

Token Sliding Reconfiguration on DAGs

Jona Dirks 

Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, Clermont-Ferrand, France

Alexandre Vigny 

Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, Clermont-Ferrand, France

Abstract

Given a graph G and two independent sets of same size, the *Independent Set Reconfiguration Problem under token sliding* asks whether one can, in a step by step manner, transform the first independent set into the second one. In each step we must preserve the condition of independence. Further, referring to solution vertices as tokens, we are only permitted to slide a token along an edge. Until the recent work of Ito et al. [Ito et al. MFCS 2022] this problem was only considered on undirected graphs. In this work, we study reconfiguration under token sliding focusing on DAGs.

We present a complete dichotomy of intractability in regard to the depth of the DAG, by proving that this problem is NP-complete for DAGs of depth 3 and W[1]-hard for depth 4 when parameterized by the number of tokens k , and that these bounds are tight. Further, we prove that it is fixed parameter tractable on DAGs parameterized by the combination of treewidth and k . We show that this result applies to undirected graphs, when the number of times a token can visit a vertex is restricted.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases Graph theory, FPT algorithms, Reconfiguration, Independent Sets

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.41

Funding This work benefited from state aid managed by the ANR under France 2030 referenced ANR-23-IACL-0006.

1 Introduction

A reconfiguration problem asks, given two solutions to a combinatorial problem, whether one can change the first solution in a step-by-step manner in order to reach the second one; while ensuring that the transformed set remains a solution throughout the reconfiguration procedure. The research interest in these topics dates back to the 19th century, as many one player games can be considered in these terms [12]. For a more recent introduction to the topic, we refer to [16].

The complexity of reconfiguration problems is often PSPACE-complete. Specific attention has been given to independent set and (connected) dominating set reconfiguration. For these problems, there are two main reconfiguration paradigms: *token sliding*, and *token jumping*. In both paradigms, we view the set to be reconfigured as several tokens placed on the vertices of the graph. Token jumping allows, in each step to move one token to be placed to any other vertex of the graph, while token sliding only allows to move a token to an adjacent vertex (to slide along an edge of the graph).

For both paradigms, independent set reconfiguration is PSPACE-complete, and remains intractable when parameterized by the size of the set to be reconfigured, or the length of reconfiguration sequence [4, 14]. In order to find efficient algorithms, it is therefore interesting to look at structural restrictions on the graphs over which the sets are being reconfigured. While these problems remain PSPACE-hard on planar graphs and graphs of bounded bandwidth, they become polynomial on cographs, forest and interval graphs [10, 13,



© Jona Dirks and Alexandre Vigny;
licensed under Creative Commons License CC-BY 4.0

50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025).

Editors: Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak; Article No. 41; pp. 41:1–41:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

18]. We however note that the two paradigms differ in complexity over bipartite graphs, where independent set reconfiguration for token sliding (ISR-TS) is PSPACE-complete, independent set reconfiguration for token jumping (ISR-TJ) is only NP-complete [13].

The two paradigms differ even more when we look at parameterized complexity. For token jumping, we know that reconfiguration for independent set over general sparse graphs classes, such as bounded degeneracy and nowhere dense, becomes FPT when parameterized by the size of the sets [6, 13, 17]. These classes are generalizations of many well known classes such as planar graph, graph of bounded treewidth, classes excluding a (topological) minor etc.

For token sliding however, the parameterized complexity of independent set reconfiguration is much less understood. On one hand it is known to be FPT on trees [8], graphs of bounded degree [3], and graphs of girth 5 [2], but hard on split graphs [5]. On the other hand, it is still open whether it is tractable on graph of bounded treewidth, or minor-free graphs [6, Question 5]. We refer the readers to the survey [6] for an extensive overview.

We also mention that for both TS and TJ the reconfiguration of every problem that is expressible in MSO_2 is FPT when parameterized by treedepth and the number of tokens [9, Theorem 4.1] and FPT when parameterized by treewidth, the length of the reconfiguration sequence, and the formula describing the considered problem [15, Theorem 7]

Directed graphs. In this paper, we consider directed graphs and allow tokens to only slide in the direction of the edges. Note, that token jumping is not well suited for directed graphs, as arbitrary jumps stay the same in the directed setting. So far very little is known for directed graph's reconfiguration. Recently, Ito et al. [11, Theorem 2] proved that Independent Set Reconfiguration for Directed Token Sliding (ISR-DTS) is NP-hard on DAGs and $W[1]$ -hard when parameterized by the number of tokens. On the tractable side of things, they show that it is linear time solvable on orientation of trees [11, Theorem 3]. The hardness extends to PSPACE-completeness for oriented planar graphs, split graphs, bipartite graphs and most noticeably bounded treewidth graphs [1].

We improve on both the side of hardness, as well as the side of tractability. We first look at the depth of the DAGs and provide sharp dichotomy in both the classical and parameterized setting. In the following statements, k refers to the number of tokens.

► **Theorem 1.1.** *ISR-DTS is polynomial on DAGs of depth 2 and FPT for DAGs of depth 3 when parameterized by k .*

► **Theorem 1.2.** *ISR-DTS is NP-complete on DAGs of depth 3.*

► **Theorem 1.3.** *ISR-DTS is $W[1]$ -hard on DAGs of depth 4, when parameterized by k .*

Additionally, we show that a combination of the parameters k and treewidth leads to tractability in DAGs. We do so by adapting the recently developed framework of galactic reconfiguration [3] to directed graphs. The following result can be considered our main contribution.

► **Theorem 1.4.** *ISR-DTS is FPT on DAGs when parameterized by $k + tw$.*

Finally, we go back to undirected graph, introduce a new parameter (the iteration ι) that measures the maximum number of times a specific token can enter a specific vertex.

► **Theorem 1.5.** *ISR-TS is FPT when parameterized by $k + tw + \iota$.*

These results for graphs of bounded treewidth is a step toward solving one of the most prominent open problem in this line of work (see [6, Question 5]): is ISR-TS FPT when parameterized by k and the treewidth of the graph?

The previous best algorithms only work on the more restricted class of bounded tree-depth [9], or require to also parameterize by the length ℓ of the reconfiguration sequence [15]. In comparison, our new parameter *iteration* is much smaller than the length of the reconfiguration sequence. In terms of techniques, parameterizing by ℓ enabled the authors of [15] to reduce to the model checking problem for MSO formula which is solved using Courcelle's Theorem. In our case, we do not believe that we can reduce to Courcelle's theorem. Thus, the presented algorithm is the first to use dynamic programming on graph of bounded treewidth “by hand” for Independent Set Reconfiguration.

Organization. The rest of the paper is structured as follows. Section 2 provides the required background and fixes notations. Section 3 is devoted to efficient algorithms and the proof of Theorem 1.1. The proof of Theorem 1.2 is provided in Section 4, followed by the proof of Theorem 1.3 in Section 5. In Section 6 we introduce additional notions around galactic reconfiguration for our main algorithm in order to prove Theorem 1.4. Finally, in Section 7 we discuss implications for undirected graphs, and Theorem 1.5.

2 Preliminaries

Graphs. We use standard graph notations. All graphs are directed and loop-free unless stated otherwise. Unless preferable for readability we abbreviate an edge (u, v) with uv . We denote the open out-neighborhood of a vertex v with $N^+(v)$ and the open in-neighborhood with $N^-(v)$. A graph without cycles is called a DAG. The *depth* of a DAG is the number of vertices on its longest directed path. Note that if a DAG has depth d , then its vertices can be grouped in d levels with every edge going from a vertex at some level to a vertex at a greater level. One can inductively define the first layer as the vertices of in-degree 0, and the i th layer as the vertices whose entire in-neighborhood is in smaller levels.

For a directed graph G , the underling undirected graph is obtained by disregarding the edge direction and removing all double edges. That is the graph $(V(G), \{\{u, v\} \mid uv \in E(G) \text{ or } vu \in E\})$. A set of vertices of a directed or undirected graph is called *independent* if the vertices are pairwise nonadjacent.

Parameterized Complexity. In the parameterized framework, a problem is said to be *fixed parameter tractable* (FPT), if there exists an algorithm that solves it in time $f(k) \cdot n^c$ for a computable function f and constant c . An alternative characterization of FPT is through the existence of a *kernel*, that is polynomial time computable function that maps each instance to an instance of size $g(k)$ for some computable function g . The returned instance should be a positive-instance exactly if the inputted instance is. Kernels are commonly presented in the form of data reduction rules, that provide instructions on how to shrink the size of the input.

Under commonly made assumptions, problems that are W[1]-hard are considered not to be FPT. Similar to NP-completeness, we use FPT-reductions to transfer hardness. However, these reductions may use a longer FPT running time but are more restricted in the sense that the parameter of the resulting instance may not depend on the size but only on the parameter of the inputted instance. More background on FPT algorithms and FPT-reduction can be found in the book [7].

Reconfiguration. A configuration α_i is a function from a set of tokens $T = \{t_1, t_2, \dots, t_k\}$ to the set of vertices V .

For two independent sets S and D of size k in a graph G , a *reconfiguration-sequence* for ISR-DTS is a sequence of configurations $(\alpha_i)_{i \in I}$ such that:

1. $\alpha_1(T) = S$ and $\alpha_{|I|}(T) = D$.
2. For every $i \in I$, and for distinct tokens t and t' : $\alpha_i(t) \neq \alpha_i(t')$.
3. For every $i \in I$, and for distinct tokens t and t' : $(\alpha_i(t), \alpha_i(t')) \notin E(G)$.
4. If $\alpha_i(t) \neq \alpha_{i+1}(t)$, then there exists an edge $(\alpha_i(t), \alpha_{i+1}(t)) \in E(G)$ and for every other token $t' \neq t$ it holds that $\alpha_i(t') = \alpha_{i+1}(t')$.

We generally assume that I is a sequence from 1 to $|I|$. For a configuration α_i we call all configurations α_j *succeeding* or a *successor*, if $j > i$. If $j = i + 1$ we say that it is a *direct* successor.

Let t be a token and $\alpha = (\alpha_i)_{i \in I}$ a sequence with $\alpha_1(t) = u$ and $\alpha_{|I|}(t) = v$. We say that the token t *comes from* u and *ends up in* v . Similarly we say that a token *comes from* a subgraph A and *ends up in* a subgraph B if $\alpha_1(t) \in V(A)$ and $\alpha_{|I|}(t) \in V(B)$. Additionally, if there is a token on v we say that this token *blocks* all $u \in N(v)$.

Galactic Graphs. Galactic graphs were recently developed to prove that ISR-TS is in FPT for undirected graphs of bounded degree [3]. We restate their definition, slightly adapted for directed graphs. A *galactic graph* is a graph where the vertex set is partitioned $V(G) = A(G) \cup B(G)$ into planets $A(G)$ and black holes $B(G)$.

For a (galactic) graph G and a subsets of its vertices V' we say that it *collapses* to G' if $G - V' = G' - b$ with $b \in B(G')$ and G' contains an edge (u, b) (respectively (b, u)) if G contains an edge (u, v) (respectively (v, u)) for some $v \in V'$. We denote this graph with $G \odot V'$. For the empty set, we define $G \odot \emptyset = G$. Intuitively, this operation takes a set of vertices (planets and black holes) and merges them into one black hole.

Note, that this operation takes a graph on the left-hand site and a vertex set on the right-hand site. Thus, multiple applications of this operator are left-associative.

We will assume that the black holes are named (or labeled) in such a way that they are uniquely identified by the planets that collapsed into it in the original graph. Meaning that for example for disjoint V_1, V_2 it holds $G \odot V_1 \odot V_2 \odot \{b_1, b_2\} = G \odot V_1 \cup V_2$, where b_1 and b_2 are the black holes created when collapsing V_1 and V_2 respectively. This can be realized by representing planets as sets containing a single element and black holes as the union of the sets representing vertices (planets or black holes) with an additional marker for black holes added.

Galactic Reconfiguration. For two multisets S and D of size k in a graph G , a *galactic reconfiguration sequence* for ISR-DTS is a sequence of configurations $(\alpha_i)_{i \in I}$. Where α_i is a function from a set of tokens $T = \{t_1, t_2, \dots, t_k\}$ to V , such that:

1. $\alpha_1(T) = S$ and $\alpha_{|I|}(T) = D$.
2. For distinct tokens t and t' : $\alpha_i(t) \neq \alpha_i(t')$ or $\alpha_i(t) \in B(G)$.
3. All $\alpha_1(T) \cap A(G), \dots, \alpha_{\ell}(T) \cap A(G)$ are independent sets.
4. If $\alpha_i(t) \neq \alpha_{i+1}(t)$, then there exists an edge $(\alpha_i(t), \alpha_{i+1}(t)) \in E(G)$ and for every other token $t' \neq t$ it holds $\alpha_i(t') = \alpha_{i+1}(t')$.

Note that in a galactic reconfiguration sequence, several tokens are allowed to be on the same vertex (including for the first and last configuration), as long as this vertex is a black hole. When we collapse an instance for ISR-DTS and the collapsed vertex set contains start and end positions, we generally assume that these positions in their multiplicity get assigned to the created black hole.

► **Definition 2.1.** Given a graph G , a set of vertices V_1 , we say that a reconfiguration sequence $(\alpha_i)_{i \in I}$, collapses to $(\beta_j)_{j \in J}$ over $G \odot V_1$ if $(\beta_j)_{j \in J}$ can be obtained by first creating an intermediate reconfiguration sequence $(\beta'_i)_{i \in I}$ where every β'_i is mapping to b (the created black hole) every token that α_i maps to a vertex of V_1 but is otherwise identical. And then removing exhaustively all configurations with an identical direct successor from $(\beta'_i)_{i \in I}$.

Moreover, if a sequence α on a graph G_1 , collapses to a sequence β on a graph G_2 , which itself collapses to a sequence γ on a graph G_3 , we also say that α collapses to γ on G_3 .

Note that in the above definition, $(\beta_j)_{j \in J}$ is a valid reconfiguration sequence, i.e. the tokens only slide along edges and are only on neighboring vertices if at least one of them is a black hole. Furthermore, from this definition, it might look like a single reconfiguration sequence can collapse to multiple sequences on the same graph (depending on the chain of collapses). We later show in Lemma 6.1 that it is not the case, and the collapse notion (with several successive collapses) is therefore well-defined.

Tree-decomposition. A tree decomposition of a undirected graph G is a pair $\mathcal{T} = (T, \text{bag})$ where T is a rooted tree and $\text{bag}: V(T) \rightarrow 2^{V(G)}$ is a mapping that assigns to each node $x \in T$ its bag $\text{bag}(x) \subseteq V(G)$ such that the following conditions are satisfied:

- For every vertex $v \in V(G)$, the set of nodes $x \in V(T)$ satisfying $v \in \text{bag}(x)$ induces a connected subtree of T .
- For every edge $\{u, v\} \in E(G)$, there exists a node $x \in V(T)$ such that $u, v \in \text{bag}(x)$.

We may assume that every internal node of T has exactly two descendants. See [7] for more details on tree decompositions in general. Further, let $\mathcal{T} = (T, \text{bag})$ be a tree decomposition of a graph G . The *adhesion* of a node $x \in V(T)$ is defined as $\text{adh}(x) := \text{bag}(\text{parent}(x)) \cap \text{bag}(x)$ and the *margin* of a node $x \in V(T)$ is defined as $\text{mrg}(x) := \text{bag}(x) \setminus \text{adh}(x)$. The *cone* at a node $x \in V(T)$ is defined as $\text{cone}(x) := \bigcup_{y \succeq_T x} \text{bag}(y)$ and the *component* at a node $x \in V(T)$ is defined as $\text{comp}(x) := \text{cone}(x) \setminus \text{adh}(x) = \bigcup_{y \succeq_T x} \text{mrg}(y)$. Here, $y \succeq_T x$ means that y is a descendant of x in T . The *width* of a tree decomposition is the maximum number of vertices in each bag (minus one). The *treewidth* of a graph G , noted $\text{tw}(G)$, is the minimum width among all possible tree decompositions of G . For directed graphs, we define these terms over their underlying undirected graphs.

3 Tractability Results for DAGs of Low Depth

This section is devoted to the proof of Theorem 1.1 that we restate here.

► **Theorem 1.1.** *ISR-DTS is polynomial on DAGs of depth 2 and FPT for DAGs of depth 3 when parameterized by k .*

We prove the first part of the theorem.

► **Lemma 3.1.** *For DAGs of depth 2 ISR-DTS is polynomial solvable.*

Proof. We are given a DAG G and two independent sets S and D both of size k . First, we can assume that $S \cup D$ is the vertex set of the entire graph, as for every other vertex v no token can start in S , go to v and then end in D , contradicting that G is a DAG of depth 2.

If (G, S, D) is a positive instance, there needs to be at least one vertex u in D that has just a single neighbor v in S , as otherwise no token can move. We can therefore start the reconfiguration sequence by moving the token from u to v . We can now remove u, v from G and proceed inductively in the same way. If at any point no token can move, we report that the input is a negative instance to ISR-DTS. ◀

After this small warm up proof, let us now move to the parameterized tractability result. Let us first show, that we can assume the input to have a uniform structure.

► **Lemma 3.2.** *For any instance (G, S, D) if G is a DAG of depth 3 we can in polynomial time find an equivalent instance (G', S', D') such that S' is equal to the first layer of G' and D' to the third and where G' is also a DAG of depth 3.*

Proof. First note, that if there is a vertex from $S - D$ at the third layer, it can reach nothing, and thus we can return a trivial no instance. While a vertex in $S \cap D$ in the third layer cannot move, blocking its in-neighborhood, so we can safely remove this vertex together with its in-neighborhood. By analogous arguments the same holds if there is a vertex from D at the first layer. Additionally, we can remove all vertices without predecessors in S (as they cannot be reached) and all vertices with no successors in D (as a token here cannot reach a destination).

Observe, that after this a vertex from S may not lie on the second layer. If it were so, it would have a predecessor, which then would lie in the first layer. Thus, it would be in S which means that S is not an independent set.

Furthermore, if a vertex $v \in D$ is in the second layer, it may not have successors. Thus, we can add a new vertex v' make it a successor to all predecessors of v as well as v itself. Then, while keeping v in the graph, we remove v from D and add v' instead.

Note that all of these basic checks as well as basic graph manipulations can be carried out in polynomial time. ◀

We show that the following reduction rule, when applied exhaustively, yields a kernel:

► **Rule 3.3.** *If a DAG of depth 3 has two vertices $u, v \notin S \cup D$ with $N(u) = N(v)$ then remove u .*

Proof of Correctness. Let G' be the graph obtained from applying the reduction rule on G . Clearly if there is a reconfiguration sequence in G' one can use the same sequence in G . Thus, we only have to show, that if there is a reconfiguration sequence in G there is one in G' .

Let $(\alpha_i)_{i \in I}$ be the reconfiguration. We show that every time $(\alpha_i)_{i \in I}$ places a token on u we can place it on v instead. As u and v have the same neighborhood, the independent set condition is not violated and neither is the requirement of moving tokens only along edges. That is, if there is no token at v at that same step. However, a token on v , blocking all of $N(u)$, would prohibit the placement of a token on u . ◀

► **Lemma 3.4.** *ISR-DTS admits a kernel on DAGs of depth 3 when parameterized by k .*

Proof. We first use Lemma 3.2 to obtain an instance, where the first layer is equal to S and the third layer to D . Then we apply Rule 3.3. Note, that now every vertex on the second layer has a distinct neighborhood which is a subset of $S \cup D$. As there are at most $2^{|S \cup D|}$ unique subsets, the total number of vertices is bound by $|S| + |D| + 2^{|S \cup D|} = 2k + 4^k$. ◀

Together Lemma 3.4 and Lemma 3.1 complete the proof of Theorem 1.1.

4 NP completeness for DAGs of depth 3

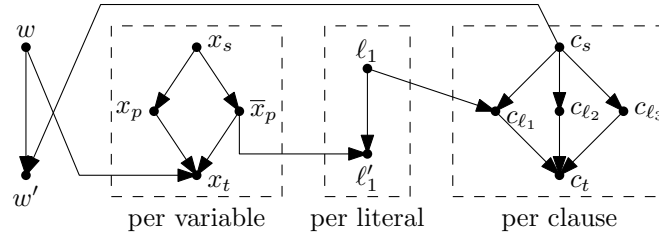
In this section we turn to hardness results, improving the lower bounds of [11]. While they prove that ISR-DTS is NP-hard on DAGs, their reduction yields DAGs of unbounded depth.

► **Theorem 1.2.** *ISR-DTS is NP-complete on DAGs of depth 3.*

The proof of Theorem 1.2 is obtained by reducing from 3-SAT. For a 3-SAT sentence φ we construct the following instance $(G_\varphi, S_\varphi, D_\varphi)$:

1. Add a timing gadget, meaning two vertices $w \in S_\varphi$ and $w' \in D_\varphi$ as well as an edge ww' .
2. For each variable x create four vertices $x_s \in S_\varphi$, $x_t \in D_\varphi$, x_p and $\bar{x}_p$ as well as edges $x_s x_p$, $x_s \bar{x}_p$, $x_p x_t$ and $\bar{x}_p x_t$. Further, connect this subgraph to the timing gadget with the edge $w x_t$.
3. For each clause c with the literals ℓ_1, ℓ_2, ℓ_3 add five vertices $c_s \in S_\varphi$, $c_t \in D_\varphi$ and $c_{\ell_1}, c_{\ell_2}, c_{\ell_3}$. Further, add edges from c_s to c_{ℓ_1}, c_{ℓ_2} and c_{ℓ_3} as well as from c_{ℓ_1}, c_{ℓ_2} and c_{ℓ_3} to c_t . Additionally, connect this subgraph to the timing gadget with the edge $c_s w'$.
4. For every literal ℓ add two vertices $\ell \in S_\varphi$ and $\ell' \in D_\varphi$ and edge (ℓ, ℓ') . For all $\bar{x} = \ell$ (note that $\bar{\bar{x}} = x$) and c containing x add edges (ℓ, c_ℓ) and $x_p \ell'$ respectively.

This construction is represented in Figure 1



■ **Figure 1** Schematic of the graph resulting from a 3-SAT formula, where the clause c contains the positive literal x .

► **Lemma 4.1.** *If φ is satisfiable then $(G_\varphi, S_\varphi, D_\varphi)$ is a positive instance for ISR-DTS.*

Proof. Let σ be a satisfying assignment of φ . We define the following reconfiguration sequence. In no particular order move a token from x_s to x_p if $\sigma(x) = \text{True}$, to $\bar{x}_p$ otherwise.

For all literals where it is possible without violating the independent set condition, slide along (ℓ, ℓ') . I.e. this move is possible for every literal of the form x (resp. $\bar{x}$) where σ sets x to True (resp. to False).

As each clause c has at least one literal ℓ that evaluates to true. Move the token from c_s to c_t over c_ℓ . This move is possible as the token in the “literal gadget” moved from ℓ to ℓ' . Now one is able to move the token from w to w' . Next, move the tokens on all x_p or $\bar{x}_p$ to x_t and finally from all remaining ℓ to ℓ' . ◀

► **Lemma 4.2.** *If $(G_\varphi, S_\varphi, D_\varphi)$ is a positive-instance of ISR-DTS then φ is satisfiable.*

Proof. We first show, that the flow of token is tightly controlled.

▷ **Claim 4.3.** The token coming from w will only slide along the edge ww' . Further, for all variables x , clauses c and literals ℓ the tokens from x_s , c_s and ℓ will end up in x_t , c_t and ℓ' respectively.

Proof. By reachability in G_φ the only possibility where this is not the case would be if some token coming from x_s ends up in ℓ' (for $x = \ell$). Implying the token from ℓ ends up in c_t ; implying that the token from c_s ends up in w' ; implying that the token from w slides to x_t . However, as w blocks w' , c_s blocks c_ℓ , ℓ blocks ℓ' and x_t locks in x_s , so none of these moves is doable. ◀

Let $(\alpha_i)_{i \in I}$ be a reconfiguration sequence certifying that $(G_\varphi, S_\varphi, D_\varphi)$ is a positive-instance. The previous claim implies that the token in w ends up in w' , so let α_i be the first configuration where a token occupies w' . We define the following set $X' = \{\ell' \in \alpha_i(T) \mid \ell \text{ is literal}\}$.

▷ **Claim 4.4.** The set X' contains one literal from each clause. Further, if ℓ is in X' , then $\bar{\ell}$ is not.

Proof. By Claim 4.3, for all clauses c the token coming from c_s is either at some $c_\ell \in \{c_{\ell_1}, c_{\ell_2}, c_{\ell_3}\}$ or is already on c_t and has previously been in one of the c_ℓ . Thus, no token can be at its neighbor ℓ (on the literal gadget). By Claim 4.3 there is now a token on ℓ' .

For the second claim, assume towards a contradiction, that in α_i tokens are placed at both ℓ and $\bar{\ell}$ for some literal ℓ . This implies that neither for α_i nor for any succeeding configuration a token is at x_p or $\bar{x}_p$ where x is the variable of the literal ℓ . As no token is on x_t there is one at x_s and by Claim 4.3 it cannot reach a valid end position. ◀

We find the following assignment $\sigma()$. Set $\sigma(x) = \text{True}$, if $x \in X'$ and otherwise $\sigma(x) = \text{False}$. Observe, that all literals in X' evaluate to true. As X' contains at least one literal per clause all clauses evaluate to true. ◀

This concludes the proof of Theorem 1.2.

5 W[1]-hardness for DAGs of Depth 4

In this section, we prove the second lower bound of this paper. Here, k is the number of tokens.

► **Theorem 1.3.** *ISR-DTS is W[1]-hard on DAGs of depth 4, when parameterized by k .*

We show this result by providing a FPT-reduction from Independent Set. That is given an instance for independent set (with parameter k), we build an instance of ISR-DTS with parameter $f(k)$, such that the input instance is a positive instance if and only if the resulting one is. Section 5.1 describes the construction while Lemma 5.1 in Section 5.2 proves the correctness of the reduction.

5.1 Construction

The reduction takes as input an undirected graph G of n vertices and an integer k and outputs the graph as described in the following. This graph is also depicted in Figure 2.

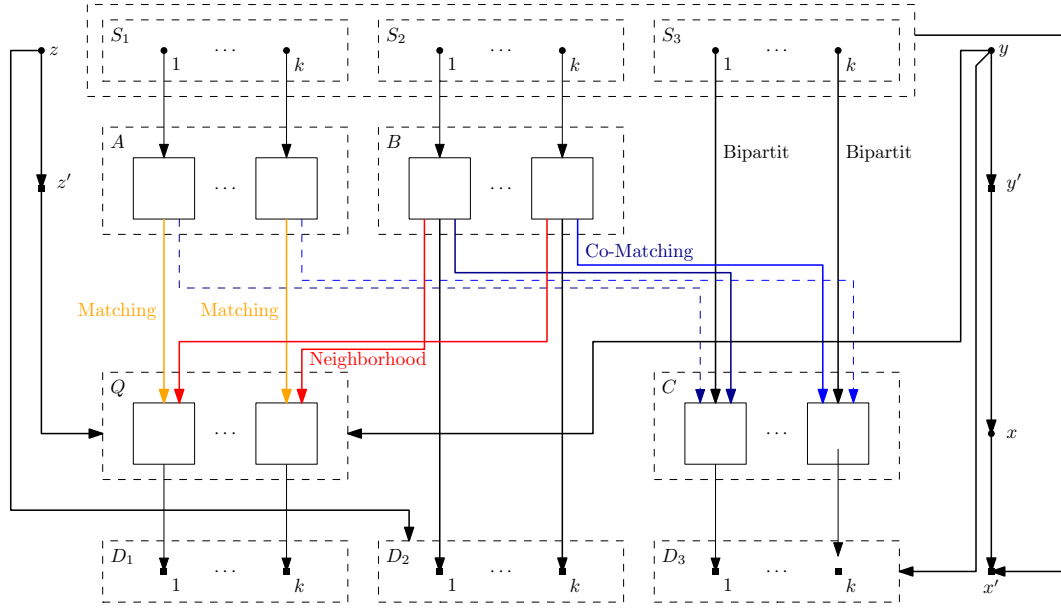
Vertices. In total the DAG consists of 10 main blocs and three helping gadgets. Let us commence with the main blocs. Six of these consist of k isolated vertices. Half of the blocs: S_1 , S_2 and S_3 contains only starting positions: $s_{1,1}$ to $s_{3,k}$. The other half: D_1 , D_2 , and D_3 only contain target (or destination) vertices: $d_{1,1}$ to $d_{3,k}$.

The remaining four components are each made up of k copies of the vertex set of G , i.e. kn isolated vertices. The components are named A , B , C , Q , and, given $i \leq k$, $v \in V(G)$, we call $a_{i,v}$ (resp. $b_{i,v}$, $c_{i,v}$, $q_{i,v}$) the copy of v in the i th copy A_i of $V(G)$ in A (resp. B , C , Q).

Lastly, we have six specific vertices, controlling the flow of tokens in the graph: x , x' , y , y' , z , z' , where x, y, z are starting positions, and x', y', z' destinations.

Edges. The edges of our construction are represented in Figure 2 and are as follows.

1. Full connections: For every $i \leq k$ and v in $V(G)$, connect $s_{1,i}$ (resp. $s_{2,i}$, $s_{3,i}$) to $a_{i,v}$ (resp. $b_{i,v}$, $c_{i,v}$). Further, connect $q_{i,v}$ (resp. $b_{i,v}$, $c_{i,v}$) to $d_{1,i}$ (resp. $d_{2,i}$, $d_{3,i}$).
2. Matching connections: For every $i \leq k$, and v in $V(G)$, connect $a_{i,v}$ to $q_{i,v}$. I.e. make a matching between A_i and Q_i .



■ **Figure 2** Conceptual view on the graph resulting from the reduction. Arrows represent directed edges between vertices in the two blocs they connected. Unless labeled otherwise the connected is complete. Dashed arrows have no different meaning and are only stylized for clarity.

3. Co-matching connections: For every $i \leq k$, and u, v in $V(G)$ if and only if $v \neq u$, connect $a_{i,v}$ to $c_{i,u}$ and connect $b_{i,v}$ to $c_{i,u}$. I.e. Make a co-matching between A_i and C_i and between B_i and C_i .
4. Neighborhood connections: For every $i, j \leq k$, and u, v in $V(G)$ if and only if $i \neq j$ and $u \in N^G(v)$, connect $b_{i,v}$ to $q_{j,u}$.
5. Gadget connections: Connect x and each vertex in S_1 , S_2 , and S_3 to x' . Connect y' to x . Further, connect y to y' , and to each vertex in Q and D_3 . Lastly, connect z to z' and to each vertex in D_2 , and connect z' to each vertex in Q .

We define the start positions as $S := S_1 \cup S_2 \cup S_3 \cup \{x, y, z\}$ and destination positions as $D := D_1 \cup D_2 \cup D_3 \cup \{x', y', z'\}$.

We call our resulting DAG H . Notice that H has $4kn + 6k + 6$ many vertices, and $3k + 3$ start / destination vertices. Furthermore, the DAG has depth 4 as illustrated in Figure 2.

5.2 Correctness

This section is devoted to the proof of the correctness of the reduction, as stated in the following lemma.

► **Lemma 5.1.** *If the input G, k is a positive instance for the independent set problem if and only if (H, S, D) is a positive ISR-DTS instance.*

We now prove the first implication. Assume that $X = \{v_1, \dots, v_k\} \subseteq V(G)$ is an independent set of G of size k . We move the tokens in H as follows.

1. For every $i \leq k$, move the token from $s_{3,i}$ to c_{i,v_i} . Similarly, move tokens from $s_{1,i}$ and $s_{2,i}$ to a_{i,v_i} , and b_{i,v_i} respectively.
2. Move the token from x to x' and the token from y to y' .
3. For every $i \leq k$, move the token from a_{i,v_i} to q_{i,v_i} . And then from q_{i,v_i} to $d_{1,i}$.
4. Move the token from z to z' .
5. For every $i \leq k$, move the token from b_{i,v_i} to $d_{2,i}$ and the tokens from c_{i,v_i} to $d_{3,i}$.

The first set of moves is possible because a token on c_{i,v_i} blocks in A_i every $a_{i,u}$ with $u \neq v_i$, but a_{i,v_i} is free. The same holds for b_{i,v_i} . Second, the clocks x, y can move to x', y' because S_1, S_2 , and S_3 , are free of token, freeing D_3 . Third, X is an independent set in G , and therefore $v_i \in X$ is not adjacent to any $v_j \in X$ for $j \neq i$. Therefore, q_{i,v_i} is not adjacent to any b_{j,v_j} for $j \neq i$. Hence, the tokens can move from A to Q and then to D_1 . Forth, as Q is empty of token, z can move to z' , freeing D_2 . Last, the tokens in B and C can freely move to D_2 and D_3 respectively.

This shows that (H, S, D) is a positive ISR-DTS instance, and conclude the first part of the proof of Lemma 5.1. The second implication in the statement of Lemma 5.1 is harder to prove. We show that any reconfiguration sequence must follow some structure, from which we derive the existence of an independent set in G . In what follows, we assume that (H, S, D) is a positive ISR-DTS instance.

▷ **Claim 5.2.** For any reconfiguration sequence, the tokens in x, y , and z end up in x', y' , and z' respectively. Furthermore, they move in this order.

Proof. While other tokens can reach x' , the token on x can only reach x' . Also, the token in y (resp. z) is the only one that can reach y' (resp. z').

Additionally, notice that both y and z' block all of Q . As the token reaching D_1 need to go through Q , if z moves to z' before y moves to y' no token can ever reach Q . So y must reach y' before z reaches z' . And since x blocks y' , x must slide to x' first. ◁

▷ **Claim 5.3.** For any reconfiguration sequence, no token slides along the edge uv , if uv is in $A \times C, B \times C$ or $B \times Q$.

Proof. As the tokens coming from the clocks stay within them, we may argue as if these vertices and their respective tokens were not present.

The vertices in D_2 can only be reached from S_2 , thus no token may leave a path from some vertex in S_2 to some vertex in D_2 which prohibits the use of edges between B and C or B and Q . Consequently and similarly, the only tokens that can end up in D_1 are these that start in S_1 , prohibiting the use of the edges between A and C . ◁

▷ **Claim 5.4.** There exists a set $X = \{v_1, \dots, v_k\} \subseteq V(G)$ such that when y moves to y' , there are tokens on vertices a_{i,v_i}, b_{i,v_i} , and c_{i,v_i} for every $i \leq k$.

Proof. When y moves to y' , x already moved to x' , so S_1, S_2 , and S_3 are empty of tokens. Furthermore, as y and z did not move yet, D_2, D_3, Q and therefore D_1 are empty of token. So the tokens in S_3 moved to C_3 . We then define v_i as the vertex of $V(G)$ such that $s_{3,i}$ moved to c_{i,v_i} . Finally, note that c_{i,v_i} blocks in A_i every $a_{i,u}$ with $u \neq v_i$. So the token from $s_{1,i}$ must now be in a_{i,v_i} , as these tokens may not move to other blocs by Claim 5.3. Similarly, the token from $s_{2,i}$ must now be in b_{i,v_i} . ◁

▷ **Claim 5.5.** X is an independent set of size k in G .

Proof. When z goes to z' , all of Q is blocked, and remains so for the rest of the reconfiguration. So the vertices in D_1 are already occupied with tokens. These tokens must be from A by Claim 5.3. So before z goes to z' each token in a_{i,v_i} must go to q_{i,v_i} (there is only one outgoing edge from a_{i,v_i}), and then to $d_{1,i}$. During these moves, the tokens in B remains there since D_2 is still blocked by z . If a_{i,v_i} can freely move to q_{i,v_i} it implies that no token on b_{j,v_j} is adjacent to q_{i,v_i} which implies that no v_j (with $j \neq i$) is adjacent to (or equals) v_i in G . As this holds for every $i \leq k$, we have that X is an independent set in G . ◁

In conclusion, the existence of a reconfiguration sequence implies the existence of an independent set in G . This concludes the proof of Lemma 5.1, and therefore of Theorem 1.3.

6 FPT when parameterized by Treewidth and k

Now that we have covered the impact of the depth of a DAG on the tractability of ISR-DTS, we look at other subfamily of DAGs that could enable FPT algorithms. Recently, Ito et al.[11, Theorem 3] proved that ISR-DTS is polynomial time solvable on DAGs whose underlying graph is a tree. We extend this results by looking at graph of bounded treewidth.

► **Theorem 1.4.** *ISR-DTS is FPT on DAGs when parameterized by $k + tw$.*

We first prove several results on galactic reconfiguration and the impact of collapses that we defined in Section 2. Then, we explain how to perform a dynamic algorithm on a tree decomposition.

6.1 Reconfiguration and Collapses

The first lemma shows that a reconfiguration defined on a graph only has one possible collapse on a collapse of the graph. Meaning that it does not matter in which sequence of collapse the new graph is obtained.

► **Lemma 6.1.** *Let G_1, G_2, G'_2 , and G_3 galactic graphs such that both G_2 and G'_2 are obtained from G_1 through collapses, and G_3 is obtained from both G_2 and G'_2 through collapses.*

Let $\alpha, \beta, \beta', \gamma$ be reconfiguration sequences over G_1, G_2, G'_2, G_3 respectively. If α collapses to β , α collapses to β' and β collapses to γ , then β' collapses to γ .

Proof. Let γ' be the sequence β' collapses to in G_3 . We show that $\gamma = \gamma'$.

Let $H_1, H_2, \dots, H_{|I|}$ be the sequence of graphs in the chain of collapses from G_1 over G_2 to G_3 . For each $i \in I \setminus \{|I|\}$ let f_i denote the function that maps vertices (planets and black holes) from H_i to vertices H_{i+1} they correspond to. That is, to itself if they were not collapsed and to the created black hole otherwise.

We have to account for the deletion of consecutive equal configurations, when collapsing reconfiguration sequences. Therefore, let g_i be the function that maps indices of the reconfiguration sequence that α collapses to in H_i to the corresponding indices in the reconfiguration sequence that α collapses to in H_{i+1} .

Let $f = f_1 \circ f_2 \circ \dots \circ f_{|I|-1}$ and define g analogously. Further, define f' and g' analogously for the chain of collapses in from G_1 to G'_2 and finally G_3 . By definition $\gamma_{g(j)}(t) = f(\alpha_j(t))$ for each configuration α_j and each token t . Similarly, the same holds for γ' with f' and g' .

As the resulting graph for both chains of collapses is the same, $f = f'$. Further, as equality classes of configurations only merge when collapsing, we may assume that consecutive equal configurations get resolved only at the end of a chain of collapses. Thus as $f = f'$ also $g = g'$. Thus in conclusion, for all token t and steps $g(i)$: $\gamma'_{g(i)}(t) = f'(\alpha_i(t)) = f(\alpha_i(t)) = \gamma_{g(i)}(t)$. ◀

Observe that these lemma can be iterated to obtain the same statement for arbitrary long chains of collapses, essentially proves that the collapsing is transitive.

The next lemma shows that on DAGs, reconfiguration are restricted.

► **Lemma 6.2.** *Let α be a reconfiguration sequence over a DAG G without back holes. Then for every $i < j < \ell$ and every token t we have that $\alpha_i(t) = \alpha_\ell(t)$ implies $\alpha_i(t) = \alpha_j(t)$.*

Furthermore, if β is a collapse of α , then for all $i < j < \ell$ and token t such that $\beta_i(t) = \beta_\ell(t)$ and $\beta_i(t) \neq \beta_j(t)$, we have that $\beta_i(t)$ is a black hole.

Proof. The first assertion is trivial. A token returning to a vertex would imply a round walk and thus a cycle. Assume that $\beta_i(t) = \beta_\ell(t) \in A(G)$ there exists corresponding indices $i' < j' < \ell'$ with $\alpha_{i'}(t) = \alpha_{\ell'}(t) \neq \alpha_{j'}(t)$ – contradicting the first statement. ◀

This in turn bound the number of possible reconfiguration sequences that can be on a DAG or its collapses.

► **Lemma 6.3.** *For a DAG without black holes G and a graph G' obtained by collapses from G where additionally G' does not contain neighboring black holes; the number and length of reconfiguration sequences over G' that are collapses of reconfiguration sequences over G is bounded by $|V(G')|^{2k^2|V(G')|}$.*

Proof. No token can return to a planet after leaving it, due to Lemma 6.2. A token might be at a black hole multiple times. However, as there are no repeating consecutive configurations and by assumption no neighboring black holes, a change for a planet has to happen in between consecutive configurations. This bounds the length of every sequence by $2k|V(G')|$.

As there are only $|V(G')|^k$ possible configurations the total number of possible reconfiguration sequences is at most $(|V(G')|^k)^{2k|V(G')|} = |V(G')|^{2k^2|V(G')|}$. ◀

Our last lemma of this section is a composition lemma that explains how to glue together two reconfiguration sequences defined on distinct collapses of a given graph.

► **Lemma 6.4 (Composition lemma).** *Let G be a graph and U, V disjoint sets of vertices such that U has no neighbors in V and vice versa. Let $(\alpha_i)_{i \in I}$ and $(\beta_j)_{j \in J}$ be reconfiguration sequences, for $G \odot U$ and $G \odot V$ respectively, that collapse to the same sequence in $G \odot U \odot V$. Then there exists a reconfiguration sequence in G that collapses to $(\alpha_i)_{i \in I}$ and $(\beta_j)_{j \in J}$ over $G \odot U$ and $G \odot V$ respectively.*

Proof. Let $G, U, \alpha = (\alpha_i)_{i \in I}$ and $\beta = (\beta_j)_{j \in J}$ defined as above. Let b_U and b_V the black holes in $G \odot U \odot V$ corresponding to the sets U and V respectively and let $\gamma = (\gamma_\ell)_{\ell \in L}$ be the shared collapsed reconfiguration sequence in $G \odot U \odot V$.

We build a configuration sequence $(\delta_d)_{d \in D}$ over G by concatenating sequences δ^ℓ (for each $\ell \in L$).

Broadly speaking, fix $\ell \in L$ and the configuration γ_ℓ , and look at the configurations in $(\alpha_i)_{i \in I}$ and $(\beta_j)_{j \in J}$ that correspond to it. All these configurations in α are equivalent when restricted to $G \odot U \odot V$, therefore tokens only move in V (and are all mapped to b_V in γ_ℓ). Similarly, the configuration in β corresponding to this γ_ℓ only differ in U . So in δ we can perform all these move independently before moving to the configuration $\gamma_{\ell+1}$.

Formally let $I_\ell \subseteq I$ the set of index of $(\alpha_i)_{i \in I}$ corresponding to γ_ℓ , and define J_ℓ similarly. Let i_ℓ be the minimum index of I_ℓ minus one and let j_ℓ be the minimum index of J_ℓ . Let δ^ℓ be the sequence of length $|I_\ell| + |J_\ell| - 1$ such that for every integer d and token t , we have:

$$\delta_d^\ell(t) := \begin{cases} \alpha_{i_\ell+d}(t) & \text{if } \alpha_{i_\ell+d}(t) \neq b_U & \text{and if } d \leq |I_\ell| \\ \beta_{j_\ell}(t) & \text{if } \alpha_{i_\ell+d}(t) = b_U & \text{and if } d \leq |I_\ell| \\ \beta_{j_\ell+d-|I_\ell|}(t) & \text{if } \beta_{j_\ell+d-|I_\ell|}(t) \neq b_V & \text{and if } d > |I_\ell| \\ \alpha_{i_\ell+|I_\ell|}(t) & \text{if } \beta_{j_\ell+d-|I_\ell|}(t) = b_V & \text{and if } d > |I_\ell| \end{cases}$$

Note that the definition is consistent as $\alpha_i(t) = \beta_j(t) = \gamma_\ell(t)$ for every $i \in I_\ell$ and every $j \in J_\ell$ as soon as $\gamma_\ell(t)$ is not one of the black hole b_U or b_V . Finally, define δ as the concatenation of every δ^ℓ for $\ell \in L$.

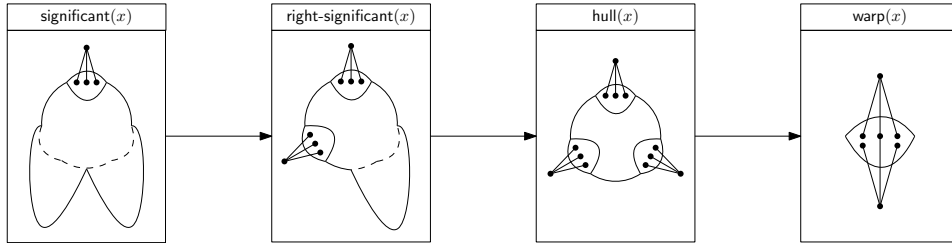
To see that δ is indeed reconfiguration sequence, note that each time only one token moves (according to the move defined in α or β). Furthermore, since U and V are disjoint and non-adjacent the set of tokens has to be an independent set (when only considering planets). Otherwise, this inconsistency already holds in α or β .

By construction, δ and α (resp. β) agree on the positioning of tokens except for b_U (resp. b_V). And therefore δ collapses to α (resp. β) on $G \odot U$ (resp. $G \odot V$). ◀

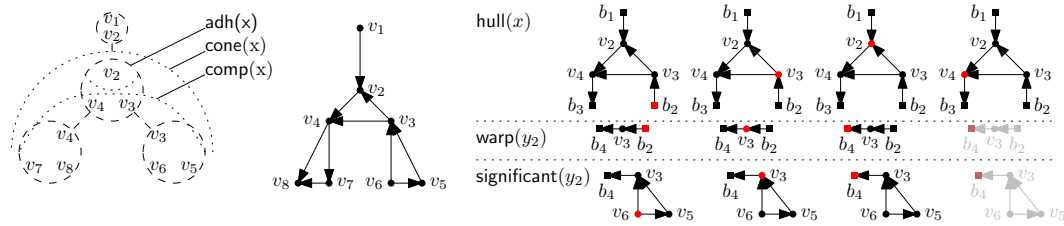
6.2 Dynamic Programming

In a nutshell, our algorithm will try every possible reconfiguration sequence for each bag. Starting from the leaf, the information of what are the possible reconfiguration sequences is computed by a bottom up manner. At every node x the algorithm tries to glue the possible reconfiguration sequences from the left and from the right children, checking that they agree on the overlapping part. Thanks to Lemma 6.3 we only have to consider small sequences at the bag level, and the gluing can be done thanks to Lemma 6.4. We now turn to formal definitions. We consider a DAG G , and its tree-decomposition as defined in Section 2.

Let x be a node of the tree-decomposition and y_1, y_2 its children. We define the *significant* of x noted $\text{significant}(x)$ as $G \odot (V \setminus \text{cone}(x))$. Intuitively, everything that is “above” x in the tree decomposition has been collapsed into one black hole. The *right-significant*, noted $\text{right-significant}(x)$ as $\text{significant}(x) \odot \text{comp}(y_1)$ and $\text{hull}(x)$ as the graph $\text{right-significant}(x) \odot \text{comp}(y_2)$. I.e. where one (or both) subtrees of x have collapsed into a black hole. Finally, we define $\text{warp}(x)$ as $\text{hull}(x) \odot \text{mrg}(x) \cup \{b_1, b_2\}$ where b_1 (respectively b_2) is the black hole obtained from collapsing $\text{comp}(y_1)$ (resp $\text{comp}(y_2)$). Note that $\text{warp}(x) = \text{significant}(x) \odot (\text{comp}(x))$. I.e. it is only composed of element of the adhesion of x with two black holes representing everything that is “above”, and “below” in the tree decomposition. Observe that $\text{hull}(x)$ can be collapsed to obtain all $\text{warp}(x)$, $\text{warp}(y_1)$ and $\text{warp}(y_2)$ in a single step. All these notions are represented in Figure 3, and an example of reconfiguration is presented in Figure 4.



■ **Figure 3** Collapsing different graphs into each other.



■ **Figure 4** A graph, its tree-decomposition (left) and reconfiguration sequences with one token (red) over collapsed graphs (right), where y_2 is the right child of x . The reconfiguration over $\text{hull}(x)$ and $\text{significant}(y_2)$ can be glued together using Lemma 6.4 as they collapse to the same sequence over $\text{warp}(y_2)$. The reconfiguration sequences are presented with repeating configuration for clarity.

► **Definition 6.5.** Let G be a DAG, together with a tree decomposition, let x be a node of the tree decomposition. The profile of x is the set of reconfiguration sequences α over $\text{warp}(x)$ such that there is a reconfiguration β over $\text{significant}(x)$ that collapses to α over $\text{warp}(x)$.

Thanks to Lemma 6.3, the profile of each node is bounded by $\text{tw}^{2k \cdot \text{tw}}$, where tw is the width of the given tree decomposition. We now show that we can compute the profile of each node in a bottom up manner.

► **Proposition 6.6.** *Given a tree decomposition of a DAG G of width tw , a node x of the decomposition, and the profile of its children y_1 and y_2 , we can compute the profile of x in time $f(k, tw)$, for some computable function f .*

Proof. For the leaf of the tree-decomposition, it holds trivially, as we can compute all possible sequences in $\text{significant}(x)$ as it is of size at most tw . Thus, we may assume that x is not a leaf. For every possible reconfiguration sequence α over $\text{warp}(x)$ the algorithm adds it to the profile of x if and only if the following holds.

There are reconfiguration sequences β_1 , β_2 , and γ over $\text{warp}(y_1)$, $\text{warp}(y_2)$, and $\text{hull}(x)$ respectively, such that :

1. β_1 (resp. β_2) belongs to the profile of y_1 (resp. y_2), and
2. γ collapses to α (resp. β_1, β_2) over $\text{warp}(x)$ (resp. $\text{warp}(y_1), \text{warp}(y_2)$).

▷ **Claim 6.7.** This checks can be performed in time $f(k, tw)$.

Proof. By Lemma 6.3, there can be only few possible sequences to consider for γ . We can then compute its collapses in $\text{warp}(y_1)$ and $\text{warp}(y_2)$, and check that the results belongs to the profiles of y_1 and y_2 that have already been computed. All of this can therefore be computed in time $f(k, tw)$. ◀

▷ **Claim 6.8.** If there exists a reconfiguration sequence δ over $\text{significant}(x)$ that collapses to α in $\text{warp}(x)$ then the algorithm adds α to the profile of x .

Proof. We first collapse δ to a reconfiguration γ in the graph $\text{hull}(x)$. By Lemma 6.1 we obtain that γ also collapses to α over $\text{warp}(x)$.

Now, we collapse δ to a sequence β'_1 over $\text{significant}(y_1)$, and both collapse to a sequence β_1 over $\text{warp}(y_1)$. Since β'_1 is in the significant of y_1 , β_1 is in the profile of y_1 . We define β'_2 and β_2 similarly for y_2 .

Note that this triple $(\gamma, \beta_1, \text{ and } \beta_2)$ must be considered by the algorithm. Finally, by Lemma 6.1, γ also collapses to β_1 and β_2 . So this triple of reconfiguration sequences passes the checks, and the algorithm selects α to the profile of x . ◀

▷ **Claim 6.9.** If the algorithm adds α to the profile of x , then there exists a reconfiguration sequence δ in $\text{significant}(x)$ that collapses to α in $\text{warp}(x)$

Proof. Let γ be the guessed sequence in $\text{hull}(x)$ and β_1, β_2 be the sequences that γ collapses to in $\text{warp}(y_1), \text{warp}(y_2)$ respectively, where y_1 and y_2 are the left and right children of x . As β_1 and β_2 are in the profile of y_1 and y_2 respectively, by definition of profile, there exist β'_1 and β'_2 in $\text{significant}(y_1)$ and $\text{significant}(y_2)$ that collapse to β_1 and β_2 in $\text{warp}(y_1)$ and $\text{warp}(y_2)$ respectively.

Now we use the Composition Lemma 6.4 on γ and β'_2 to first get a reconfiguration sequence γ' in $\text{right-significant}(x)$. This is possible because they both collapse to β_2 according to the check of the algorithms, and because by construction the vertices in $\text{comp}(y_2)$ are only adjacent to the vertices of $\text{adh}(y_2)$ (which are in $\text{warp}(y_2)$) and not to the other vertices of $\text{right-significant}(x)$.

As γ' collapses to γ , and the algorithm checked that γ collapses to β_1 , by Lemma 6.1, γ' also collapses to β_1 . We can finally use again our Composition Lemma 6.4, because both β'_1 and γ' collapse to β_1 over $\text{warp}(y_1)$. And as before, the vertices of $\text{significant}(y_1)$ that are not in $\text{warp}(y_1)$ (i.e. the vertices of $\text{comp}(y_1)$) are only adjacent to the vertices of $\text{adh}(y_1)$, which are in $\text{warp}(y_1)$, and not to the other vertices of $\text{significant}(x)$. This yields a reconfiguration sequence δ over $\text{significant}(x)$. With several uses of Lemma 6.1, we obtain that δ which collapses to γ' also collapses to γ , and therefore also collapses to α .

In conclusion, this sequence α is indeed in the profile of x . ◀

This therefore concludes the proof of Proposition 6.6. ◀

With all of that in mind, we can also conclude the proof of Theorem 1.4. Thanks to Proposition 6.6, we compute the profile of every node in time linear in the number of nodes in the tree decomposition (with constant factors depending on k and tw). As the significant of the root is the entire graph, the non-emptiness of the profile of the root certifies the existence of a reconfiguration sequence in the entire graph. Such a reconfiguration sequence can be computed in a top-down traversal of the tree decomposition, selecting at each level a sequence from the profile that is compatible with the one selected by the parent.

7 Back to undirected graphs

One of the biggest open problem in token sliding reconfiguration is whether ISR-TS parameterized by k is FPT on graphs of bounded treewidth (see [6, Question 5]). Our study of directed graphs and Theorem 1.4 can be seen as a step towards solving this open problem. In fact the techniques that we develop in Section 6 have consequences for undirected graphs.

For example, note that the only place in the proof of Theorem 1.4 where we use that the given graph is directed and is a DAG is in Lemma 6.2. The DAG property is only used to show that a token never goes twice to the same vertex. This in turn bounds the number of possible reconfiguration sequences in collapses of our original DAG (Lemma 6.3).

Our proof then directly provide the following result: Given an (undirected) instance (G, S, D) for ISR-TS, deciding whether there is a reconfiguration sequence where no token goes twice to the same vertex can be decided in FPT, parameterized by $k + \text{tw}(G)$.

To generalize this idea, we introduce another parameter.

► **Definition 7.1.** *Given a reconfiguration sequence $(\alpha_i)_{i \in I}$, a token t and a vertex v , the iteration of (v, t) is the number of time the token enters v (plus one if t starts in v). The iteration ι of a reconfiguration sequence $(\alpha_i)_{i \in I}$ is the maximum iteration of all pair (v, t) :*

$$\iota := \max_{(v,t) \in V \times T} |\{0 \mid \alpha_0(t) = v\} \cup \{i \in I \mid 0 < i \text{ and } \alpha_{i-1}(t) \neq \alpha_i(t) = v\}|.$$

For galactic graphs, the iteration is only defined on planets, with no constraints on the number of time a token enters a black holes. Observe then that when taking the collapse of a reconfiguration sequence, its iteration does not increase.

► **Lemma 7.2** (Analogue of Lemma 6.3). *Let G be a (undirected) galactic graph with no neighboring black holes; the number and length of reconfiguration sequences with k tokens and of iteration ι over G is bounded by $g(|V(G)|, k, \iota)$ for some computable function g .*

Proof. First observe that with the absence of neighboring black holes, each step introduces a change for a planet (losing or receiving a token). As, a planet only witness $2k\iota$ many changes, this bounds the length of every sequence by $2k\iota n$, where $n = |V(G)|$.

As there are n^k possible configurations, the total number of possible reconfiguration sequences of iteration ι is at most $(n^k)^{2k\iota n} = n^{2k^2\iota n}$. ◀

We then call the ι -profile of x the set of reconfiguration α of iteration ι over $\text{warp}(x)$ such that there is a reconfiguration β over $\text{significant}(x)$ that collapses to α over $\text{warp}(x)$.

► **Proposition 7.3** (Analogue of Proposition 6.6). *Given a tree decomposition of a graph G of width tw , a node x of the decomposition, and the ι -profile of its children y_1 and y_2 , we can compute the ι -profile of x in time $f(k, \text{tw}, \iota)$.*

Which in turn yields the proof for Theorem 1.5.

8 Conclusion

We used galactic graphs, to show that ISR-DTS in DAGs is tractable, when parameterized by treewidth and the number of tokens k . Otherwise the problem becomes hard, unless either the depth is at most 2, or at most 3 and k small. Noticeable, the treewidth result applies to undirected graphs, if the iteration ι is restricted.

Thus, it might be interesting, to study this parameters behavior in graph classes other than DAGs. Interesting further results might also arise from applying the used techniques to other problems (Dominating Set Reconfiguration might be a good candidate) or to generalize to other directed graph classes that resemble DAGs.

References

- 1 Niranka Banerjee, Christian Engels, and Duc A. Hoang. Directed token sliding, 2024. doi:10.48550/arXiv.2411.16149.
- 2 Valentin Bartier, Nicolas Bousquet, Jihad Hanna, Amer E. Mouawad, and Sebastian Siebertz. Token sliding on graphs of girth five. *Algorithmica*, 86(2):638–655, 2024. doi:10.1007/S00453-023-01181-5.
- 3 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *J. Comput. Syst. Sci.*, 136:220–248, 2023. doi:10.1016/J.JCSS.2023.03.008.
- 4 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. In *Intl. Symp. on Parameterized and Exact Computation (IPEC)*, volume 214 of *LIPICs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.9.
- 5 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2017. doi:10.1007/978-3-319-68705-6_10.
- 6 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of reconfiguration problems. *Comput. Sci. Rev.*, 53:100663, 2024. doi:10.1016/J.COSREV.2024.100663.
- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 8 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Polynomial-time algorithm for sliding tokens on trees. In *Intl. Symp. on Algorithms and Computation (ISAAC)*, volume 8889 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 2014. doi:10.1007/978-3-319-13075-0_31.
- 9 Tatsuya Gima, Takehiro Ito, Yasuaki Kobayashi, and Yota Otachi. Algorithmic meta-theorems for combinatorial reconfiguration revisited. *Algorithmica*, 86(11):3395–3424, 2024. doi:10.1007/S00453-024-01261-0.
- 10 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.*, 651:37–49, 2016. doi:10.1016/J.TCS.2016.08.016.
- 11 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, Masahiro Takahashi, and Kunihiro Wasa. Independent set reconfiguration on directed graphs. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 241 of *LIPICs*, pages 58:1–58:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.MFCS.2022.58.
- 12 Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. URL: <http://www.jstor.org/stable/2369492>.

- 13 Daniel Lokshantov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.
- 14 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/S00453-016-0159-2.
- 15 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In *Intl. Symp. on Parameterized and Exact Computation (IPEC)*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2014. doi:10.1007/978-3-319-13524-3_21.
- 16 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.
- 17 Sebastian Siebertz. Reconfiguration on nowhere dense graph classes. *Electron. J. Comb.*, 25(3):3, 2018. doi:10.37236/7458.
- 18 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth, 2018. doi:10.1016/J.JCSS.2017.11.003.