show that with only n remaining swaps, the motion of item tokens is so constrained that they must behave "as intended" and therefore correspond to swaps in the original Star STS instance.

3.2 Construction of token swapping instance

Suppose we have an instance of Star STS where the star has center 0 and leaves $1, \ldots, m$ and each vertex has a token of its same label. We have a permutation π of the tokens with $\pi(0) = 0$ and a sequence $s_1, \ldots, s_n$ with $s_j \in \{1, \ldots, m\}$ that specifies the allowed swaps. As in the weighted case in Section 2, we will refer to the tokens of the Star STS instance as ***items*** and the leaves as ***slots***. We assume without loss of generality that every slot appears in the sequence (otherwise remove that slot from the problem) and that no slot appears twice in a row in the sequence.

Construct a tree as in the weighted case except that each individual x_j and y_j is replaced by a sequence of k vertices (and tokens) with $k = (mn)^c$ for a large constant c, to be set later. Each such sequence of length k is called a ***big segment***. Refer to Figure 2 where the tree is drawn with the root in the middle, the ordering gadget to the left, and the slot gadgets to the right. We will refer to left and right as in the figure. The target ordering of tokens within a big segment behaves as though the segment just slides along the shortest path to its target, i.e., the left to right order of tokens in a segment is the same in the initial and the final configurations.

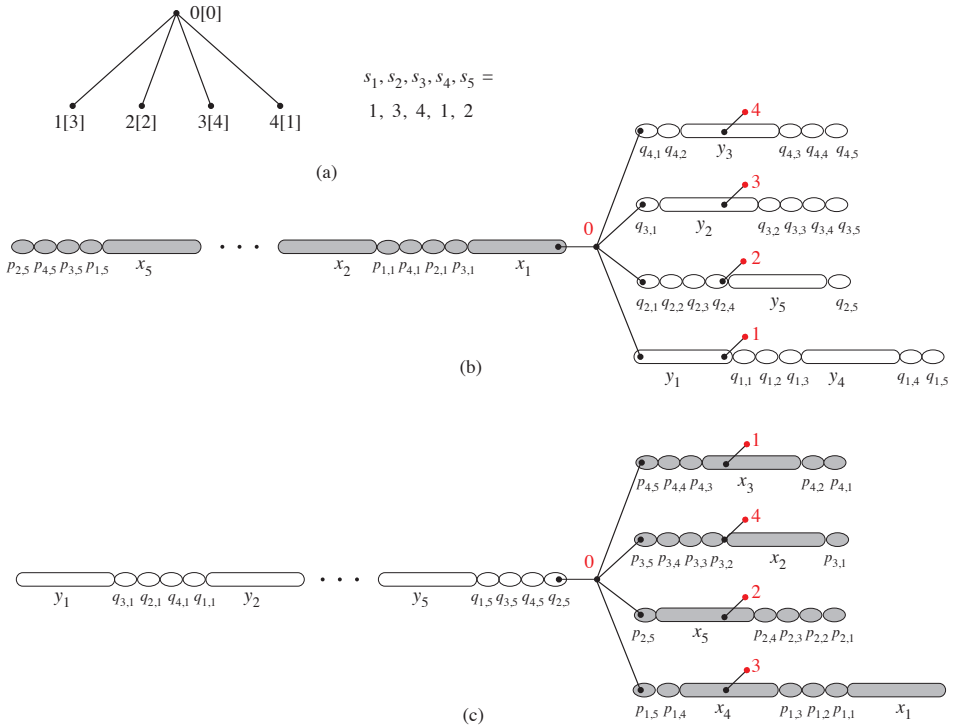

Figure 2 (a) An instance of Star STS with $m = 4$ slots and a sequence of length $n = 5$. (b) The corresponding instance of token swapping with the initial token positions. The root has item token 0 (coloured red). The ordering gadget lies to the left of the root. There are 4 slot gadgets to the right of the root. A long oval indicates a big segment of length k, and a short oval indicates a padding segment of length $k' = k/n^8$. Each nook vertex (coloured red) is attached to the k^{th} vertex from the root along the slot gadget path. Item tokens are coloured red; non-slot tokens are in the segment ovals coloured gray; and slot tokens are in the segment ovals coloured white. (c) The target token positions. In the first round of the "intended" solution, big segments y_1 and x_1 first change places. As y_1 moves left, item token 0 moves to nook parent 1 where it may swap with item token 1. As x_1 moves right, the item token moves back to the root. Then segment y_1 moves to the far left of the ordering gadget and x_1 moves to the far right of the first slot gadget. Next, padding segments $p_{3,1}$ and $q_{3,1}$ change places across the root and move to their target locations; then $p_{2,1}$ and $q_{2,1}$; then $p_{4,1}$ and $q_{4,1}$; and finally $p_{1,1}$ and $q_{1,1}$. Note the ordering $q_{3,1}, q_{2,1}, q_{4,1}, q_{1,1}$ of padding segments that lie to the right of y_1 in the final configuration – $q_{1,1}$ is last because $s_1 = 1$ and $q_{3,1}$ is first because $s_2 = 3$.

The nook vertex in each slot gadget is attached to the vertex at distance k from the root in the slot gadget, and this vertex is called the ***nook parent***. The edge between the nook vertex and the nook parent is called the ***nook edge***, see Figure 2. In the "intended" solution, the big segments leave the slot gadgets and enter the ordering gadget in the order $y_1, \ldots, y_n$.

Although we will not give details, the construction so far allows "cheating" via interference between slot gadgets. To prevent this, we add a total of $2nm$ ***padding segments*** each of length $k' = k/n^8$. The intuition is in the "intended" solution, after y_j enters the ordering gadget, one padding segment from each slot gadget will enter the ordering gadget, and then y_{j+1} will do so. We now give the details of the padding segments in the initial/final configurations of the slot/ordering gadgets. In the initial configuration there are nm padding segments $q_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n$ in the slot gadgets, and nm padding segments $p_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n$ in the ordering gadget. In the final configuration, the $q_{i,j}$'s

are in the ordering gadget and the $p_{i,j}$'s are in the slot gadgets. In the initial configuration, slot gadget i, $i = 1, \ldots, m$, contains n padding segments $q_{i,j}, j = 1, \ldots, n$. They appear in order from left to right with big segments mixed among them. Specifically, if $y_{j'}$ is a big segment in slot gadget i then $y_{j'}$ appears just before $q_{i,j'}$. In the final configuration of the ordering gadget there are m padding segments after each of the n big segments. The padding segments after y_j are $q_{i,j}, i = 1, \ldots, m$. They appear in a particular left-to-right order: $q_{s_j,j}$ is last, $q_{s_{j+1},j}$ is first, and the others appear in order of index i. Within one padding segment, the left-to-right ordering of tokens is the same in the initial and final configurations.

The initial configuration of the ordering gadget is obtained from the final configuration by: reversing (from left to right) the pattern of big segments and padding segments; changing y's to x's, and changing q's to p's. Similarly, the final configuration of slot gadget i is obtained from the initial configuration of slot gadget i in the same way.

Let A be the set of non-item tokens, and for any token t, let d_t be the distance between t's initial and target positions. To complete the reduction, we will set H to be $\frac{1}{2}\sum_{t\in A}(d_t+1)$ and set the bound K to be $H + n$. The decision question is whether this token swapping instance can be solved with at most K swaps. The reduction takes polynomial time.

The remainder of the proof can be found in the full version [1].

4 Known techniques preclude approximation factors less than 2

Previous results about sequential token swapping on trees include three different polynomial time 2-approximation algorithms and some lower bounds on approximation factors. The three algorithms all have the property that if a token at a leaf is at its destination (i.e., it is a "happy leaf token") then the algorithm will not move it. Biniaz et al. [6] proved that any algorithm with this property has a (worst case) approximation factor at least $\frac{4}{3}$. They also proved via ad-hoc arguments that the approximation factor is exactly 2 for two of the known 2-approximation algorithms (the "Happy Swap Algorithm" and the "Cycle Algorithm"). For the third 2-approximation algorithm, the Vaughan-Portier algorithm, they could not prove a lower bound better than $\frac{4}{3}$.

We prove that the Vaughan-Portier algorithm has approximation factor exactly 2. We also extend the approximation lower bounds of Biniaz et al. by proving that the approximation factor is at least 2 for a larger family of algorithms. We formalize this family as follows. For token t let P_t be the path from t's initial position to its final position. A sequence of token swaps is ***ℓ-straying*** if at all intermediate points along the sequence, every token t is within distance ℓ of the last vertex of P_t that it has reached up to this point. A token swapping algorithm is ***ℓ-straying*** if it produces ℓ-straying sequences.

Both approximation lower bounds will be proved for the same family of trees that was used by Biniaz et al. [6]. For any k and any odd b we define a tree $T_{k,b}$ together with initial and final positions of tokens. The tree $T_{k,b}$ has b paths of length k attached to a central vertex c, and a set L of k leaves also attached to c. See Figure 3. The tokens at c and L are ***happy*** – they are at their final positions. The tokens in branch $i, 0 \le i \le b-1$, have their final positions in branch $i+1$, addition modulo b, with the initial and final positions equally far from the center c.

Biniaz et al. [6] proved that the optimum number of swaps for $T_{k,b}$ is at most $(b+1)(\binom{k+1}{2}+2k)$. The solution repeatedly exchanges the tokens in branch $i, 0 \le i \le b-1$, modulo b, with the tokens at L. The first exchange moves the tokens initially at L into branch 0, and the $(b+1)^{\text{st}}$ exchange moves those tokens back to L. In the full version [1] we prove that for $T_{k,b}$ the approximation factor is not better than 2 for ℓ-straying algorithms and for the Vaughan-Portier algorithm.

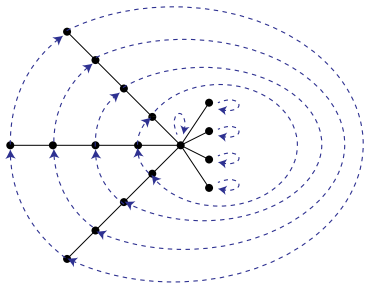

Figure 3 Tree $T_{k,b}$ with $b = 3$ branches each of length $k = 4$, and with $k = 4$ leaves attached to the center node. The dashed arrows go from a token's initial to final position. The figure is from [6].

5 Open Problems

Many interesting related problems remain open. For sequential token swapping on trees, where is the divide between NP-complete and polynomial-time? Our reduction's tree is a subdivided star (as for parallel token swapping) with the addition of one extra leaf (the nook) per path. By contrast, there is a polynomial-time algorithm for the case of a broom (a star with only one edge subdivided) [28, 18, 6]. What about a star with two subdivided edges?

For parallel token swapping, even the case of a single long path is open. Kawahara et al. [18] gave an additive approximation algorithm that uses at most one extra round. Is there an optimal algorithm, or is parallel token swapping NP-hard on paths?

There are also open problems in approximation algorithms. In parallel token swapping, we know that there is no PTAS [18]. Is there an $O(1)$-approximation for trees or general graphs? For sequential token swapping, there is a 4-approximation algorithm [23]. Is 4 a lower bound on the approximation factor of this algorithm? Is 4-approximation the best possible for general graphs?

Although this paper focused on the best reconfiguration sequence, much research is devoted to understanding the worst-case behavior for a given graph; is it NP-hard to determine the diameter of the Cayley graph, i.e., the maximum number of reconfiguration steps that can be required for any pair of token configurations? This problem is open for both sequential token swapping (implicit in [10]) and parallel token swapping [3].

References

1 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Jayson Lynch, Anna Lubiw, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. *CoRR*, abs/2103.06707, 2021. `arXiv:2103.06707`.

2 Sheldon B. Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.

3 Noga Alon, F. R. K. Chung, and R. L. Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7(3):513–530, 1994. `doi:10.1137/S0895480192236628`.

4 Amihood Amir and Benny Porat. On the hardness of optimal vertex relabeling and restricted vertex relabeling. In *Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2015.

5 Indranil Banerjee and Dana Richards. New results on routing via matchings on graphs. In R. Klasing and M. Zeitoun, editors, *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory*, volume 10472 of *Lecture Notes in Computer Science*, 2017.

6 Ahmad Biniaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. arXiv preprint, 2019. arXiv:1903.06981.

7 Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.

8 Arthur Cayley. LXXVII. Note on the theory of permutations. *Philosophical Magazine Series 3*, 34(232):527–529, 1849.

9 Arthur Cayley. Desiderata and suggestions: No. 2. the theory of groups: graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878.

10 Bhadrachalam Chitturi and Priyanshu Das. Sorting permutations with transpositions in $O(n^3)$ amortized time. *Theoretical Computer Science*, 766:30–37, 2019. doi:10.1016/j.tcs.2018.09.015.

11 Bhadrachalam Chitturi and Indulekha T S. Sorting permutations with a transposition tree. arXiv preprint, 2018. arXiv:1811.07443.

12 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. In *Proceedings of the 34th International Symposium on Computational Geometry*, pages 29:1–29:15, June 2018.

13 Ashwin Ganesan. An efficient algorithm for the diameter of Cayley graphs generated by transposition trees. *International Journal of Applied Mathematics*, 42(4), 2012. arXiv:1202.5888.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

15 Michael R. Garey, David S. Johnson, Gary L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980.

16 Laurent Gourvès, Julien Lesca, and Anaëlle Wilczynski. Object allocation via swaps along a social network. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 213–219, 2017. URL: https://www.ijcai.org/Proceedings/2017/0031.pdf.

17 Mark R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.

18 Jun Kawahara, Toshiki Saitoh, and Ryo Yoshinaka. The time complexity of permutation routing via matching, token swapping and a variant. *Journal of Graph Algorithms and Applications*, 23(1):29–70, 2019.

19 Dohan Kim. Sorting on graphs by adjacent swaps using permutation groups. *Computer Science Review*, 22:89–105, 2016.

20 Donald Ervin Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 2nd edition, 1998.

21 Benjamin Kraft. Diameters of Cayley graphs generated by transposition trees. *Discrete Applied Mathematics*, 184:178–188, 2015.

22 Wei-Tian Li, Linyuan Lu, and Yiting Yang. Routing numbers of cycles, complete bipartite graphs, and hypercubes. *SIAM Journal on Discrete Mathematics*, 24(4):1482–1494, 2010. doi:10.1137/090776317.

23 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, 2016.

24 Igor Pak. Reduced decompositions of permutations in terms of star transpositions, generalized Catalan numbers and k-ary trees. *Discrete Mathematics*, 204(1-3):329–335, 1999.

25 Frederick J. Portier and Theresa P. Vaughan. Whitney numbers of the second kind for the star poset. *European Journal of Combinatorics*, 11(3):277–288, 1990.

26 Pavel Surynek. Multi-agent path finding with generalized conflicts: An experimental study. In Jaap van den Herik, Ana Paula Rocha, and Luc Steels, editors, *Revised Selected Papers from the 11th International Conference on Agents and Artificial Intelligence*, pages 118–142, February 2019.

27 Theresa P. Vaughan. Bounds for the rank of a permutation on a tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 10:65–81, 1991.

28 Theresa P. Vaughan. Factoring a permutation on a broom. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 30:129–148, 1999.

29 Theresa P. Vaughan and Frederick J. Portier. An algorithm for the factorization of permutations on a tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 18:11–31, 1995.

30 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015.

31 Katsuhisa Yamanaka, Takashi Horiyama, J. Mark Keil, David Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. *Theoretical Computer Science*, 729:1–10, 2018.

32 Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *Inf. Process. Soc. Japan. SIG Tech. Rep*, 2015(14):1–4, 2015.

33 Louxin Zhang. Optimal bounds for matching routing on trees. *SIAM Journal on Discrete Mathematics*, 12(1):64–77, 1999. doi:10.1137/S0895480197323159.