Reconfiguration of Spanning Trees with Many or Few Leaves

Nicolas Bousquet
CNRS, LIRIS, Université de Lyon,
Université Claude Bernard Lyon 1, France
nicolas.bousquet@liris.cnrs.fr

Takehiro Ito
Graduate School of Information Sciences,
Tohoku University, Japan
takehiro@ecei.tohoku.ac.jp

Yusuke Kobayashi
Research Institute for Mathematical Sciences,
Kyoto University, Japan
yusuke@kurims.kyoto-u.ac.jp

Haruka Mizuta
Graduate School of Information Sciences,
Tohoku University, Japan
haruka.mizuta.s4@dc.tohoku.ac.jp

Paul Ouvrard
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI,
UMR5800, Talence, France
paul.ouvrard@u-bordeaux.fr

Akira Suzuki
Graduate School of Information Sciences,
Tohoku University, Japan
a.suzuki@ecei.tohoku.ac.jp

Kunihiro Wasa
Department of Computer Science and Engineering, Toyohashi University of Technology, Japan
wasa@cs.tut.ac.jp

Abstract

Let G be a graph and T_1, T_2 be two spanning trees of G. We say that T_1 can be transformed into T_2 via an edge flip if there exist two edges $e \in T_1$ and f in T_2 such that $T_2 = (T_1 \setminus e) \cup f$. Since spanning trees form a matroid, one can indeed transform a spanning tree into any other via a sequence of edge flips, as observed in [11].

We investigate the problem of determining, given two spanning trees T_1, T_2 with an additional property Π, if there exists an edge flip transformation from T_1 to T_2 keeping property Π all along.

First we show that determining if there exists a transformation from T_1 to T_2 such that all the trees of the sequence have at most k (for any fixed $k \geq 3$) leaves is PSPACE-complete.

We then prove that determining if there exists a transformation from T_1 to T_2 such that all the trees of the sequence have at least k leaves (where k is part of the input) is PSPACE-complete even restricted to split, bipartite or planar graphs. We complete this result by showing that the problem becomes polynomial for cographs, interval graphs and when $k = n - 2$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases combinatorial reconfiguration, spanning trees, PSPACE, polynomial-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.24

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.14309`.

Funding Partially supported by JSPS and MEAE-MESRI under the Japan-France Integrated Action Program (SAKURA).
Nicolas Bousquet: This work was supported by ANR project GrR (ANR-18-CE40-0032).
Takehiro Ito: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.
Yusuke Kobayashi: Supported by JSPS KAKENHI Grant Numbers JP17K19960, JP18H05291, and JP20K11692, Japan.
Haruka Mizuta: Partially supported by JSPS KAKENHI Grant Number JP19J10042, Japan.
Paul Ouvrard: This work was supported by ANR project GrR (ANR-18-CE40-0032).

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP20K11666, Japan.
Kunihiro Wasa: Partially supported by JST CREST Grant Numbers JPMJCR18K3 and JPMJCR1401, and JSPS KAKENHI Grant Number JP19K20350, Japan.

1 Introduction

Given an instance of some combinatorial search problem and two of its feasible solutions, a *reconfiguration problem* asks whether one solution can be transformed into the other in a step-by-step fashion, such that each intermediate solution is also feasible. Reconfiguration problems capture dynamic situations, where some solution is in place and we would like to move to a desired alternative solution without becoming infeasible. A systematic study of the complexity of reconfiguration problems was initiated in [11]. Recently the topic has gained a lot of attention in the context of constraint satisfaction problems and graph problems, such as the independent set problem, the matching problem, and the dominating set problem. Reconfiguration problems naturally arise for operational research problems but also are closely related to uniform sampling using Markov chains (see e.g. [5]) or enumeration of solutions of a problem. Reconfiguration problems received an important attention in the last few years. For an overview of recent results on reconfiguration problems, the reader is referred to the surveys of van den Heuvel [14] and Nishimura [13].

In this paper, our reference problem is the spanning tree problem. Let $G = (V, E)$ be a connected graph on n vertices. A *spanning tree* of G is a tree (chordless graph) with exactly $n - 1$ edges. Given a tree T, a vertex v is a *leaf* if its degree is one and is an *internal node* otherwise. A *branching node* is a vertex of degree at least three.

In order to define valid step-by-step transformations, an adjacency relation on the set of feasible solutions is needed. Depending on the problem, there may be different natural choices of adjacency relations. Let T_1 and T_2 be two spanning trees of G. We say that T_1 and T_2 differs by an *edge flip* if there exist $e_1 \in E(T_1)$ and $e_2 \in E(T_2)$ such that $T_2 = (T_1 \setminus e_1) \cup e_2$. Two trees T_1 and T_2 are adjacent if one can transform T_1 into T_2 via an edge flip. A *transformation* from T_s to T_t is a sequence of trees $\langle T_0 := T_\mathsf{s}, T_1, \ldots, T_r := T_\mathsf{t}\rangle$ such that two consecutive trees are adjacent. Ito et al. [11] remarked that any spanning tree can be transformed into any other via a sequence of edge flips. It easily follows from the exchange properties for matroid. Unfortunately, the problem becomes much harder when we add some restriction on the intermediate spanning trees. One can then ask the following question: does it still exist a transformation when we add some constraints on the spanning tree? If not, is it possible to decide efficiently if such a transformation exists? This problem was already studied for vertex modification between Steiner trees [12] for instance.

In this paper, we consider spanning trees with restrictions on the number of leaves. More precisely, what happens if we ask the number of leaves to be large (or small) all along the transformation? We formally consider the following problems:

Spanning Tree with Many Leaves
Input: A graph G, an integer k, two trees T_1 and T_2 with at least k leaves.
Output: yes if and only if there exists a transformation from T_1 to T_2 such that all the intermediate trees have at least k leaves.

In the Spanning Tree with At Most k Leaves problem, we instead want to find a transformation such that all the intermediate trees have at most k leaves (where k is a fixed constant).

Our results

We prove that both variants are PSPACE-complete. In other words, we show that Spanning Tree with Many Leaves and Spanning Tree with At Most k Leaves for every $k \geq 3$ are PSPACE-complete. This constrasts with many existing results on reconfiguration problems using edge flips which are polynomial such as matching reconfiguration [11], cycle, tree or clique reconfiguration [8]. As far as we know there does not exist any PSPACE-hardness proof for any problem via edge flip. We hope that our results will help to design more.

▶ **Theorem 1.** *Spanning Tree with Many Leaves is PSPACE-complete restricted to bipartite graphs, split graphs or planar graphs.*

These results are obtained from two different reductions. In both reductions, we need an arbitrarily large number of leaves in order to make the reduction work. In particular, one can ask the following question: is Spanning Tree with at least $n - k$ Leaves hard for some constant k (where n is the size of the instance)? We do not answer this question but we prove that, for the "dual" problem, the PSPACE-hardness is obtained even for $k = 3$.

▶ **Theorem 2.** *Spanning Tree with At Most k Leaves is PSPACE-complete for every $k \geq 3$.*

This proof is the most technically involved proof of this article and is based on a reduction from the decision problem of Vertex Cover to the decision problem of Hamiltonian Path. Let (G, k) be an instance of Vertex Cover. We first show that, on the graph H obtained when we apply this reduction, we can associate with any spanning tree T of H a vertex cover of G. The hard part of the proof consists in showing that *(i)* if T has at most three leaves, then the vertex cover associated with T has at most $k + 1$ vertices; and *(ii)* each edge flip consists of a modification of at most one vertex of the associated vertex cover.

One can note that for $k = 2$, the problem becomes the Hamiltonian Path Reconfiguration problem. We were not able to determine the complexity of this problem and we left it as an open problem.

We complete these results by providing some polynomial-time algorithms:

▶ **Theorem 3.** *Spanning Tree with Many Leaves can be decided in polynomial time on interval graphs, on cographs, or if the number of leaves is $n - 2$.*

We show that Spanning Tree with Many Leaves can be decided in polynomial time if the number of leaves is $n - 2$. As we already said, we left as an open question to determine if this result can be extended to any value $n - k$ for some fixed k. If such an algorithm exists, is it true that the problem is FPT parameterized by k?

We then show that in the case of cographs, the answer is always positive as long as the number of leaves is at most $n - 3$. Since there is a polynomial-time algorithm to decide the problem when $k = 2$ that completes the picture for cographs.

Since the problem is known to be PSPACE-complete for split graphs by Theorem 1 (and thus for chordal graphs), the interval graphs result is the best we can hope for in a sense. The interval graph result is based on a dynamic programming algorithm inspired by [2] where it is proved that the Independent Set Reconfiguration problem in the token sliding model is polynomial. Even if dynamic algorithms work quite well to decide combinatorial problems on interval (and even chordal) graphs, they are much harder to use in the reconfiguration setting. In particular, many reconfiguration problems become hard on chordal graphs (see e.g. [1, 9]) since the transformations can go back and forth.

Since the problem is hard on planar graphs, it would be interesting to determine its complexity on outerplanar graphs. We left this question as an open problem.

Related work

In the last few years, many graph reconfiguration problems have been studied through the lens of edge flips such as matchings [11, 4], paths or cycles [8]. None of these works provide any PSPACE-hardness results, only a NP-hardness result is obtained for (non Hamiltonian) path reconfiguration via edge flips in [8]. Even if the reachability problem is known to be polynomial in many cases, approximating the shortest transformation is often hard, see e.g. [4]. Flips are also often considered in computational geometry, for instance to measure the distance between two triangulations. In that setting, a flip of a triangulation is the modification of a diagonal of a C_4 for the other one. Usually, proving the existence of a transformation is straightforward and the main questions are about the length of a transformation which is not the problem addressed in this paper.

If, instead of "edge flips", we consider "vertex flips" the problems become much harder. For instance, the problem consisting in transforming an (induced) tree into another one (of the same size) is PSPACE-complete [8] (while the exchange property ensures that it is polynomial for the edge version). Mizuta et al. [12] also showed that the existence of vertex exchanges between two Steiner trees is PSPACE-complete. But transforming subsets of vertices with some properties is known to PSPACE-complete for a long time, for instance for independent sets or cliques [10].

Definitions

Given two sets S_1 and S_2, we denote by $S_1 \triangle S_2$ the *symmetric difference* of the sets S_1 and S_2, that is $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$.

For a spanning tree T, every vertex of degree one is a *leaf* and every vertex of degree at least two is an *internal node*. A vertex of degree at least three is called a *branching node*. Recall that the number of leaves of any tree T is equal to $(\sum_{v \in T}(\max\{0, d_T(v) - 2\})) + 2$. We denote by $in(T)$ the number of internal nodes of T. Note that if T contains n nodes, the number of leaves is indeed $n - in(T)$.

Let $G = (V, E)$ be a graph. A *vertex cover* C of G is a subset of vertices such that for every edge $e \in E$, C contains at least one endpoint of e. C is *minimum* if its cardinality is minimum among all vertex covers of G. Note that in particular, C is inclusion-wise minimal and thus for every vertex $u \in C$, there is an edge $e \in E$ which is covered only by u. We denote by $\tau(G)$ the size of a minimum vertex cover of G.

Let X, Y be two vertex covers of G. X and Y are *TAR-adjacent*[1] (resp. TJ-adjacent) if there exists a vertex x (resp. x and y) such that $X = Y \cup \{x\}$ or $Y = X \cup \{x\}$ (resp. $X = Y \setminus \{y\} \cup \{x\}$). We will consider the following problem:

MINIMUM TAR-VERTEX COVER RECONFIGURATION
Input: A graph G, two minimum vertex covers X, Y of size k.
Output: yes if and only if there exists a sequence from X to Y of TAR-adjacent vertex covers, all of size at most $k + 1$.

[1] TAR stands for "Token Additional Removal".

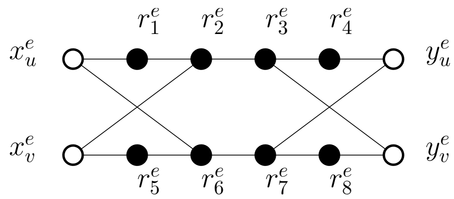


Figure 1 edge-gadget. The white vertices are the only ones connected to the outside.

Similarly, one can define the Minimum TJ-Vertex Cover Reconfiguration (MVCR for short) where we want to determine whether there exists a sequence of TJ-adjacent vertex covers from X to Y. Note that all the vertex covers must be of size $|X| = |Y| = k$.

2 Spanning trees with few leaves

▶ **Theorem 4.** *Spanning Tree with At Most three Leaves is PSPACE-complete.* [2]

In order to prove Theorem 4, we will provide a reduction from Minimum TAR-Vertex Cover Reconfiguration to Spanning Tree with At Most three Leaves.

▶ **Theorem 5** (Wrochna [15]). TAR-*Vertex Cover Reconfiguration is PSPACE complete even for bounded bandwidth graphs.*

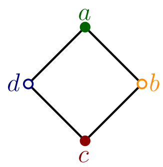
The idea of the proof of Theorem 4 consists in adapting a reduction from Vertex cover to Hamiltonian Path (for the optimization version). Let $(G = (V, E), k)$ be an instance of Vertex Cover. This reduction creates a graph $H(G)$ which contains a Hamiltonian path if and only if G admits a vertex cover of size k. The reduction is given in Section 2.1 together with some properties of the spanning trees with at most three leaves in $H(G)$. In order to adapt the proof in the reconfiguration setting, we need to prove that the proof is "robust" with respect to several meanings of the word. First, we need to show that, if we consider a spanning tree with at most three leaves in $H(G)$ then there is a "canonical" vertex cover of size at most $k + 1$ associated with it (it is the most technical part of the proof). Then, for any edge flip between two spanning trees with at most three leaves, we need to show that the corresponding vertex covers associated with them are TAR-adjacent . We will indeed also need to prove the reverse direction.

2.1 The Reduction

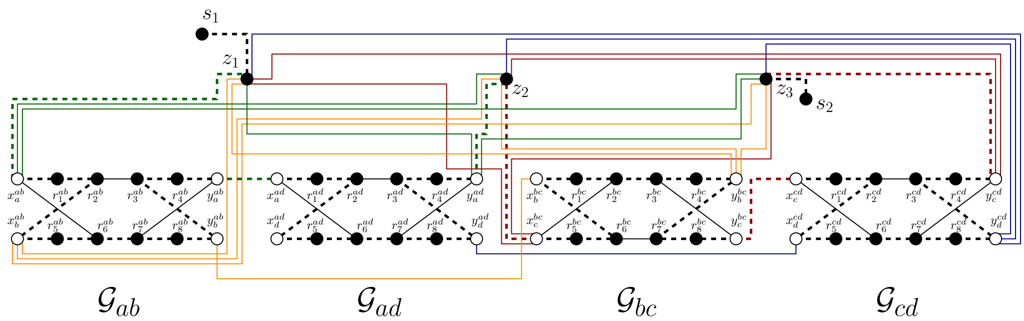
The reduction is a classical reduction (see Theorem 3.4 of [6] for a reference) from the optimization version of Vertex Cover to the optimization version of Hamiltonian Path. Let G be a graph and k be an integer. Let us construct a graph $H(G)$ (abbreviated into H when no confusion is possible) as follows:

Construction of $H(G)$. For each edge $e = uv$ of G, we create the following *edge-gadget* $\mathcal{G}_e$ represented in Figure 1. The edge-gadget $\mathcal{G}_e$ has four *special vertices* denoted by $x_u^e, x_v^e, y_u^e, y_v^e$. The vertices x_u^e and x_v^e are called the *entering vertices* and y_u^e and y_v^e the *exit vertices*. The gadget contains eight additional vertices denoted by $r_1^e, \dots, r_8^e$. When e is clear from context, we will omit the superscript. The graph induced by these twelve vertices is represented in Figure 1. The vertices $r_1^e, \dots, r_8^e$ are *local vertices* and their neighborhood will be included in the gadget. The only vertices connected to the rest of the graphs are the special vertices.

[2] Note that the reduction can be easily adapted to more leaves.



(a) Original instance (G, k) of Minimum Vertex Cover with a vertex cover $\{a, b\}$.



(b) Graph $H(G)$ obtained from the reduction. The ordering for the vertices of the vertex cover $\{a, b\}$ of G is the lexicographic ordering, as well as the ordering of the edges incident to each vertex. The corresponding Hamiltonian path is depicted by the thick dashed edges.

Figure 2 Illustration of the reduction of Theorem 4.

We add an independent set $Z := \{z_1, \ldots, z_{k+1}\}$ of $k + 1$ new vertices to $V(H)$. And we finally add to $V(H)$ two more vertices s_1, s_2 in such a way that z_1 (resp. z_{k+1}) is the only neighbor of s_1 (resp. s_2) in $H(G)$. Since s_1 and s_2 have degree one in $H(G)$, s_1 and s_2 are leaves in any spanning tree of $H(G)$. In particular, the two endpoints of any Hamiltonian path of $H(G)$ are necessarily s_1 and s_2.

Let us now complete the description of $H(G)$ by explaining how the special vertices are connected to the other vertices of $H(G)$. Let $u \in V(G)$. Let $E' = e_1, \ldots, e_\ell$ be the set of edges incident to u in an arbitrary order. We connect $x_u^{e_1}$ and $y_u^{e_\ell}$ to all the vertices of Z. For every $1 \leq i \leq \ell - 1$, we connect $y_u^{e_i}$ to $x_u^{e_{i+1}}$. The edges $y_u^{e_i} x_u^{e_{i+1}}$ are called the *special edges of* u. The *special edges* of $H(G)$ are the union of the special edges for every $u \in V(G)$ plus the edges incident to Z but $s_1 z_1$ and $s_2 z_{k+1}$. This completes the construction of $H(G)$ (see Figure 2 for an example).

▶ Remark 6. If T is a spanning tree of $H(G)$ with at most ℓ leaves, then at most $\ell - 2$ of them are in $V(H) \setminus \{s_1, s_2\}$.

Let T be a spanning tree of $H(G)$. An edge-gadget is *irregular* if at least one of its twelve vertices is not of degree two in T. An edge-gadget is *regular* if it is not irregular. By abuse of notation we say that $e \in E(G)$ is regular (resp. irregular) if the edge-gadget of e is regular (resp. irregular). A vertex u is *regular* if every edge incident to u is regular. The vertex u is *irregular* otherwise.

Let S be a subset of vertices of $H(G)$. We denote by $\delta_T(S)$ the set of edges with exactly one endpoint in S. When there is no ambiguity, we omit the subscript T. Moreover, if S is the singleton $\{u\}$, we write $\delta_T(u)$ for $\delta_T(\{u\})$. The restriction $T(\mathcal{G}_e)$ of a spanning tree T *around an edge-gadget* $\mathcal{G}_e$ is the set of edges with both endpoints in $\mathcal{G}_e$ plus the edges of $\delta_T(\mathcal{G}_e)$ (which are considered as "semi edge" with one endpoint in $\mathcal{G}_e$).

▶ **Lemma 7.** *Let T be a spanning tree of H and $\mathcal{G}$ be a regular edge-gadget. Then the tree T around the edge-gadget $\mathcal{G}$ is one of the two graphs represented in Figure 3. Note that the graph of Figure 3(b) has to be considered up to symmetry between u and v.*

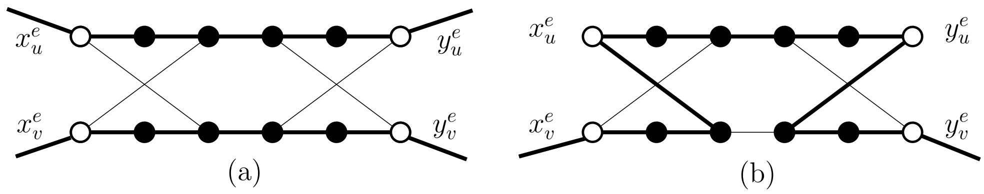


Figure 3 The two possible sub-graphs around a regular edge-gadget $\mathcal{G}$. Bold edges are edges in the tree. Edges with one endpoint in the gadget are edges of $\delta(\mathcal{G})$.

▶ **Lemma 8 (*).** *Let G be a graph, T be a spanning tree of $H(G)$, and u be a regular vertex of T. If there exists an edge $e \in E(G)$ with endpoint u such that x_u^e or y_u^e has degree one in the subgraph of T induced by the vertices of $H[\mathcal{G}_e]$, then, for every edge e' with endpoint u, $x_u^{e'}$ and $y_u^{e'}$ have degree one in the subgraph of T induced by the vertices of $H[\mathcal{G}_{e'}]$.*
In particular, there is an edge of T between Z and the first entering vertex of u and an edge between Z and the last exit vertex of u.

If, for a regular vertex u and an edge $e = uv$, x_u^e or y_u^e have degree one in $H[\mathcal{G}_e]$, then there is a path between two vertices of Z passing through all the special vertices $x_u^{e'}$ and $y_u^{e'}$ for every e' incident to u and all the vertices on this path have degree two. Note that the union of all such vertices forms a vertex cover of G.

2.2 Reconfiguration hardness

Let T be a spanning tree with at most three leaves. By Lemma 7, for every edge-gadget $\mathcal{G}_e$, if $T(\mathcal{G}_e)$ is not one of the two graphs of Figure 3, $\mathcal{G}_e$ contains a branching node or a leaf. So Remark 6 implies:

▶ Remark 9. There are at most two irregular edge-gadgets. Thus there are at most four irregular vertices.

Indeed, if T has two leaves, all the edge-gadgets are regular. If T has three leaves, the third leaf must be in an edge-gadget, creating an irregular edge-gadget. And this leaf might create a new branching node which might be in another edge-gadget than the one of the third leaf. So the number of irregular edge-gadget is at most two, and thus the number of irregular vertices is at most four (if the edges corresponding to these two edge-gadgets have pairwise distinct endpoints).

Let T be a spanning tree of $H(G)$ with at most three leaves. A vertex v is *good* if there exists an edge $e = vw$ for $w \in V(G)$ such that x_v^e or y_v^e has degree one in the subtree of T induced by the twelve vertices of the edge-gadget of e. In other words, if we simply look at the edges of T with both endpoints in $\mathcal{G}_e$, x_v^e or y_v^e has degree one (or said again differently, x_v^e or y_v^e are adjacent to exactly one local vertex). Let us denote by $S(T)$ the set of good vertices. Using the fact that every gadget contains at most one vertex of degree three and one vertex of degree one by Remark 6, we can show:

▶ **Lemma 10 (*).** *Let T be a spanning tree with at most three leaves of $H(G)$ and $e = uv$ be an edge of G. At least one special vertex of the edge-gadget $\mathcal{G}_e$ has degree one in the subgraph of T induced by the vertices of $\mathcal{G}_e$. In particular, $S(T)$ is a vertex cover.*

So, for every tree T with at most three leaves, $S(T)$ is a vertex cover. We say that $S(T)$ is *the vertex cover associated with T.*

The next two technical lemmas ensure that an edge flip transformation provides a TAR-vertex cover reconfiguration sequence.

▶ **Lemma 11** (*). *Every spanning tree T of $H(G)$ with at most three leaves satisfies $|S(T)| \leq k+1$.*

Sketch of the proof. Assume by contradiction that $|S| \geq k+2$. By Remark 9, at least $k-2$ vertices of S are regular. By Lemma 8, for each regular vertex $w \in S$, there is an edge of T between Z and the first entering vertex of w and Z and the last exit vertex of w. So at least $2k-4$ edges of $\delta_T(Z)$ are incident to regular vertices. Moreover two edges of $\delta_T(Z)$ are incident to s_1 and s_2. So, T already has $2k-2$ edges in $\delta_T(Z)$. Since $|Z| = k+1$ and T has at most three leaves, Remark 6 ensures that $\delta_T(Z)$ has size $2k+1, 2k+2$ or $2k+3$. The main part of the proof, not included in this extended abstract, consists in proving that the edges between Z and entering or exit vertices of irregular vertices is too large. ◀

So the vertex cover $S(T)$ associated with every spanning tree T with at most three leaves has size at most $k+1$. In order to prove that a spanning tree transformation provides a vertex cover transformation for the TAR setting, we have to prove that, for every edge flip, then either S is not modified, or one vertex is added to S or one vertex is removed from S.

▶ **Lemma 12** (*). *Let T_1 and T_2 be two adjacent trees with at most three leaves. Then the symmetric difference between the sets S associated with the two trees is at most one.*

Lemmas 11 and 12 immediately implies the following:

▶ **Lemma 13.** *If there is an edge flip reconfiguration sequence between two spanning trees T_1 and T_2, then there is a TAR-reconfiguration sequence (with threshold $k+1$) between $S(T_1)$ and $S(T_2)$.*

We refer the reader to the complete version for a proof of the converse direction.

3 Spanning tree with many leaves

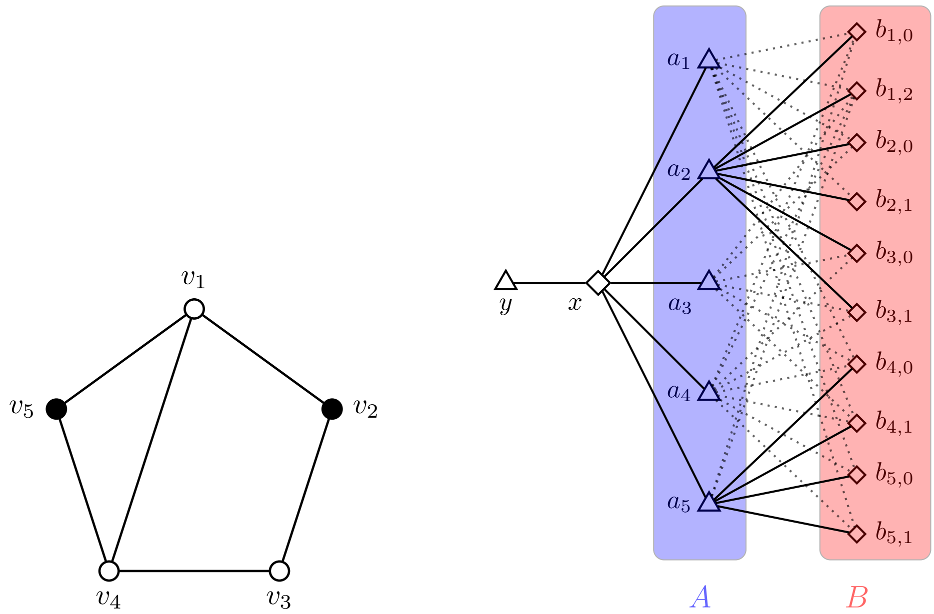
Before stating the main results of this section, let us prove the following:

▶ **Lemma 14** (*). *Let G be a graph and T_1, T_2 be two trees. There exists a transformation from T_1 to T_2 such that every intermediate tree T satisfies $in(T) \subseteq in(T_1) \cup in(T_2)$.*
In particular, all the trees with the same set of internal nodes are in the same connected component of the reconfiguration graph.

3.1 Hardness results

▶ **Theorem 15.** *Spanning Tree with Many Leaves is PSPACE-complete even restricted to bipartite graphs or split graphs.*

Sketch of the proof. We first briefly explain the proof for bipartite graphs. We provide a polynomial-time reduction from the TAR-Dominating Set Reconfiguration problem (abbreviated in TAR-DSR problem). Haddadan et al [7]. showed that the TAR reconfiguration of dominating sets is PSPACE-complete. More precisely, they proved that given a graph G and D_s, D_t two dominating sets of G, deciding whether there is a reconfiguration sequence between D_s and D_t under the TAR$(\max(|D_s|, |D_t|)+1)$ rule is PSPACE-complete.



(a) Original graph G. **(b)** Corresponding bipartite graph G'.

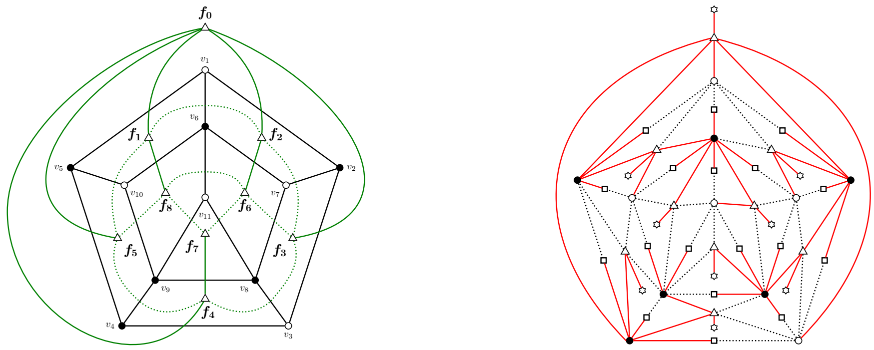
Figure 4 Example for the reduction of Theorem 15: the dominating set $D = \{v_2, v_5\}$ of G is depicted by the black vertices and the spanning tree of G' associated with D is the tree induced by the solid edges. For the split case, we add all the possible edges in $G'[A]$ so that $G'[A \cup \{x\}]$ is a clique and $G'[B \cup \{y\}]$ an independent set.

Let $G = (V, E)$ be a graph with vertex set $V(G) = \{v_1, v_2, \dots, v_n\}$ and let D_{s}, D_{t} be two dominating sets of G. Free to add vertices to the set of smallest size, we can assume without loss of generality that D_{s} and D_{t} are both of size k. Let $(G, k+1, D_{\mathsf{s}}, D_{\mathsf{t}})$ be the corresponding instance of Dominating Set Reconfiguration under TAR, where $k+1$ is the threshold that we cannot exceed. We construct the bipartite graph G' as follows: we make a first copy $A = \{a_1, a_2, \dots, a_n\}$ of the vertex set of G, and a second copy $B = \{b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}, \dots, b_{n,0}, b_{n,1}\}$ where we double each vertex. We add an edge between $a_i \in A$ and $b_{j,k} \in B$ for $k \in \{0, 1\}$ if and only if $v_j \in N_G[v_i]$. Note that $N(b_{i,0}) = N(b_{i,1})$, for every $1 \leq i \leq n$. We finally add a vertex x adjacent to all the vertices in A and we attach it to a degree-one vertex y. Note that G' is bipartite since $A \cup \{y\}$ and $B \cup \{x\}$ induce two independent sets (see Figure 4 for an illustration).

▷ Claim 16 (*). For every spanning tree T of G', $in(T) \cap A$ is a dominating set of G.

▷ Claim 17 (*). For every spanning tree T of G', there exists a tree T_A in the same connected component of T in the reconfiguration graph such that $in(T_A) \subseteq in(T) \cap (A \cup \{x\})$.

Let D be a dominating set of G of size k. We can associate with D a spanning tree of G' with $k+1$ internal nodes as follows. We attach every vertex in $A \cup \{y\}$ to x. Every vertex $b_i \in B$ is a leaf adjacent to a vertex that dominates v_i in D. If v_i has more than one neighbor in D, we choose the one with the smallest index. This spanning tree is called the *spanning tree associated with* D. Due to space restrictions, the proof that $(G, k+1, D_{\mathsf{s}}, D_{\mathsf{t}})$ is yes-instance of TAR-DSR if and only if $(G', k', T_{\mathsf{s}}, T_{\mathsf{t}})$ is a yes-instance of Spanning Tree with Many Leaves is not included in this extended abstract.



(a) Original labeled planar graph G. **(b)** Corresponding planar graph G'.

Figure 5 Reduction for Theorem 18. The vertex cover C of G is depicted by the black vertices. The dual graph is the graph induced by the green edges. The spanning tree obtained from the BFS is represented by the solid edges. The face-vertices (respectively edge-vertices) of G' are depicted by triangles (resp. squares). The spanning tree T of G' associated with the vertex cover C is the tree induced by the red edges. The number of leaves of T is $2(|E(G)|+1)-|C| = 32$.

Let us now quickly explain how to adapt this proof for split graphs. We first add an edge between any two vertices in A so that $G'[A]$ is a clique. Then, observe that $G'[A \cup \{x\}]$ is a clique, and $G'[B \cup \{y\}]$ an independent set. The proof that the resulting instance is a yes-instance of Spanning Tree with Many Leaves if and only if $(G, k+1, D_{\mathsf{s}}, D_{\mathsf{t}})$ is a yes-instance of TAR-DSR is similar to the one for bipartite graphs (see the full version). ◀

▶ **Theorem 18.** *Spanning Tree with Many Leaves is PSPACE-complete even restricted to planar graphs.*

The reduction. First, observe that MVCR is PSPACE-complete, even if the input graph is planar [10][3]. We use a reduction from MVCR, which is a slight adaptation of the reduction used in [12, Theorem 4]. Let $G = (V, E)$ be a planar graph and let $(G, C_{\mathsf{s}}, C_{\mathsf{t}})$ be an instance of MVCR. We can assume that G is given with a planar embedding of G since such an embedding can be found in polynomial time. Let $F(G)$ be the set of faces of G (including the outer face). We construct the corresponding instance $(G', k, T_{\mathsf{s}}, T_{\mathsf{t}})$ as follows:

We define G' from G as follows. We start from G and first subdivide every edge $uv \in E(G)$ by adding a new vertex w_{uv}. Then, for every face $f \in F(G)$, we add a new vertex w_f adjacent to all the vertices of the face f. Finally, we attach a leaf u_f to every vertex w_f. Note that G' is a planar graph and $|V(G')| = |V(G)| + |E(G)| + 2 \cdot |F(G)|$. The vertices w_{uv} for $uv \in E$ (resp. w_f for $f \in F$) are *edge-vertices* (resp. *face-vertices*). The vertices u_f for every f are called the *leaf-vertices*. Note that, for every spanning tree T, all the face-vertices are internal nodes of T and all the leaf-vertices are leaves of T. The vertices of $V(G')$ which are neither edge, face of leaf vertices are called *original vertices*. Finally, we choose an arbitrarily ordering of $V(G)$ and F. It will permit us to define later a canonical spanning tree for every vertex cover (see Figure 5 for an example).

▶ **Lemma 19 (*).** *Every spanning tree of G' has at most $2(|E(G)|+1) - \tau(G)$ leaves.*

[3] Actually, Hearn and Demaine [10] showed the PSPACE-completeness for the reconfiguration of maximum independent sets. Since the complement of a maximum independent set is a minimum vertex cover, we directly get the PSPACE-completeness of MVCR.

▶ **Lemma 20 (*).** *For any minimum vertex cover C of $G = (V, E)$, we can define a canonical tree with exactly $k := 2(|E(G)| + 1) - \tau(G)$ leaves which are all the edge-vertices, all the leaf-vertices and all the original vertices but the ones in C. Moreover, this spanning can be computed in polynomial time.*

Recall that $(G, C_{\mathsf{s}}, C_{\mathsf{t}})$ is an instance of Minimum Vertex Cover Reconfiguration. By Lemma 20, we can compute in polynomial time two spanning trees T_{s} and T_{t} from C_{s} and C_{t} with $2(|E(G)| + 1) - \tau(G)$ leaves. Finally, we set $k := 2(|E(G)| + 1) - \tau(G))$. Let $(G', k, T_{\mathsf{s}}, T_{\mathsf{t}})$ be the resulting instance of Spanning Tree with Many Leaves. It remains to prove that $(G, C_{\mathsf{s}}, C_{\mathsf{t}})$ is a yes-instance if and only $(G', k, T_{\mathsf{s}}, T_{\mathsf{t}})$ is a yes-instance. Suppose first that we have a reconfiguration sequence S between C_{s} and C_{t}. By Lemma 20, we can associate with each vertex cover C_i of S a spanning tree T_i of G'. To show that there is a reconfiguration sequence S' between T_{s} and T_{t}, we show that we can transform two consecutive spanning trees of S' without increasing the number of internal nodes. Note that we use the fact that each C_i of S is a minimum vertex cover. For the converse direction, we show that all the edge-vertices of any spanning tree in a reconfiguration sequence S from T_{s} to T_{t} is a leaf. Hence, one can directly deduce a vertex cover C_i of G from a spanning tree $T_i \in S$. Finally, we show that *(i)* each vertex cover is of size $\tau(G)$ and; *(ii)* $|C_i \triangle C_{i+1}| \in \{0, 2\}$ for any two consecutive vertex covers.

3.2 Two internal nodes and cographs

Recall that, for every tree, the number of leaves is equal to n minus the number of internal nodes. So, for convenience, our goal would consist in minimizing the number of internal nodes rather than maximizing the number of leaves.

▶ **Theorem 21.** *Let G be a graph and T_{s} or T_{t} be two spanning trees with at most two internal nodes. Then we can check in polynomial time if one can transform the other via a sequence of spanning trees with at most two internal nodes.*

Sketch of the proof. If T_{s} or T_{t} has one internal node, the problem can be easily decided. So we restrict to the case $|in(T_{\mathsf{s}})| = |in(T_{\mathsf{t}})| = 2$. Moreover, if $in(T_{\mathsf{s}}) = in(T_{\mathsf{t}})$, then $(G, k, T_{\mathsf{s}}, T_{\mathsf{t}})$ is a yes-instance. So we only consider the case $in(T_{\mathsf{s}}) \neq in(T_{\mathsf{t}})$.

A vertex u is a *pivot* vertex of G if $\deg u \geq n - 2$ in G ($\deg u$ being the size of the neighborhood of u, u not included). A spanning tree T of G is *frozen* if all the spanning trees in its connected component of the reconfiguration graph have the same internal nodes.

▷ Claim 22 (*). Let T be a spanning tree of G. If $in(T)$ does not contain a pivot vertex, then T is frozen.

▷ Claim 23 (*). Let u be a pivot vertex. All the trees containing u as internal vertex are in the same connected component of the reconfiguration graph.

Using these two claims, we can prove that the result follows. ◀

One can naturally wonder if this can be extended to larger values of k or if it is special for $k = 2$. We left this as an open problem. We were only interested in the case $k = 2$ since it was of particular interest for cographs. Indeed, if $k \geq 3$, one can prove that the answer is always positive for cographs. Together with Theorem 21, it implies:

▶ **Theorem 24 (*).** *Spanning Tree with Many Leaves can be decided in polynomial time on cographs.*

3.3 Interval graphs

A graph G is an *interval graph* if G can be represented as an intersection of segments on the line. More formally, each vertex can be represented with a pair (a, b) (where $a \leq b$) and vertices $u = (a, b)$ and $v = (c, d)$ are adjacent if the intervals (a, b) and (c, d) intersect. Let $u = (a, b)$ be a vertex; a is the *left extremity* of u and b the *right extremity* of u. Given an interval graph, a representation of this graph as the intersection of intervals in the plane can be found in $\mathcal{O}(|V| + |E|)$ time (see e.g. [3]). In the rest of the section we assume that a representation is given.

▶ **Theorem 25.** *Spanning Tree with Many Leaves can be decided in polynomial time on interval graphs.*

The proof techniques are inspired from [2]. The rest of this section is devoted to prove Theorem 25. Moreover, if G is a clique, then G is a cograph and then the problem can be decided in polynomial by Theorem 24. So, from now on, we can assume that G is not a clique and in particular $in(G) \geq 2$.

C-minimum spanning trees. Let k be an integer, G be a graph. We denote by $\mathcal{R}(G, k)$ the edge flip reconfiguration graph of the spanning trees of G with at most k internal nodes.

Let T, T' be two spanning trees with the same set of internal nodes. Lemma 14 ensures that T and T' are in the same connected component of $\mathcal{R}(G, k)$. So in what follows, we will often associate a tree T with its set $in(T)$ of internal nodes.

For every interval graph, we can define a spanning tree T_C called the *canonical tree* which minimizes the number of internal vertices and such that for every i, the right extremity of the i-th internal node is maximized.

A tree T is *C-minimum* if no tree T' in the connected component of T in $\mathcal{R}(G, k)$ contains fewer internal nodes than T. The goal of this part consists in showing that all the trees that are not C-minimum are in the connected component of T_C in $\mathcal{R}(G, k)$. The following lemmas follow from basic transformation on spanning trees:

▶ **Lemma 26 (*).** *Let T be a spanning tree of G and $k \geq in(T)$. If there exist two internal nodes u, v of T such that the interval of u is included in the interval of v then T is not C-minimum in $\mathcal{R}(G, k)$. Moreover a tree with internal nodes included in $in(T) \setminus \{u\}$ in the component of T can be found in polynomial time, if it exists.*

▶ **Lemma 27 (*).** *Let T be a spanning tree of G. If there exist three pairwise adjacent internal nodes u, v, w such that $N[u] \subseteq N[v] \cup N[w]$ then T is not C-minimum. Moreover a tree with internal nodes included in $in(T) \setminus \{u\}$ in the connected component of T can be found in polynomial time.*

Note that if u, v, w induce a triangle, then Lemma 26 or 27 holds. So, free to perform some pre-processing operations, we can assume that the set of internal nodes of a spanning T of G induces a path. Indeed, if an internal node x is incident to three other internal nodes u, v, w, then either at least two of them contain the left extremity (or right extremity) of x, or one interval is strictly included in the interval of x. In the first case there is a triangle and we can apply Lemma 26 or 27. In the second case, we can apply Lemma 26.

▶ **Lemma 28 (*).** *Let G be an interval graph and k be an integer. Any spanning tree T of G satisfying $in(T) < k$ is in the connected component of T_C in $\mathcal{R}(G, k)$.*

Sketch of the proof. The proof consists in showing that we can iteratively increase the number of internal nodes on which T and T_C agree without increasing the number of internal nodes at the end of the sequence (and increase it by at most one during the sequence). ◀

Full access. Let T be a tree such that $in(T)$ induces a path. Recall that the left and right extremities orderings agree. The *leftmost vertex* of T is the vertex of $in(T)$ that is minimal for both l and r. The *i-th internal node* of T is the internal node with the i-th smallest left extremity.

Let G be an interval graph and $v \in V(G)$. The *auxiliary graph* H_v of G on v is defined as follows. The vertex set of H_v is v plus the set W of vertices w which end after v and start after the beginning of v (i.e. vertices whose interval ends after v but does not contain v) plus a new vertex x, called the *artificial vertex*. The set of edges of H_v is the set of edges induced by $G[W \cup \{v\}]$ plus the edge xv.

▷ Claim 29 (*). Let G be an interval graph and v be a vertex of G. The graph H_v is an interval graph.

Let $v \in V(G)$. Every spanning tree of H_v necessarily contains v in its set of internal nodes. Indeed, by construction, the graph H_v contains a vertex x of degree one which is only incident to v. Moreover, v is the leftmost internal node of any spanning tree T of H_v.

Let G be an interval graph, $k \in \mathbb{N}$ and T be a spanning tree with internal nodes I such that $|I| = k$. Let $v \in V(G)$. The *restriction* of a spanning tree T to H_v is any spanning tree of H_v with internal nodes included in $(in(T) \cup \{v\}) \cap V(H_v)$. We denote by k'_v (or k' when no confusion is possible) the value $|(in(T) \cup \{v\}) \cap V(H_v)|$. Let T' be the restriction of T to H_v as defined above. One can easily check that the number of internal nodes of T' is at most k'.

The vertex v is *good* if the restriction of T to H_v is not C-minimum in $\mathcal{R}(H_v, k')$. The vertex v is *normal* otherwise. Let v be a normal vertex. Recall that v is the leftmost internal node of any spanning tree of H_v. Let C be the connected component of the restriction of T to H_v in $\mathcal{R}(H_v, k')$. We denote by $\ell'_v(T)$ the second internal node of a spanning tree of H_v in C that minimizes its left extremity. Similarly we denote by $r'_v(T)$ the second internal node of a spanning tree of H_v in C that maximizes its right extremity. When they do not exist[4], we set $\ell'_v(T) = -\infty$ and $r'_v(T) = +\infty$.

We say that we have *full access to* T if, for every vertex $v \in V(G)$, we have a constant time oracle saying if v is good or normal. And if v is normal, we moreover have a constant time access to $\ell'_v(T)$ and $r'_v(T)$. What remains to be proved is that *(i)* knowing this information for two spanning trees T and T' is enough to determine if they are in the same connected component of $\mathcal{R}(G, k)$, and that *(ii)* this information can be computed in polynomial time.

Dynamic programming algorithm. Let us first state the following useful lemma.

▶ **Lemma 30 (*).** *Let G be an interval graph and $k \in \mathbb{N}$. Let T be a spanning tree of G and v be an internal node of T. Let $J := in(T) \cap V(H_v)$ and $k' = |J|$. If a tree T' with internal nodes J can be transformed into a tree with internal nodes K in $\mathcal{R}(H_v, k')$ then T can be transformed into a tree with internal nodes $(in(T) \setminus J) \cup K$ in $\mathcal{R}(G, k)$.*

In particular, if T' is not C-minimum in $\mathcal{R}(H_v, k')$ then T is not C-minimum in $\mathcal{R}(G, k)$.

Let us now prove that if we have full access to H_v for any v we can determine if T is C-minimum and, if it is, the rightmost possible right extremity of the first internal node of the trees in the connected component of T in $\mathcal{R}(G, k)$.

[4] It is the case if and only if H_v is a clique.

▶ **Lemma 31 (*).** *Let G be an interval graph, $k \in \mathbb{N}$, and T be a spanning tree of G with at most k internal nodes. Assuming full access to T:*

- *We can decide in polynomial time if T is C-minimum in $\mathcal{R}(G, k)$ and,*
- *If T is C-minimum, we can compute in polynomial time the rightmost possible right extremity of the first internal node of a tree in the connected component of T in $\mathcal{R}(G, k)$.*

Sketch of the proof. Let v be the first internal node of T. Since we have full access to T, we can compute $w := \ell'_v(T)$. Lemma 30 ensures that there exists a spanning tree in the component of T in $\mathcal{R}(G, k)$ with second internal node w. We now determine how far we can move to the right the vertex v knowing this vertex. ◀

We say that we have *full access to T after v* if for every vertex $w \in V(G)$ with $w > v$, we have access in constant time to a table that permits us to know whether w is good or normal. And if w is normal, we also have access to $\ell'_w(T)$ and $r'_w(T)$. Using a proof similar to the one of Lemma 31, one can prove the following:

▶ **Lemma 32 (*).** *Let G be an interval graph, $k \in \mathbb{N}$, $v \in V(G)$ and T be a spanning tree of G with at most k internal nodes.*

- *We can decide in polynomial time if v is good if we have full access to T after v.*
- *If T is C-minimum, we can moreover compute $r'_v(T)$ and $\ell'_v(T)$ in polynomial time.*

Lemmas 32 ensures that we can, using backward induction on the ordering of the vertices, decide in polynomial time for all the vertices v of the graph if a vertex is good and if not we can compute $r'_v(T)$ and $\ell'_v(T)$. So we have full access to T in polynomial time.

▶ **Lemma 33 (*).** *Let G be an interval graph and v be a vertex of G. Let T_1, T_2 be two spanning trees of G with internal nodes I_1 and I_2 of H_v such that v is normal for both T_1 and T_2. Let $i_1 := r'_v(I_1)$ and $i_2 := r'_v(I_2)$. The trees T_1 and T_2 are in the same connected component of H_v if and only if:*

- *$i_1 = i_2$ and,*
- *Any spanning trees with internal nodes $(I_1 \setminus \{v\}) \cup \{i_1\}$ and $(I_2 \setminus \{v\}) \cup \{i_2\}$ are in the same connected component of $\mathcal{R}(H_{i_1}, k)$.*

We now have all the ingredients to prove Theorem 25.

Proof of Theorem 25. We can determine in polynomial time if the spanning trees are C-minimum by Lemma 31. If both of them are not, then both of them can be reconfigured to T_C and there exists a transformation from T_1 to T_2 by Lemma 31. If only one of them is, say T_1, we can replace T_1 by T_C (since they are in the same connected component in the reconfiguration graph). So we can assume that T_1 and T_2 are C-minimum. And the conclusion follows by Lemma 33. ◀

References

1 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 13:1–13:17, 2019. doi:10.4230/LIPIcs.STACS.2019.13.

2 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, pages 127–139, 2017. doi:10.1007/978-3-319-68705-6_10.

3 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

4 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest reconfiguration of matchings. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019*, pages 162–174, 2019. doi:10.1007/978-3-030-30786-8_13.

5 Alan Frieze and Eric Vigoda. A survey on the use of markov chains to randomly sample colourings. *Oxford Lecture Series in Mathematics and its Applications*, 34:53, 2007.

6 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

7 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theor. Comput. Sci.*, 651(C):37–49, October 2016. doi:10.1016/j.tcs.2016.08.016.

8 Tesshu Hanaka, Takehiro Ito, Haruka Mizuta, Benjamin Moore, Naomi Nishimura, Vijay Subramanya, Akira Suzuki, and Krishna Vaidyanathan. Reconfiguring spanning and induced subgraphs. *Theor. Comput. Sci.*, 806:553–566, 2020. doi:10.1016/j.tcs.2019.09.018.

9 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part I*, pages 152–162, 2017. doi:10.1007/978-3-319-71150-8_15.

10 Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, October 2005. doi:10.1016/j.tcs.2005.05.008.

11 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

12 Haruka Mizuta, Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Reconfiguration of minimum steiner trees via vertex exchanges. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany.*, pages 79:1–79:11, 2019. doi:10.4230/LIPIcs.MFCS.2019.79.

13 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.

14 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

15 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.